Desarrollo de librerías y herramientas de software para programación multihilos, multiprocesos, escalonamiento, comunicación de interprocesos y protocolos de red en lenguaje C++

Rodrigo José García Ambrosy

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Desarrollo de librerías y herramientas de software para programación multihilos, multiprocesos, escalonamiento, comunicación de interprocesos y protocolos de red en lenguaje C++

Trabajo de graduación presentado por Rodrigo José García Ambrosy
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2023

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

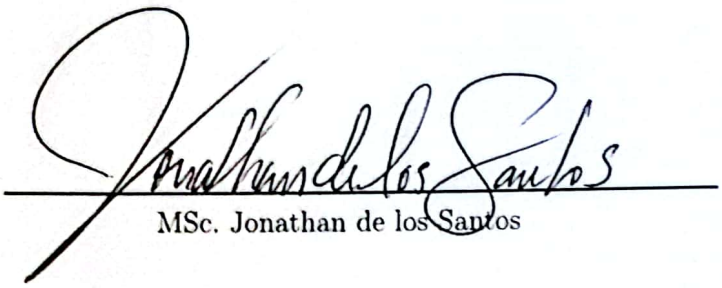
Desarrollo de librerías y herramientas de software para programación multihilos, multiprocesos, escalonamiento, comunicación de interprocesos y protocolos de red en lenguaje C++

Trabajo de graduación presentado por Rodrigo José García Ambrosy
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

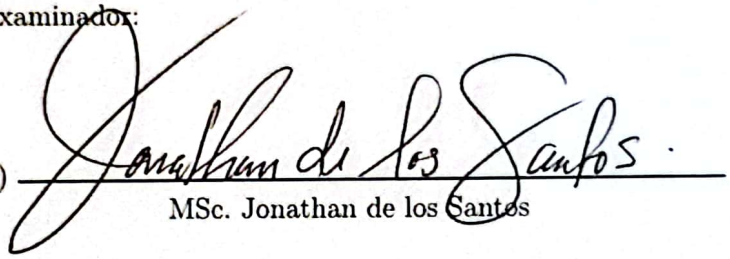
2023

Vo.Bo.:



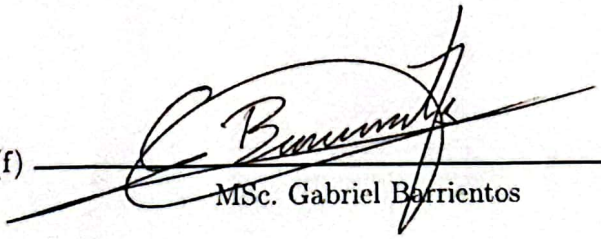
(f) _____
MSc. Jonathan de los Santos

Tribunal Examinador:



(f) _____
MSc. Jonathan de los Santos

(f) _____
MSc. José Morales y Morales



(f) _____
MSc. Gabriel Barrientos

Fecha de aprobación: Guatemala, 6 de enero de 2024.

Prefacio

Agradezco a la Universidad del Valle de Guatemala por las distintas oportunidades de crecimiento personal y profesional que me ha brindado, así como por los de maestros que me han guiado hasta este punto.

Le agradezco mucho a mis padres, quienes han sido mi apoyo en todos estos años y quienes han permitido mi crecimiento profesional y personal; también mi hermana, por ese apoyo incondicional. Le quiero agradecer especialmente a mis amigos de la universidad, ya que me han acompañado en este camino.

Índice

Lista de figuras

Lista de cuadros

Resumen

C++ es un lenguaje de programación muy amplio con el que se pueden realizar distintos programas. Por eso, se trabajó principalmente con la Raspberry Pi 3 modelo B y Raspberry Pi 4 para demostrar las capacidades del lenguaje en cuanto el uso de multihilos, multiprocesos, protocolos de red y algoritmos de escalonamiento.

Para la demostración de dichas capacidades, se usaron los laboratorios del curso Digital 3 con el fin de abarcar cada tema como un trabajo en específico para el análisis y desarrollo de estas herramientas de software. Se adaptaron estos laboratorios, hechos en el lenguaje de programación C, y todo el trabajo fue realizado en el sistema operativo de Raspbian (el sistema operativo de Linux diseñado para las Raspberry Pi).

Al realizar la traducción de C a C++ para demostrar las capacidades mencionadas, se encontró que sí es posible, incluso algunas capacidades son más sencillas de implementar en C++ sin la necesidad de utilizar muchas librerías, por ejemplo: las librerías *thread*, *chrono* y *iostream* simplifican el desarrollo de algunos trabajos, además de que implican un ahorro al ser librerías propias del sistema operativo de Linux. Otra ventaja en C++ es el uso de clases en algunos programas, ya que facilitó los programas de C que necesitan trabajar con librerías estáticas. Estos resultados fueron verificados al momento de traducir los laboratorios 3, 5, 6, 7 y 8. Esto permitió analizar detalladamente ciertas herramientas y librerías específicas para un mejor entendimiento. Además, la traducción del proyecto proporciona una visión más clara de cómo estas herramientas se comportan en un sistema SCADA programado en lenguaje C++.

Las limitantes que se encontraron al trabajar con el lenguaje de programación C++ fue las distintas versiones existentes, ya que se tiene que tomar en cuenta con qué versión se está trabajando para utilizar ciertas librerías estándar específicas. Entre más nueva la versión, se tiene un mayor repertorio de librerías. Al momento de encontrar librerías que no se pueden implementar en la versión instalada del sistema, se optó por tener en cuenta las librerías estándar de C, ya que C++ permite seguir utilizándolas a pesar de ser otro lenguaje de programación.

Abstract

C++ is a very broad programming language that can be used to create different programs. Therefore, we worked mainly with the Raspberry Pi 3 model B and Raspberry Pi 4 to demonstrate the capabilities of the language in terms of the use of multithreading, multiprocessing, network protocols and scaling algorithms.

For the demonstration of these capabilities, the labs of the Digital 3 course were used in order to cover each topic as a specific work for the analysis and development of these software tools. These labs, done in the C programming language, were adapted and all the work was done on the Raspbian operating system (the Linux operating system designed for Raspberry Pi).

When translating from C to C++ to demonstrate the mentioned capabilities, it was found that it is possible, even some capabilities are easier to implement in C++ without the need to use many libraries, for example: the libraries *thread*, *chrono* and *iostream* simplify the development of some works, besides that they imply a saving because they are libraries of the Linux operating system. Another advantage in C++ is the use of classes in some programs, since it facilitated the C programs that need to work with static libraries. These results were verified when translating labs 3, 5, 6, 7 and 8. This allowed a detailed analysis of certain tools and specific libraries for a better understanding. In addition, the translation of the project provides a clearer view of how these tools behave in a SCADA system programmed in C++ language.

The limitations encountered when working with the C++ programming language were the different existing versions, since one has to take into account which version one is working with in order to use certain specific standard libraries. The newer the version, the larger the repertoire of libraries. When finding libraries that cannot be implemented in the installed version of the system, it was decided to take into account the standard C libraries, since C++ allows to continue using them even though it is another programming language.

CAPÍTULO 1

Introducción

En un mundo de programación moderna, el desarrollo de aplicaciones se ha vuelto cada vez más complejo. Por lo que es curcial la grestión de múltiples procesos y la comunicación entre ellos, así como la implementación de red para facilitar la interacción entre sistemas. En estos escenarios, emergen los lenguajes de programación como C y C++, sirviendo como herramientas poderosas para cumplir con este tipo de retos. Entre ambos lenguajes de programación, el que tiene mejor capacidad para este tipo de tareas es C++ debido a su capacidad de brindar control de bajo nivel y abstracciones de alto nivel de manera simultánea.

Este trabajo de graduación tiene como objetivo enfrentar los desafíos en el desarrollo de aplicaciones modernas y eficientes. La programación multihilos y multiprocesos es uno de estos en cuanto se refiere a la capacidad de una aplicación para dividir su ejecución en unidades independientes que pueden ejecutarse en paralelo, incluyendo el escalonamiento (planificación y gestión de la ejecución de estos hilos o procesos para lograr optimizar los recursos de hardware disponibles). Otro tema crucial es cuando las distintas partes de una aplicación necesitan compartir información entre ellas, conocido como comunicación de interprocesos. Esta puede lograrse a través de mecanismos como colas y memoria compartida. Al implementar protocolos de red, permite que aplicaciones en diferentes sistemas puedan intercambiar datos y se comuniquen de manera fiable a través de redes.

En este contexto, las características del lenguaje de programación C++ permiten a los desarrolladores crear librerías y herramientas que simplifican y optimizan el desarrollo de sistemas complejos, además de facilitar el desarrollo de aplicaciones sencillas. Por lo tanto en este trabajo de graduación se espera evaluar, analizar y desarrollar librerías y herramientas de software para la programación de multihilos, multiprocesos, escalonamiento, comunicación de interprocesos, protocolos de comunicación y de red.

CAPÍTULO 2

Antecedentes

Curso de Electrónica Digital 3

El uso práctico de los sistemas embebidos tanto en *software* como en *hardware*, ha sido utilizado para el desarrollo de estos al momento de que estos abarquen temas relacionados al *kernel* y los sistemas operativos. La idea de utilizar los sistemas embebidos es lograr desarrollar un pensamiento lógico y analítico al momento de trabajar con procesos múltiples, comunicación y sincronización entre procesos, interfaces con *hardware* y dispositivos periféricos, sensores, *drivers*, y comunicación de red. Todo este tipo de objetivos se han puesto en práctica utilizando Raspberry Pi y computadoras que utilizan el sistema operativo CentOS en el curso de Digital 3 de la Universidad del Valle de Guatemala. Dicho curso existe desde el año 2018 supervisado por el Dr. Luis Alberto Rivera, con el objetivo de implementar y desarrollar sistemas que utilicen hilos, *semaphores*, multiprocesos y promover el uso de intercambio de datos en redes LAN.

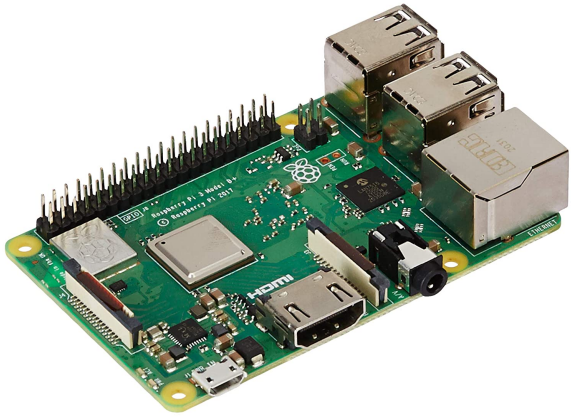


Figura 1: Raspberry Pi 3

1. **Interconexión de Raspberry Pi**

   La interconexión de Raspberry Pi es una práctica que se ha realizado anteriormente en el curso de Digital 3 en los laboratorios de este, con la intención de comprender el funcionamiento del uso de *sockets* para establecer comunicación y sincronización de procesos en ambientes distribuidos.

2. **La manipulación de prioridades en la Raspberry Pi**

   El manejo de prioridades en una Raspberry Pi es otro tipo de práctica que tiene como objetivo sincronizar tareas, tal como se realiza en los laboratorios de Digital 3, la idea es encontrar problemas, limitaciones y algún tipo de efecto que pueda surgir al momento de utilizar distintas prioridades en los hilos que se programen. Es posible que mediante las prioridades se puedan encontrar distintos algoritmos de escalonamiento.

Diseño e implementación de capacidades automáticas de navegación para un robot explorador modular.

Durante la implementación de navegación para el robot explorador se utilizó un piloto automático conocido como *PixHawk* y la configuración de un programa llamado *Mission Planner*, en este trabajo de tesis fue necesario el uso adicional de una Raspberry Pi 3 que funcionaba específicamente para el manejo y envío de datos a través de Internet. Para este trabajo se utilizaron los puertos seriales de la Raspberry para establecer la comunicación de datos que luego se estaría transmitiendo mediante Internet, esto se puede observar en la Figura 2. [1] Parte de los paralelos de este trabajo con lo que se ha realizado en el curso de Digital 3 es el uso de comunicación entre Raspberry Pi y una computadora o un sistema que obtenga información que intercambia con la Raspberry Pi.

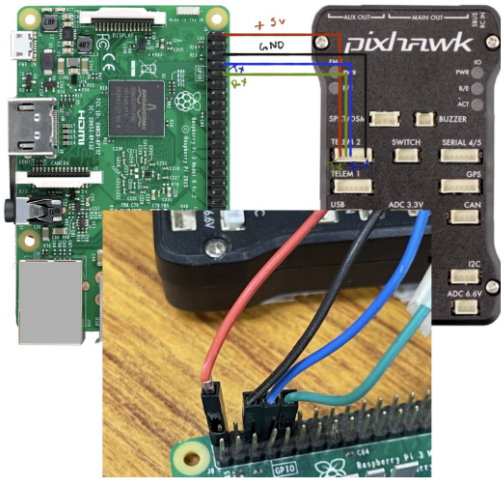


Figura 2: Conexión entre *Pixhawk* y Raspberry, y conexión física

Desarrollo y uso de código C++ en la Raspberry Pi para el control de dispositivos equipados con RS-232

En el trabajo *Controlling RS-232 equipped devices using Raspberry Pi and C++* un trabajo de tesis realizado en la Universidad de Ciencias Aplicadas de Metropolia, se desarrolló un prototipo para controlar dispositivos que todavía utilicen comunicación por RS-232. En el trabajo que realizaron fue necesario utilizar un dispositivo de respuesta controlable para establecer la comunicación serial programada en C++ con la Raspberry Pi con el fin de mantener el intercambio de datos entre los dispositivos. [2] Parte del curso de Digital 3 es entender cómo aplicar los distintos métodos de intercambio de datos y como se utilizan con la Raspberry Pi. Por lo que este trabajo es un ejemplo de lo que se espera luego de comprender este tipo de aplicaciones.

CAPÍTULO 3

Justificación

El propósito de este trabajo es desarrollar herramientas que puedan ser utilizadas en diversas aplicaciones de forma amplia, más allá del curso específico de Digital 3, con el fin de este trabajo es proporcionar oportunidades de aplicación en el área de la programación orientada a objetos. Al trabajar en C++ se tiene una gran versatilidad del trabajo en *software* y *hardware* haciendo posible proponer soluciones en proyectos que involucren sistemas embebidos que trabajen en aplicaciones relacionadas a los procesos de multihilos, multiprocesos, escalonamiento, protocolos de red y comunicación interprocesos, dentro del lenguaje de programación C++.

Una de las principales ventajas de este trabajo es que no se enfoca exclusivamente en el uso de la Raspberry Pi, ya que al trabajar con el lenguaje de programación C++ se tiene la flexibilidad de trabajar con distintos dispositivos por lo que se le da un enfoque a la programación orientada a objetos. Adicionalmente a esto se tiene una estructura más formal al trabajar con este lenguaje que permite el uso de códigos mejor estructurados, lo que facilita su reutilización en futuros proyectos.

Al momento de trabajar en el lenguaje de programación C++, los conceptos de multihilos, multiprocesos y *semaphores* son aplicaciones fundamentales que se manejan en este trabajo. Como parte del trabajo es demostrar la eficiencia y rendimiento que el lenguaje de programación C++ puede ofrecer al momento de realizar trabajos con este tipo de herramientas. Parte de demostrar esto es comparar este lenguaje de programación con el lenguaje de programación C. Además, al manejar de manera adecuada estos conceptos puede resultar esencial para el desarrollo de aplicaciones robustas y escalables en el campo de la programación orientada a objetos, como se mencionaba anteriormente. En este sentido, el conocimiento y la aplicación efectiva de multihilos, multiprocesos y *semaphores* son habilidades que se valoran bastante en el área laboral.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Desarrollar librerías y herramientas de software en el lenguaje de programación C++ para programación multihilos, multiprocesos, escalonamiento, comunicación interprocesos, protocolos de comunicación y de red.

4.2. Objetivos específicos

- Evaluar y analizar las librerías disponibles en C++ que permitan la implementación de programación multihilos y multiprocesos.
- Evaluar y analizar las librerías disponibles en C++ que permitan la implementación de algoritmos de escalonamiento, sincronización y comunicación interprocesos.
- Realizar una evaluación de la librería *wiringPi* en C++ para el manejo de puertos de entrada/salida y los módulos de comunicación I^2C y SPI.
- Evaluar y analizar las librerías disponibles en C++ que permitan la comunicación entre dispositivos en una red.
- Recrear y documentar los ejemplos de clase, laboratorios y proyecto del curso de Digital 3 en el lenguaje de programación de C++.
- Crear un repositorio debidamente documentado para las librerías y herramientas de software desarrolladas.

CAPÍTULO 5

Alcance

Existe un amplio conocimiento del lenguaje de programación C que ha logrado ahondar en sus distintas formas de aplicación. Esta base permitió que el desarrollo, análisis e implementación de librerías utilizados en el lenguaje C++ se enfoque principalmente en facilitar la programación de sistemas multihilo y multiproceso, la gestión de tareas mediante algoritmos de escalonamiento, la comunicación interprocesos y los protocolos de red. La elección por utilizar el lenguaje C++ se justifica por su capacidad para generar códigos más concisos en comparación con el lenguaje C, así como por su aprovechamiento de la programación orientada a objetos mediante el uso de clases, facilitando los procesos.

Este trabajo establece las bases de uso que demuestran las capacidades tanto del lenguaje como de su funcionamiento en la Raspberry Pi 3B, siendo de gran utilidad en el punto de inicio para futuros proyectos y desarrollos relacionados a la programación de multihulos, multiprocesos, escalonamiento, protocolos de red, entre otros. Se pone especial énfasis en la facilitación de programas, dedicando atención específica a cada herramienta utilizada en C++. Esta estrategia permitirá unificar los conocimientos adquiridos, promoviendo un mayor entendimiento al trabajar con librerías o herramientas específicas.

Para establecer estos parámetros, es necesario traducir los laboratorios tres, cuatro, cinco, seis, siete y ocho. Cada uno de estos trabajos muestra el funcionamiento de las herramientas y librerías utilizadas en la programación de sistemas multihilo y multiproceso, la manipulación de las entradas y salidas de la Raspberry Pi 3B la gestión de tareas mediante algoritmos de escalonamiento, la comunicación interprocesos y los protocolos de red. Al proporcionar la traducción de estos laboratorios; y tomando en cuenta el logro de implementar las herramientas en el proyecto, el alcance de este trabajo permite demostrar cómo funcionan en el lenguaje de C++ y brindar un mejor proceso de desarrollo y utilidad. Además, utilizar estas herramientas con C++ brindaría un mejor entendimiento de cómo utilizar estas funciones unificadas al momento de trabajar con un sistema SCADA generando un mejor uso del tiempo, eficiencia y facilitando su aplicación en un contexto general.

CAPÍTULO 6

Marco teórico

Lenguaje de programación C y C++

El lenguaje de programación C fue utilizado en el sistema operativo Unix originalmente, el cual tiempo después fue adoptado por el estándar ANSI en 1989. Este estándar de 1989 funcionó como base para el estándar de C++. El lenguaje de programación C por lo general es llamado como un lenguaje informático de nivel medio ya que combina los mejores elementos de los lenguajes de alto nivel con el control y la flexibilidad del lenguaje Assembler [3]. El lenguaje de C++ mantiene las mismas ventajas que C con respecto a la flexibilidad y eficiencia, logrando así eliminar ciertas dificultades o limitaciones que tiene C. Uno de los puntos importantes de C++ es la manera en cómo se declaran las funciones.

El lenguaje de C++ mantiene una compatibilidad casi completa con C, ya que trata con algunas modificaciones que solo facilitan el uso del lenguaje y no cambian mucho la naturaleza del lenguaje. Para evitar mezclar los archivos de los dos lenguajes es importante diferenciar un archivo de C de uno de C++ dependiendo de la extensión del nombre de los ficheros, para C sabemos que se utiliza **.c* y en el caso de C++ se usa la extensión de **.cpp* (*C plus plus*, esa es la forma de llamar al lenguaje en inglés). Diferenciar la extensión del archivo determina que compilador utilizar si el de C o el de C++, si se utilizan de manera incorrecta se pueden tener problemas al momento de realizar la compilación del programa. A parte de tener en mente la extensión del archivo, también es necesario aprender a utilizar la sintaxis específica para cada lenguaje, un ejemplo sencillo de esto es realizando un código que imprima en la terminal "*Hello World!*".[4][5][6]

Código C:

```c
#include <stdio.h>

int main() {
  printf("Hello World!");
  return 0;
}
```

Código C++:

```cpp
#include <iostream>
using namespace std;

int main() {
  cout << "Hello World!";
  return 0;
}
```

En ambos códigos se tienen dos librerías distintas que permiten el uso de objetos de entrada y salida, en el caso de C al usar *stdio.h* permite el uso de *printf()* y en el caso de C++ se utiliza *iostream* el cual utiliza *cout* para lograr imprimir en la terminal el mensaje de *Hello World!*. A pesar de que C++ permite el uso de *printf()* en este caso al usar *cout* no es necesario especificar el tipo de dato que va a ser impreso, ya que se asocia con un formato determinado.[7]

En la línea 2 del código de C++ nos indica que podemos utilizar nombres de objetos y variables de la librería estándar. Si en caso se omite esta línea el código en C++ sería de la siguiente forma:

```cpp
#include <iostream>

int main() {
  std::cout << "Hello World!";
  return 0;
}
```

Al omitir la línea "*using namesoace std*", la sintaxis del código ahora tiene que utilizar la palabra clave "*std*" seguida del operador "::" para ciertos objetos. Es importante tomar en cuenta este tipo de aspectos al momento de trabajar con el lenguaje de programación de C++.

Librerías estándar del lenguaje de programación C++

En el lenguaje de programación C++, las librerías se utilizan de igual manera que en C, las cuales permiten el uso de distintas aplicaciones para uso general. Dependiendo de el

programa que se diseñe, se tienen que especificar las librerías a utilizar al principio para que funcionen y así, al momento de programar, el compilador reconozca sus respectivas funciones. Algunas de las librerías estándar utilizadas son las siguientes:

- ***iostream***: forma parte de la biblioteca de entrada/salida basada en flujos.
- ***chrono***: utiliza utilidades relacionadas al tiempo.
- ***thread***: permite que los programas ejecuten múltiples funciones de manera simultánea.
- ***cstdio***: proporciona soporte genérico para operaciones de archivo y proporciona funciones con capacidades limitadas a entradas/salidas de caracteres.
- ***cstdlib***: proporciona utilidades varias como funciones para gestionar la finalización del programa y la limpieza de recursos.
- ***ctime***: es utilizada para la manipulación de tiempo.
- ***fstream***: implementa operaciones de entrada/salida de flujo de archivos de alto nivel.
- ***string***: se encarga de la manipulación y almacenamiento de secuencias de caracteres. La creación, manipulación y destrucción de cadenas de caracteres se realiza mediante un cómodo conjunto de métodos de clase y funciones relacionadas.
- ***vector***: es una librería de contenedores de secuencias que representan matrices que pueden cambiar de tamaño.
- ***cerrno***: es una librería para el tratamiento de errores.
- ***semaphore***: gestiona las estructuras de datos en una cola y un contador.
- ***sstream***: implementa operaciones de entrada/salida de cadenas de alto nivel.

Además de las librerías estándar de C++, se pueden utilizar también las de C, ya que C++tiene compatibilidad con estas. Además, en Linux, se tienen librerías propias del sistema como las siguientes:

- unistd.h
- sched.h
- fcntl.h
- timerfd.h
- types.h
- getopt.h
- mman.h

WiringPi

WiringPi es una librería de acceso GPIO basada en pines escrita en C para los dispositivos SoC BCM2835, BCM2836 y BCM2837 utilizados en todas las versiones de Raspberry Pi. Por lo que su uso es diseñado específicamente para las Raspberry Pi que utilizan Raspbian de 32 bits[8]. Entre las funcionalidades de la librería se encuentran[9]:

- Utilidad gpio para programar y configurar los pines GPIO.
- Admite la lectura y escritura analógica.
- Usa la librería *devlib* que permite acceso fácil a algunos periféricos populares.
- Funciones de temporización.
- Funciones de prioridad e hilos.
- SPI, I^2C y Serial.
- Librería PWM software.
- Compatibilidad con *Gertboard*

Concepto de sincronización de procesos

El concepto de la sincronización de procesos es ejecutar múltiples programas de manera simultáneamente. Sabemos que un proceso es un programa que se encuentra funcionando, el cual se considera como una unidad de trabajo en un sistema moderno de tiempo compartido. En los sistemas operativos, los procesos que se ejecutan de forma concurrente son procesos independientes, estos se consideran de esta forma si no pueden afectar a otros procesos que se estén ejecutando en el sistema. Existe el caso de los procesos cooperantes que son aquellos que comparten datos con otros procesos. Lo cual es preferible utilizar un ambiente de trabajo que permita la cooperación entre procesos ya que esto nos permite el intercambio de información, modularidad, comodidad para que múltiples trabajos estén funcionando al mismo tiempo, de manera en que nos permita utilizar el mecanismo de comunicación entre procesos.[10]

Considerando que un sistema operativo está formado por n cantidad de procesos, estos tienen un segmento de código que se le conoce como sección crítica, en donde el proceso se puede encontrar intercambiando variables, usando alguna tabla, etc. Cuando un proceso se encuentra en la sección crítica, este no permite que ningún otro proceso se ejecute en la misma sección. Por lo tanto, dos procesos no pueden ejecutar sus secciones críticas al mismo tiempo y estos tienen que solicitar permiso para entrar en su sección crítica. Una solución para el problema de las secciones criticas es el uso de una herramienta de sincronización conocida como *semaphore*.

Semaphore

Los *semaphores* son un objeto con un valor entero que se puede manipular con dos rutinas; en el estándar POSIX, estas rutinas son *sem wait()* y *sem post()*. Antes de que pueda funcionar es necesario inicializar el *semaphore* con algún valor.[11]

Los *semaphores* pueden ser de dos tipos: los de conteo y los binarios. Los binarios son conocidos como bloqueos *mutex* en algunos sistemas y se les consideran bloqueos ya que proporcionan exclusión mutua[10]. Al implementar los *semaphores* la mayor desventaja que tienen es su requisito de espera ocupada. Es decir, cuando un proceso está en su sección crítica, y al mismo tiempo si otro proceso está intentando entrar en su sección crítica, este segundo proceso entra en un bucle continuo en el código de entrada. Estos bucles pueden ser un problema en el sistema ya que pueden desperdiciar ciclos del CPU que pueden ser aprovechados por otros procesos. Existen dos tipos de situaciones que se pueden presentar al trabajar con *semaphores* en caso de que se implementen con una cola de espera, los cuáles serían *deadlock* y *starvation*. Un *deadlock* se genera cuando dos o más procesos están esperando de manera indefinida para que ocurra algún evento. En el caso de *starvation* es cuando un proceso espera de manera indefinida dentro del *semaphore*.

Multihilos

Muchos sistemas embebidos están pensados para realizar una única función principal. Las operaciones que debe de realizar esa función están todas interrelacionadas. Por lo que la naturaleza de una aplicación que se ejecuta como una única función primaria sugiere que el proceso asociado debería descomponerse en una serie de subtareas que se ejecuten en paralelo. En el tiempo de ejecución, el proceso puede pasar la CPU a cada una de estas subtareas, permitiendo así que cada una haga su trabajo.

Tomando en cuenta que cada uno de los trabajos más pequeños tiene su propio hilo de ejecución, se le denomina diseño de proceso único-multihilo. A diferencia de los procesos o tareas, los hilos son independientes entre sí. Estos pueden acceder a cualquier dirección dentro del proceso. Esto es importante tomarlo en cuenta ya que un cambio de contexto entre hilos puede ser mucho más simple y rápido que entre procesos. Hay que tomar en cuenta que, al estar cambiando entre hilos, se guarda y se restaura mucha menos información.[12]

Algoritmos de escalonamiento

En un sistema que realiza multitareas, el objetivo principal es que algún proceso utilice el CPU en todo momento. Este esquema maximiza el uso de ese recurso. Es responsabilidad del planificador asegurarse de que el CPU se utilice de la manera más eficiente posible, de tal manera en que las tareas se ejecuten en un orden que tenga ciertas restricciones requeridas. Por eso al trabajar con algoritmos de escalonamiento, se deben de tener en cuenta las prioridades de las tareas. Esto lo asigna la persona que planifica que algoritmo utilizar basándose en varios criterios diferentes. Estos criterios se usan para determinar qué tarea se ejecutará cuando haya bastantes esperando ser ejecutadas y que estas estén listas.[12]

Los algoritmos de escalonamiento deben de tomarse en cuenta al momento de diseñar el desarrollo del sistema, porque el algoritmo que se utilice puede optimizar y mejorar el rendimiento general del sistema. Si se tienen ciertos criterios a seguir, es necesario asegurarse de que las tareas y acciones puedan cumplir con lo requerido. Algunos algoritmos de escalonamiento son:

- ***Earliest Deadline First***

  El algoritmo *Earliest Deadline* utiliza un algoritmo dinámico en el cual se asigna prioridad a la tarea con el plazo más cercano. El programa se establece y se modifica durante su ejecución, ya que solo entonces se pueden evaluar los plazos. Es posible planificar de manera eficiente un conjunto de tareas si la carga de estas es menor al 100 por ciento. El algoritmo deja de ser estable si el tiempo de ejecución de las tareas lo supera, ya que esto hace que alguna tarea pierda su plazo.[12] [13]

  Para detallar un poco más el algoritmo es necesario comprender que consiste en cuatro pasos los cuales son: inicialización, priorización, programación y ejecución de tareas.

  El primer paso al momento de inicializar las tareas le asigna a cada una la fecha límite dependiendo de los requisitos de finalización. Luego se les asigna una prioridad a las tareas, para que de esta manera el sistema las ordene según su función de plazos. Además, el sistema selecciona la tarea con la fecha límite más temprana para su ejecución. En caso de tener tareas con la misma fecha límite, se selecciona la tarea con la prioridad más alta.

  Por último, la CPU ejecuta la tarea seleccionada hasta que se cumpla o si esta alcanza la fecha límite. Si esta tarea termina antes de la fecha límite, el sistema selecciona la siguiente tarea para ejecutar. En caso de que la tarea alcanza la fecha límite, la tarea se puede considerar perdida. Por lo que se tendría que volver al paso de priorización de tareas y repetir todos los pasos hasta que termine el algoritmo.[14]

- ***Least Laxity First***

  En el caso de *Least Laxity* se toma en cuenta el tiempo de ejecución de la tarea. Por lo que la prioridad de la tarea se basa en la siguiente relación:

  $$laxitud = plazo - tiempodeejecucion$$

  Este algoritmo puede utilizarse en sistemas que utilizan una mezcla de plazos duros y blandos, ya que se le puede dar prioridad a las tareas duras sobre las que tienen restricciones menos rígidas. Este algoritmo tiende a dedicar ciclos de CPU a tareas que claramente van a llegar tarde y, por lo tanto, provoca que más tareas incumplan con los plazos.[12]

- ***First-Come First-Served (FIFO)***

  Es uno de los algoritmos más simples que existen ya que gestiona de manera fácil con una cola de tiempo "primero en llegar, primero en ser atendido". Al momento de que un proceso entra en la cola de que está listo, el bloque de control de tareas se enlaza en la parte final de la cola. Es decir, cuando el CPU queda libre, se asigna al proceso que se encuentra al principio de la cola y el proceso que se está ejecutando actualmente se elimina de la cola. Uno de los problemas de este algoritmo es que puede

tener problemas al trabajar en un sistema que tenga restricciones de tiempo real.[12] [15]

La idea de FIFO es marcar cada tarea con una prioridad siguiendo una simple regla la cual es: programar el primer proceso en llegar, y dejar que se ejecute hasta su finalización. Por lo que es un algoritmo de programación no preferente, lo que significa que sólo puede ejecutarse un proceso a la vez. De manera independiente de si utiliza los recursos del sistema de forma eficaz, y también de si hay cola de otros procesos esperando. [15][16]

- ***Round Robin***

  El algoritmo de *Round Robin* es similar al algoritmo de FIFO, pero con la adición de preempción para alternar entre procesos. Se define una pequeña unidad de tiempo llamada "*time quantum*" o "*slice*", por lo que la cola de listos se trata como una cola circular. Se recorre esta cola y se le va asignando al CPU a cada proceso por un *slice*. Si un proceso se completa en menos tiempo del asignado el CPU se libera; de lo contrario, el proceso se interrumpe cuando se agota el tiempo y se coloca hasta el final de la cola. Cualquier nuevo proceso que se agregue entra al final de la cola.[12]

  En términos simples *Round Robin* es un algoritmo sencillo, para ejemplificar su funcionamiento digamos que se tienen 3 procesos (A,B,C). Lo que va a realizar el algoritmo es ejecutarlos en la secuencia A, B, C, A, B, C, A y así sucesivamente, hasta que todos los procesos hayan terminado.[15]

- ***Shortest Job First***

  Al usar el algoritmo de "*Shortest Job First*" o "el trabajo más corto primero", asume que el CPU está funcionando en ráfagas de actividad. A cada tarea en el sistema se le asigna una estimación de cuánto tiempo va a necesitar la tarea la próxima vez que se le asigne el CPU. Este algoritmo puede ser tanto preemptivo como no preemptivo. Al ser preemptivo, el proceso que se está ejecutando puede ser interrumpido por uno con un tiempo restante más corto para que este se complete.[12]

  Cabe mencionar que este algoritmo es una versión modificada de FIFO. Cuyo propósito principal es reducir el tiempo de espera promedio entre los procesos que están funcionando. Además, este tiende a tener las mismas debilidades que FIFO con respecto a la eficiencia en el uso de los recursos.[17]

Protocolo de comunicación de red TCP

TCP (*Transmission Control Protocol*), definido en RFC 793, 1122, 2018, 5681, y 7323. Es un protocolo de transporte orientado a la conexión porque antes de que un proceso de aplicación pueda comenzar a enviar datos a otro, los dos procesos deben de realizar un "*handshake*" entre sí, es decir, deben de enviarse algunos segmentos preliminares para establecer los parámetros necesarios para la transferencia de datos subsiguiente.[18]

Para comprender y establecer una conexión TCP es necesario que tanto el cliente como el servidor participe en lo que es conocido como un *three-way handshake* . El proceso es el siguiente[19]:

1. Un cliente le manda al servidor un paquete SYN. Es una solicitud de conexión desde su puerto de origen al puerto de destino del servidor.
2. El servidor responde con un paquete SYN/ACK, reconociendo la solicitud de la conexión.
3. El cliente recibe el paquete SYN/ACK y responde con un paquete ACK propio.

La conexión TCP no es un circuito TDM o FDM de extremo a extremo como en una red de conmutación de circuitos. En su lugar, la "*conexión*", es lógica con un estado común que reside únicamente en los TCP de los dos sistemas finales que se comunican. El protocolo TCP sólo se ejecuta en los sistemas finales y no en los elementos de red intermedios ya que estos no mantienen el estado de la conexión TCP. Las conexiones TCP proveen un servicio *full-duplex* el cual si un proceso se encuentra en un *host* y otro proceso se encuentra en un *host* diferente, la información puede fluir sin problemas de un *host* a otro en la capa y viceversa de aplicación. Además, las conexiones TCP siempre son "*point-to-point*", es decir, entre solo un receptor y un solo emisor. El "*multicasting*"no es posible con TCP.[18]

Las conexiones que se establecen en TCP descomponen los datos transmitidos en segmentos, cada uno de los cuales se empaqueta en un datagrama y se envía a su destino. Para contemplar un poco como se envuelven estos paquetes, TCP utiliza un *header* el cual contiene 10 campos obligatorios con un total de 20 bytes, los cuales son los siguientes[19]:

- *Source port*
- *Destination port*
- *Sequence number*
- *Acknowledgment numbet*
- *TCP data offset*
- *Reserved data*
- *Control flags*
- *Window size TCP checksum*
- *Urgent pointer*
- *mTCP optional data*

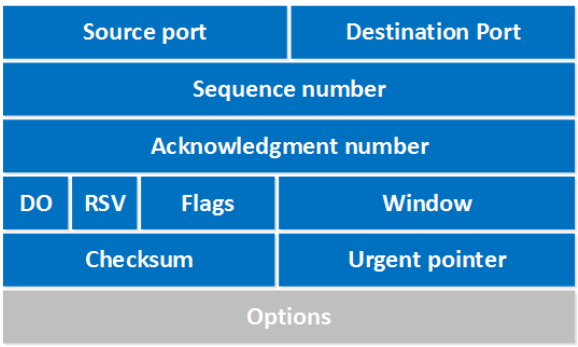


Figura 3: Campos de estructura de un *header* de TCP

Protocolo de comunicación de red UDP

UDP (*User Datagram Protocol*), definido en [RFC 768], realiza lo todo lo que puede hacer un protocolo de transporte. Aparte de las funciones de multiplexación/demultiplexación y alguna ligera comprobación de errores no añade nada a las IP. UDP toma los mensajes del proceso de aplicación, adjunta los campos de número de puerto de origen y destino para el servicio de multiplexación, añade otros dos pequeños campos y pasa el segmento resultante a la capa de red. La capa de red encapsula el segmento de la capa de transporte en un datagrama IP y, a continuación, hace intento por entregar el segmento al host receptor. UDP utiliza el número de puerto de destino para entregar los datos del segmento al proceso de aplicación correcto. Hay que tomar en cuenta que UDP no realiza un *handshaking* entre la capa de transporte emisora y receptora antes de enviar un segmento. Por esta razón, se dicce que UDP no tiene conexión[18]. Un ejemplo del uso de UDP es el protocolo DNS. Una de las ventajas de utilizar UDP es que una vez se procesa la información a UDP, esta mandara el paquete inmediatamente al segmento de la capa de red.

Para tener una descripción más detallada de UDP es necesario analizar sus ventajas y sus desventajas. Dependiendo de la aplicación que se le dé al protocolo de UDP se pueden tener las siguientes ventajas[20]:

- No tiene retrasos de retransmisión. Es decir, que UDP al momento de trabajar con aplicaciones sensibles al tiempo no pueden permitirse por retrasos de retransmisión de paquetes perdidos.
- La velocidad de UDP lo hace útil para protocolos de consulta-respuesta como DNS,
- Adecuado para transmisiones. La falta de comunicación extremo a extremo lo hace adecuado para transmisiones, en las que los paquetes de datos transmitidos se dirigen como recibidos por todos los dispositivos de internet.

Parte de las desventajas de utilizar UDP es que al momento de no contar con los requisitos de conexión y verificación de datos se pueden crear problemas con la transmisión de paquetes. Entre las desventajas de utilizar UDP se encuentran[20]:

- No se garantiza el orden de los paquetes.

- No se verifica la disponibilidad del ordenador que recibe el mensaje.
- No hay protección contra paquetes duplicados.
- No hay garantía de que el destino reciba todos los bytes transmitidos. Sin embargo, UDP proporciona una suma de comprobación para verificar la integridad de los paquetes individuales.

Para comprender como está compuesto UDP, es necesario conocer la estructura del *header* de los paquetes. Estos contienen cuatro campos con un total de ocho bytes. Los cuatro campos son los siguientes:

- *Source port*
- *Destination port*
- *Length*
- *Checksum*

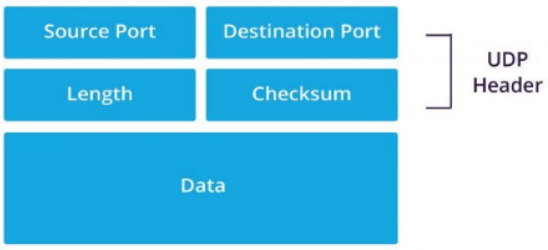


Figura 4: Campos de estructura de un *header* de UDP

Sistema operativo Linux

El sistema operativo Linux se desarrolló en 1991, el cual mantiene compatibilidad con UNIX. El primer *kernel* de Linux se desarrolló para tener compatibilidad con los procesadores de 32 bits de Intel. Además, el código fuente de Linux se puso a disposición de los usuarios de forma gratuita a través de su página oficial. Esto se le conoce como *open source* por lo que cualquiera puede usarlo sin restricciones de distribución. Con esto en mente las distintas distribuciones que existen incluyen funcionalidades extras como utilidades adicionales de instalación del sistema, paquetes precompilados y listos para instalar herramientas de UNIX que son habituales, navegadores de web y procesadores de texto y edición.[21]

Para comprender un poco mejor el funcionamiento del sistema operativo Linux es necesario comprender como funciona el *kernel*[22]. El *kernel* tiene 4 trabajos principales:

1. Gestión de la memoria
2. Gestión de procesos

3. Controladores de dispositivos
4. Llamadas al sistema y seguridad

Si el *kernel* es implementado de manera correcta, trabaja en lo que es conocido como el espacio del *kernel*, en donde se asegura de cumplir con cada uno de estos trabajos principales.

Linux es conocido por sus comandos que se utilizan en la terminal, ya que son esenciales para navegar y utilizar ciertas funciones del sistema operativo. Estos comandos siguen cuatro categorías las cuales son:

- *Shell builtins* : Los comandos integrados directamente en el *shell* con la ejecución más rápida.
- *Shell functions*: Scripts que se encuentran en el *shell*.
- *Aliases*: Atajos de comandos personalizados.
- *Executable programs*: Programas compilados e instalados.

Por último, tenemos las distribuciones de Linux, los cuales son sistemas operativos personalizados del sistema original. En este caso las distribuciones que nos interesan son las compatibles con la Raspberry Pi. Incluso la misma empresa que desarrollo la Raspberry Pi, crearon su propia distribución llamada Raspberry OS o Raspbian[23]. Entre las otras distribuciones compatibles se encuentran[24]:

- TwisterOS
- DietPi
- Arch Linx ARM
- Ubuntu
- Ubuntu MATE
- HamPi
- Gentoo
- Manjaro
- PiCore
- Debian
- Fedora
- Void

Algunos de los comandos básicos y más utilizados en Linux son los siguientes [25]:

Comando	**Función**	**Sintaxis en la terminal**
pwd	Imprime ubicación actual	pwd
ls	Lista de contenido del directorio actual	ls
cd	Cambia el directorio actual	cd <directory>
cat	Imprime contenido de archivo	cat <path>/<filename>\ cat <file 1><file 2>
touch	Modificar la fecha y hora de un archivo existente	touch <filename>
cp	Copiar archivos y directorios	cp <source file><target file>
mv	Mover un archivo de directorio	mv <filename>∼/Documents/<filename>
mkdir	Crear nuevo directorio	mkdir <directory name>
rmdir	Elimina directorios	rmdir <directory name>
locate	Buscar archivos	locate <filename>
find	Buscar archivos en todo el sistema	find -name <file or directory>
grep	Busca en el texto de un archivo	grep <search string><filename>
sudo	Habilita permisos de administrador en el comando	sudo <command>
df	Estadisticas del disco de almacenamiento del sistema	df
du	Muestra el espacio de un archivo	du
head	Truncar salidas largas	head <filename>
tail	Muestra las ultimas 10 lineas de un archivo	tail <filename>
diff	Compara dos archivos diferentes e imprime sus contenidos	diff <file 1><file 2>
tar	Archiva, comprime y extrae ficheros archivados	tar <filename>
chmod	Cambia los permisos del archivo o directorio	chmod <permission><file or directory>
chown	Cambia la propiedad de un archivo o directorio	sudo chown <new owner name or UID><file or directory>
ps	Imprime los procesos funcionando en el sistema	ps
top	Versión extendida de ps	top
kill	Termina un proceso en el sistema	kill <signal option><process ID>
ping	Comprueba la conectividad a internet	ping google.com
wget	Descarga archivos de internet	wget <URL>
history	Muestra el registro historico de comandos	history
man	Despliega el manual de un comando en la terminal	man <command>
echo	Imprime argumentos en la terminal	echo <argument>
useradd	Crea un nuevo usuario al sistema de Linux	sudo useradd <username>

Cuadro 1: Comandos de Linux

Qt framework

Qt es un marco de desarrollo multiplataforma para aplicaciones de escritorio y dispositivos integrados, escrito en C++. Aunque no es un lenguaje de programación, permite crear aplicaciones y bibliotecas que pueden compilarse con cualquier compilador de C++ compatible con estándares como GCC y MinGW.

Incluye su propio entorno de desarrollo integrado (IDE), llamado Qt Creator, que es compatible con sistemas operativos como Linux y Windows. Este IDE ofrece características como completado inteligente de código, resaltado de sintaxis, sistema de ayuda, depurador e integración con los principales sistemas de control de versiones, como git [26][27].

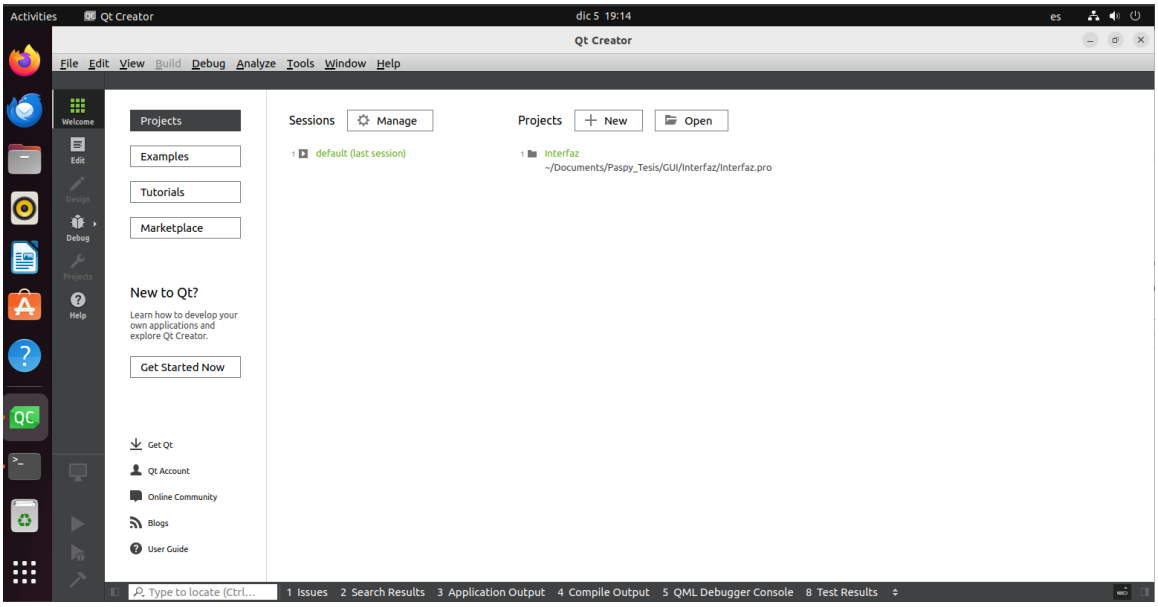


Figura 5: Adaptada de Qt Project(2023)

CAPÍTULO 7

Traducción de laboratorios de lenguaje C a C++

Los laboratorios núm.1, núm.2 y núm.4 no se tomaron en cuenta ya que estos son laboratorios para familiarizarse con el uso del sistema operativo de Linux, funciones de la Raspberry Pi en la terminal y guías de instalación de IDE. Las guías de los siguientes laboratorios se encuentran en anexos. Todos estos trabajos se realizaron en una Raspberry Pi 3 y se hicieron pruebas también para verificar si funcionaban de igual manera en una Raspberry Pi 4, lo cual sí funcionó. En los anexos, se proporciona un enlace que lleva a un repositorio en Github donde se encuentran todos los códigos utilizados en estos trabajos.

7.1. Laboratorio núm. 3

El laboratorio núm. 3 tiene como objetivo el crear y ejecutar procesos con uno o más hilos de ejecución, observar las similitudes y diferencias entre los procesos y los hilos. El uso de *pthreads* en el lenguaje de C++ la cual utiliza la librería threads, la función fork() y comprender los conceptos relacionados a la Condición de Carrera.

La primera parte del laboratorio consiste en analizar los múltiples procesos y los múltiples hilos para entender cómo funcionan varios procesos/hilos de manera simultánea. Los primeros programas que se tradujeron a C++ fueron L3_Hello.c y L3_World.c, la traducción a C++ se realizó de manera simple al ser dos programas sencillos. Los cuales utilizaron las librerías de chrono, thread y iostream. Luego fue necesario analizar los programas L3_-Hilos_Ej1.c y L3_Hilos_Ej2.c para comprender el concepto de los hilos.

Los programas ahora tienen la terminación .cpp ya que se tradujeron a C++, luego de eso lo primero que se cambió fueron las librerías correspondientes a C++ para cumplir con los mismos objetivos y cumplir con los objetivos del laboratorio. La principal diferencia es la sintaxis que se usa en C++, el siguiente código es el resultado de traducir los códigos de

L3_Hello y L3_World.

```cpp
/*
 ================================================================================
 Nombre: L3_Hello.cpp
 Autor:  Rodrigo José García Ambrosy
 ================================================================================
 */

#include <iostream> //Librería estándar de entrada/salida
#include <thread>   //Librería para utilizar hilos
#include <chrono>   //Librería para manejo de tiempos
int main()
{
    while (1)
    {
        std::cout << "Hello ";
        std::cout.flush(); //Limpieza de buffers asociados al estándar output
        std::this_thread::sleep_for(std::chrono::microseconds(1100000));
        //Configuración para que ese hilo haga una pausa durante 1.1 segundos
    }
}
```

Cuadro 2: Código Lab3_Hello.cpp

```cpp
/*
 ================================================================================
 Nombre: L3_World.cpp
 Autor:  Rodrigo José García Ambrosy
 ================================================================================
 */

#include <iostream> //Librería estándar de entrada/salida
#include <thread>   //Librería para utilizar hilos
#include <chrono>   //Librería para manejo de tiempos
int main()
{
    while (1)
    {
        std::cout << "World "; //Imprime la palabra "World" en la consola
        std::cout.flush();//Limpieza de buffers asociados al estándar output
        std::this_thread::sleep_for(std::chrono::milliseconds(1000000)); //Pausa por
            1 segundo
    }
}
```

Cuadro 3: Código Lab3_World.cpp

Los siguientes códigos son las traducciones de L3_Hilos_Ej1 y L3_Hilos_Ej2.

```cpp
/*
 ============================================================================
 Nombre: L3_Hilos_Ej1.cpp
 Autor:  Rodrigo José García Ambrosy
 ============================================================================
 */
#include <iostream> //Librería estándar de entrada/salida
#include <thread>   //Librería para utilizar hilos
#include <chrono>   //Librería para manejo de tiempos


// Código a ejecutar por el segundo hilo
void My_Thread()
{
    std::cout << "No soy el primer hilo." << std::endl;
    std::cout.flush();
    std::this_thread::sleep_for(std::chrono::seconds(2));
}

// Función principal (primer hilo de ejecución)
int main()
{
    std::thread thread2;  // Variable para identificar el 2do hilo que se creará.

    // La siguiente función crea un hilo usando std::thread.
    thread2 = std::thread(My_Thread);

    std::cout << "Soy el primer hilo." << std::endl;
    std::cout.flush();
    // La función join espera a que el hilo indicado termine (bloquea el hilo
        principal).
    thread2.join();

    std::cout << "Después de que el 2do hilo haya terminado." << std::endl;

    return 0;
}
```

Cuadro 4: Código L3_Hilos_Ej1.cpp

```cpp
/*
 ============================================================================
 Nombre: L3_Hilos_Ej2.cpp
 Autor:  Rodrigo José García Ambrosy
 ============================================================================
 */

#include <iostream> //Librería estándar de entrada/salida
#include <thread>   //Librería para utilizar hilos
#include <chrono>   //Librería para manejo de tiempos

void My_Thread(void *ptr)
{
    // La función que se ejecutará por el hilo.
    // El parámetro ptr es un puntero a un mensaje que se imprimirá.
    char *message;
    message = static_cast<char *>(ptr); //Se convierte el puntero void* a char*

    while (true)
    {
```

```
21          std::cout << message; //Se imprime el mensaje
22          std::cout.flush();    //Limpieza de búfer de salida
23          std::this_thread::sleep_for(std::chrono::microseconds(1100000));
                //Configuración para pausar por 1.1 segundos
24      }
25  }
26
27  int main()
28  {
29      std::thread thread2; //Se declara una variable para el hilo
30      char *message1 = const_cast<char *>("Hello "); //Puntero a "Hello " (constante)
31      char *message2 = "World\n"; //Puntero a "World"
32
33      thread2 = std::thread(My_Thread, static_cast<void *>(message1)); //Se crea el
            hilo y se pasa el mensaje "Hello "
34
35      while (true)
36      {
37          std::cout << message2; //Se imprime el mensaje "World"
38          std::cout.flush();     //Limpieza de búfer de salida
39          std::this_thread::sleep_for(std::chrono::microseconds(1000000)); //Pausa de
                1 segundo
40      }
41
42      return 0; //Indicador para salidas exitosas
43  }
```

Cuadro 5: Código L3_Hilos_Ej2.cpp

La segunda parte del laboratorio consiste en comprender el uso de la función **fork()** y comprender el contexto entre los procesos e hilos. En este caso se tradujeron también los programas L3_fork_Ej1.c, L3_fork_contexto.c y L3_pthread_contexto.c. Estos utilizaron las librerías de chrono, thread, iostream y se mantuvo la librería unistd.h.

Los siguientes códigos son las traducciones de L3_fork_Ej1, L3_fork_contexto y L3_-pthread_contexto.

```
1   /*
2    ================================================================================
3    Nombre: L3_fork_Ej1.cpp
4    Autor:  Rodrigo José García Ambrosy
5    ================================================================================
6    */
7
8   #include <iostream> //Librería estándar de entrada/salida
9   #include <thread>   //Librería para utilizar hilos
10  #include <chrono>   //Librería para manejo de tiempos
11  #include <unistd.h> //Se utiliza para crear nuevos procesos utilizando fork()
12
13  void child(){ //Función que se ejecuta en el proceso hijo y realiza la tarea de
        imprimir 'Hello' cada 1.1 segundos
14      while (1){
15          std::cout << "Hello ";
16          std::cout.flush();//Limpieza de buffers asociados al estándart output para
                que se imprima inmediatamente
17          std::this_thread::sleep_for(std::chrono::microseconds(1100000)); //Pausa por
                1.1 segundos
18      }
19  }
```

```
20
21 int main(){
22     int a;
23     if ((a = fork()) < 0){ //Se crea un nuevo proceso hijo
24         std::cout << "fork error \n";
25         exit(-1);
26     }
27
28     if (a == 0){
29         child();
30     }
31
32     while (1){ //Código del proceso padre que imprime 'World' cada 1 segundo
33         std::cout << "World\n";
34         std::cout.flush();
35         std::this_thread::sleep_for(std::chrono::microseconds(1000000)); //Pausa por
               1 segundo
36     }
37
38     return 0;
39 }
```

Cuadro 6: Código L3_fork_Ej1.cpp

```
1  /*
2   ============================================================================
3   Nombre: L3_fork_contexto.cpp
4   Autor:  Rodrigo José García Ambrosy
5   ============================================================================
6   */
7
8  #include <iostream> //Librería estándar de entrada/salida
9  #include <thread>   //Librería para utilizar hilos
10 #include <chrono>   //Librería para manejo de tiempos
11 #include <unistd.h> //Se utiliza para crear nuevos procesos utilizando fork()
12
13 //Variable global que funciona como contador
14 int counter = 0;
15
16 void child(){ //Función del proceso hijo
17     std::cout << "Child initial count: " << counter << std::endl;
18     counter = 200;
19
20     while (1)
21     {
22         counter++;
23         std::cout << "Child counter = " << counter << std::endl; //Contador del
               proceso hijo
24         std::cout.flush(); // Limpieza de búfer de salida
25         std::this_thread::sleep_for(std::chrono::microseconds(1100000));
26     }
27 }
28
29 int main()
30 {
31     int a;
32     counter = 100;
33
34     std::cout << "Parent initial count: " << counter << std::endl;
35
36     //Se crea un nuevo proceso hijo
37     if ((a = fork()) < 0)
38     {
```

```
        std::cout << "fork error" << std::endl;
        exit(-1);
    }

    if (a == 0) // Proceso hijo
        child(); // Se inicia la ejecución de la función del proceso hijo

    //Proceso padre
    while (true)
    {
        counter++;
        std::cout << "Parent counter = " << counter << std::endl; //Contador del
            proceso padre
        std::cout.flush();
        std::this_thread::sleep_for(std::chrono::microseconds(1000000));
    }

    return 0;
}
```

Cuadro 7: Código L3_fork_contexto.cpp

```
/*
 ============================================================================
 Nombre: L3_pthread_contexto.cpp
 Autor:  Rodrigo José García Ambrosy
 ============================================================================
 */
#include <iostream> //Librería estándar de entrada/salida
#include <thread>   //Librería para utilizar hilos
#include <chrono>   //Librería para manejo de tiempos
#include <unistd.h> //Se utiliza para crear nuevos procesos utilizando fork()

int counter = 0; //Contador global

void My_thread(){ //Función del hilo secundario
    std::cout << "My_thread initial count: " << counter << std::endl;
    counter = 200;

    while (true)
    {
        counter++;
        std::cout << "My_thread: " << counter << std::endl;
        std::cout.flush(); // Limpieza de búfer de salida
        std::this_thread::sleep_for(std::chrono::microseconds(1100000));
    }
}

int main(){
    std::thread thread2(My_thread); //Creación de un hilo secundario

    counter = 100;
    std::cout << "ParentThread initial count: " << counter << std::endl;

    //Proceso del hilo principal
    while (true)
    {
        counter++;
        std::cout << "ParentThread: " << counter << std::endl; //Contador del hilo
            principal
        std::cout.flush(); // Limpieza de búfer de salida
        std::this_thread::sleep_for(std::chrono::microseconds(1000000)); //Espera de
            1 segundo
```

```
    }

}
```

Cuadro 8: Código L3_pthread_contexto.cpp

La tercera parte del laboratorio consiste en analizar más ejemplos relacionados a los temas cubiertos en la primera y segunda parte, en este caso se tradujo L3_varios_forks.c.

```
/*
 ============================================================================
 Nombre: L3_varios_forks.cpp
 Autor:  Rodrigo José García Ambrosy
 ============================================================================
 */
#include <iostream> //Librería estándar de entrada/salida
#include <thread>   //Librería para utilizar hilos
#include <chrono>   //Librería para manejo de tiempos
#include <unistd.h> //Se utiliza para crear nuevos procesos utilizando fork()

int main(){
    int a;

    a = fork(); //Se crea el primer proceso hijo
    a = fork(); //Se crean dos procesos hijos más
    a = fork();

    //Todos los procesos ejecutan esta línea de código
    std::cout << "Mensaje, valor =" << a << std::endl;
    std::cout.flush();

    //Pausa de 10 segundos antes de que terminen los procesos
    std::this_thread::sleep_for(std::chrono::seconds(10));
    return 0;
}
```

Cuadro 9: Código L3_varios_forks.cpp

7.2. Laboratorio núm. 5

El laboratorio núm. 5 tiene como objetivo aprender el manejo de puertos GPIO de tal manera para familiarizarse con los puertos de entrada/salida de uso general, utilizando la librería de Wiringpi.h.

Al realizar la traducción de los programas de este laboratorio, como se esperaba y se indica en la página oficial de WiringPi esta librería si es compatible con C++. En este se desarrollaron dos programas uno que funciona con LEDs y el otro con una bocina. En estos programas se agregaron las librerías estándar de C++ como cstdlib y ctime, además de las ya utilizadas en el laboratorio núm.3.

El programa realizado para la configuración de LEDs es la siguiente:

```cpp
/*
 ============================================================================
 Nombre: L5_LEDs.cpp
 Autor: Rodrigo José García Ambrosy
 ============================================================================
 */
#include <iostream> // Librería estándar de entrada/salida
#include <cstdlib>  // Librería estándar para funciones generales (Se incluye para utilizar la función rand)
#include <ctime>    // Librería para acceder y manipular el tiempo
#include <chrono>   // Librería para manejo de tiempos
#include <thread>   // Librería para utilizar hilos
#include <wiringPi.h> // Librería para controlar pines GPIO

// Definición de pines para los LEDs
#define LED1  8 // Se define el pin para el LED 1
#define LED2  9 // Se define el pin para el LED 2
#define MAX_RANDOM 1000000 // Valor máximo para la generación de números random

int main() {
    std::srand(static_cast<unsigned>(std::time(0))); // Semilla de la generación de números aleatorios basada en el tiempo actual

    if (wiringPiSetup() == -1) { // Verificar si se inicializó WiringPi correctamente
        std::cerr << "Error initializing WiringPi." << std::endl;
        return 1;
    }

    // Configuración de pines como salida
    pinMode(LED1, OUTPUT);
    pinMode(LED2, OUTPUT);

    // Se apagan los LEDs al inicio
    digitalWrite(LED1, LOW);
    digitalWrite(LED2, LOW);

    while (true) {
        std::this_thread::sleep_for(std::chrono::microseconds(500000)); // Pausa de 0.5 segundos
        std::this_thread::sleep_for(std::chrono::microseconds(std::rand() % MAX_RANDOM)); // Pausa aleatoria

        // Encender LED1 y apagar LED2
        digitalWrite(LED1, HIGH);
        digitalWrite(LED2, LOW);

        std::this_thread::sleep_for(std::chrono::microseconds(500000));
        std::this_thread::sleep_for(std::chrono::microseconds(std::rand() % MAX_RANDOM));

        // Encender LED2 y apagar LED1
        digitalWrite(LED1, LOW);
        digitalWrite(LED2, HIGH);
    }

    return 0;
}
```

Cuadro 10: Código Lab5_LEDs.cpp

En el programa realizado para el uso de la bocina no fue necesario mantener la librería ctime, ya que no es necesaria para el código que se realizó. El cual es el siguiente:

```
/*
 ================================================================================
 Nombre: L5_bocina.cpp
 Autor: Rodrigo José García Ambrosy
 ================================================================================
 */

#include <iostream>     // Librería estándar de entrada/salida
#include <cstdlib>      // Librería estándar para funciones generales
#include <unistd.h>     // Se utiliza para crear nuevos procesos utilizando fork()
#include <wiringPi.h>   // Librería para controlar pines GPIO
#include <thread>       // Librería para utilizar hilos
#include <chrono>       // Librería para manejo de tiempos


#define SPKR 22 // Se define el pin 22 para la bocina
#define BTN1 27 // Se define el pin 27 para el botón



// Función que espera la entrada desde el teclado
void *teclado(void *ptr) {
    char *input = static_cast<char *>(ptr); // Se obtiene el puntero al caracter ingresado

    while (*input != 's') { // Mientras no se ingrese la tecla 's'
        std::cin >> *input; // Se lee la entrada del teclado
    }
}

int main() { // Función principal
    char opcion = 'r';
    int boton = LOW; // Estado del botón
    std::thread teclado_thr(teclado, &opcion); // Se crea un hilo para leer la entrada del teclado

    if (wiringPiSetup() == -1) { // Se inicializa la librería WiringPi
        std::cerr << "Error initializing WiringPi." << std::endl;
        return 1;
    }

    pinMode(SPKR, OUTPUT); // Configura el pin de la bocina como salida
    pinMode(BTN1, INPUT);  // Configura el pin del botón como entrada
    pullUpDnControl(BTN1, PUD_DOWN); // Configura el botón como pull-down

    std::cout << "Presione el botón para iniciar el sonido." << std::endl; // Mensaje de inicio
    std::cout.flush();

    while (!boton) {
        boton = digitalRead(BTN1); // Lee el estado del botón
        std::this_thread::sleep_for(std::chrono::microseconds(1000)); // Espera 1 ms
    }

    // Muestra las opciones de control por teclado
    std::cout << "Opciones del teclado:" << std::endl;
    std::cout << "\np - pausar\nr - reanudar\ns - salir del programa\n\n";
    std::cout.flush(); // Limpieza de búfer de salida

    while (opcion != 's') { // Mientras no se ingrese la tecla 's'
        if (opcion == 'r') { // Si se ingresa la tecla 'r'
            std::this_thread::sleep_for(std::chrono::microseconds(1000)); // Espera 1 ms
            digitalWrite(SPKR, HIGH); // Enciende la bocina
```

```
            std::this_thread::sleep_for(std::chrono::microseconds(1000));
            digitalWrite(SPKR, LOW); // Apaga la bocina
        } else {
            std::this_thread::sleep_for(std::chrono::microseconds(1000));
        }
    }
    // Espera a que termine el hilo de lectura del teclado
    teclado_thr.join();
    std::cout << "Saliendo del programa..." << std::endl << std::endl;

    return 0; // Termina el programa
}
```

Cuadro 11: Código Lab5_bocina.cpp

7.3. Laboratorio núm. 6

El laboratorio núm. 6 tiene como objetivo aprender a configurar temporizadores, hilos como tareas periódicas estableciendo sus prioridades y observar las ventajas y desventajas de sincronizar tareas utilizando únicamente periodos y tiempos de inicio.

Este laboratorio consiste en unir dos textos divididos en líneas pares e impares, por lo que se debe de reconstruir el texto. Se pide analizar previamente, dos archivos, los cuales son Lab6_files_y_strings.c y Lab6_Timer_functions.c. Luego, con base en el análisis, se crea el programa principal para unir los dos textos. El primer programa que se analizó fue Lab6_files_y_strings en el cual se utilizaron las librerías estándar iostream, fstream, string y vector. Este código se encarga de separar un texto en dos archivos distintos.

```
/*
 ================================================================================
 Nombre: Lab6_files_y_strings.cpp
 Autor:  Rodrigo José García Ambrosy
 ================================================================================
 */

#include <iostream> //Librería estándar de entrada/salida
#include <fstream> //Librería para manejo de archivos
#include <string>  //Librería para la manipulación de cadenas
#include <vector>  //Librería para el uso de vectores

constexpr int MAX_LETRAS = 100; //Máximo de letras
constexpr int MAX_CADENAS = 60; //Máximo de cadenas
constexpr char ORIGINAL[] = "Prueba.txt"; //Archivo original
constexpr char PRIMERO[] = "primero.txt"; //Primer archivo
constexpr char SEGUNDO[] = "segundo.txt"; //Segundo archivo

int main() //Función principal
{
    std::ifstream fp_original(ORIGINAL); //Se abre el archivo original
    if (!fp_original.is_open()) //Condicional para verificar si se abrio el archivo
    {
        std::cerr << "Error al abrir el archivo." << std::endl;
        exit(1);
    }
    //Se crea un vector para almacenar las lineas leidas
```

```
28      std::vector<std::string> StringArray;
29      std::string line; //Se crea una variable para almacenar las lineas leidas
30      while (std::getline(fp_original, line)) //Se lee una linea del archivo
31      {
32          StringArray.push_back(line); //Se agrega una linea en el vector
33          std::cout << line << std::endl; //Se imprime la salida
34      }
35      fp_original.close(); //Se cierra el archivo original
36
37      std::cout << "\nNúmero de líneas leídas: " << StringArray.size() << std::endl;
38
39      std::ofstream fp_primero(PRIMERO); //Se abre el primer archivo
40      std::ofstream fp_segundo(SEGUNDO); //Se abre el segundo archivo
41
42      //Funcion para separar las lineas en dos archivos diferentes
43      for (size_t i = 0; i < StringArray.size(); i++)
44      {
45          if (i % 2 == 0) //Se verifica si el indice es par
46              fp_primero << StringArray[i] << std::endl; //Se escribe en el primer
                    archivo
47          else
48              fp_segundo << StringArray[i] << std::endl; //Se escribe en el segundo
                    archivo
49      }
50
51      fp_primero.close();//Se cierran los archivos de salida
52      fp_segundo.close();
53
54      std::cout << "\nListo..." << std::endl;
55
56      return 0;
57  }
```

Cuadro 12: Código Lab6_files_y_strings.cpp

Para el codigo de Lab6_Timer_functions se utilizaron las librerías estándar de C++ las cuales son cstdlib y cstdint. También se mantuvieron las librerías propias de Linux como sched.h y timerfd.h, el código se mantuvo casi completamente similar al código de C ya que las librerías que se usan para configurar el timer y el algoritmo de escalonamiento pueden funcionar de igual manera ya sea en C o en C++. El código Lab6_Timer_functions.cpp es el siguiente:

```cpp
/*
 ============================================================================
 Nombre: Lab6_Timer_functions.cpp
 Autor:  Rodrigo José García Ambrosy
 ============================================================================
 */

#include <unistd.h> //Librería que proporciona funciones y constantes específicas de sistemas Unix.
#include <iostream> //Librería estándar de entrada/salida
#include <cstdlib> //Librería estándar para funciones generales
#include <cstdint> //Librería estándar para tipos de datos enteros
#include <sys/timerfd.h> //Librería para utilizar el timer
#include <sched.h> //Librería para utilizar la planificación

#define MI_PRIORIDAD 1 //Se define la prioridad del proceso
#define Periodo_segundos 1 //Se define el periodo del timer
#define Periodo_nanosegundos 0 //Se define el periodo del timer
#define TiempoIni_segundos 0 //Se define el tiempo inicial del timer
#define TiempoIni_nanosegundos 0 //Se define el tiempo inicial del timer

int main(){
    struct sched_param param; //Se define una estructura para la planificación
    // Asignar prioridad y la política de escalonamiento
    param.sched_priority = MI_PRIORIDAD;
    sched_setscheduler(0, SCHED_FIFO, &param); //Se asigna la prioridad y la política de escalonamiento

    // --- Configurar Timer (se podría crear una función que regrese el file descriptor) ---
    int timer_fd = timerfd_create(CLOCK_MONOTONIC, 0);

    struct itimerspec itval;
    // El Timer se "dispara" cada ## segundos + ## nanosegundos
    itval.it_interval.tv_sec = Periodo_segundos;      // Revisar el tipo de la variable ( int , long?)
    itval.it_interval.tv_nsec = Periodo_nanosegundos; // Revisar el tipo de la variable ( int , long?)

    // El Timer empezará en ## segundos + ## nanosegundos desde el momento en que se inicie el Timer
    itval.it_value.tv_sec = TiempoIni_segundos;      // Revisar el tipo de la variable
    itval.it_value.tv_nsec = TiempoIni_nanosegundos; // Revisar el tipo de la variable

    // Arrancar el Timer
    timerfd_settime(timer_fd, 0, &itval, nullptr);

    uint64_t num_periods; //Se define una variable para almacenar el número de periodos

    num_periods = 0;
    read(timer_fd, &num_periods, sizeof(num_periods));   // esta función retorna algo
                        // Siempre es bueno chequear errores. Revisar la función read()
    if(num_periods > 1)
    {
        std::cout << "MISSED WINDOW" << std::endl;  // Manda la cadena al estándar output
        exit(1);
    }
    return 0;
}
```

Cuadro 13: Código Lab6_Timer_functions.cpp

El resultado final del laboratorio tomando en cuenta los programas anteriores queda de la siguiente manera. Las librerías adicionales de C++ son cstdio y cerrno aparte de las librerías que ya se usaron en los programas anteriores. Adicionalmente se incluyó la librería de Linux types.h. El resultado del programa que se llamó Lab6.cpp, es el siguiente.

```
/*
 ============================================================================
 Nombre: Lab6.cpp
 Autor:  Rodrigo José García Ambrosy
 ============================================================================
 */

#include <iostream> //Librería estándar de entrada/salida
#include <cstdio>   //Librería estándar de entrada/salida
#include <cstdlib>  //Librería estándar para funciones generales
#include <thread>   //Librería para utilizar hilos
#include <chrono>   //Librería para utilizar la función sleep_for
#include <string>   //Librería para la manipulación de cadenas
#include <cstring>  //Librería para usar strcpy
#include <cerrno>   //Librería para utilizar la variable errno
#include <sys/timerfd.h> //Librería para utilizar el timer
#include <unistd.h> //Librería que proporciona funciones y constantes específicas de sistemas Unix.
#include <linux/types.h> // Librería que contiene definiciones de tipos de datos comunes en sistemas Linux
#include <sched.h> //Librería para utilizar la planificación
#include <fstream> //Librería para utilizar archivos



#define MI_PRIORIDAD 10  // Rango entre 1 y 99. A mayor valor, más alta la prioridad
                         // Parece que no deja asignar prioridades mayores a 95 (RPi)
#define MAX_LETRAS 100   // Máximo de letras
#define MAX_CADENAS 60   // Máximo de cadenas
#define MILI_A_NANO 1000000 // Valor para convertir de milisegundos a nanosegundos
#define PRIMERO "Lab6_primero.txt" //Primer archivo
#define SEGUNDO "Lab6_segundo.txt" //Segundo archivo
#define RECONSTRUIDO "Lab6_reconstruido.txt" //Archivo reconstruido
#define PERIODO_1_y_2 20 //Periodo de los hilos 1 y 2
#define PERIODO_3     10 //Periodo del hilo 3
#define INIT_1   1 //Tiempo inicial del hilo 1
#define INIT_2  11 //Tiempo inicial del hilo 2
#define INIT_3   6 //Tiempo inicial del hilo 3

//Declarar funciones
int timer_config(int, int); //Configurar el timer
void wait_period(const int); //Esperar el periodo
void imprimir_y_guardar(); //Imprimir y guardar el archivo reconstruido

//Defino variables globales
char StringArray[MAX_CADENAS][MAX_LETRAS]; //Arreglo de cadenas
int cont1 = 0, cont2 = 0; //Contadores de cadenas


void Primero(char *Buff) { //Función del hilo 1
    Buff = static_cast<char *>(Buff); //Casteo de la variable Buff
    struct sched_param param; //Se define una estructura para la planificación
    param.sched_priority = MI_PRIORIDAD; //Se asigna la prioridad y la política de escalonamiento
    if (sched_setscheduler(0, SCHED_FIFO, &param) == -1) { //Se asigna la prioridad y la política de escalonamiento
        perror("sched_setscheduler falló, thread 1");
```

```
54          exit(20);
55      }
56
57      FILE *fp_primero = fopen(PRIMERO, "r"); //Se abre el primer archivo
58
59      if (fp_primero == nullptr) { //Condicional para verificar si se abrio el archivo
60          std::cout << "El archivo " << PRIMERO << " no se abrió correctamente." <<
                std::endl;
61          exit(0);
62      }
63
64      int timer_fd = timer_config(PERIODO_1_y_2, INIT_1); //Se configura el timer
65      wait_period(timer_fd); //Se espera el periodo
66
67      while (fgets(Buff, MAX_LETRAS, fp_primero) != nullptr) { //Se lee una linea del
            archivo
68          cont1++; //Se incrementa el contador
69          wait_period(timer_fd); //Se espera el periodo
70      }
71      fclose(fp_primero); //Se cierra el archivo
72  }
73
74  void Segundo(char *Buff) { //Función del hilo 2
75      Buff = static_cast<char *>(Buff); //Casteo de la variable Buff
76      struct sched_param param;
77      param.sched_priority = MI_PRIORIDAD;
78      if (sched_setscheduler(0, SCHED_FIFO, &param) == -1) {
79          perror("sched_setscheduler falló, thread 2");
80          exit(20);
81      }
82
83      FILE *fp_segundo = fopen(SEGUNDO, "r"); //Se abre el segundo archivo
84
85      if (fp_segundo == nullptr) {
86          std::cout << "El archivo " << SEGUNDO << " no se abrió correctamente." <<
                std::endl;
87          exit(0);
88      }
89
90      int timer_fd = timer_config(PERIODO_1_y_2, INIT_2); //Se configura el timer
91      wait_period(timer_fd);
92
93      while (fgets(Buff, MAX_LETRAS, fp_segundo) != nullptr) { //Se lee una linea del
            archivo
94          cont2++; //Se incrementa el contador
95          wait_period(timer_fd); //Se espera el periodo
96      }
97      fclose(fp_segundo); //Se cierra el archivo
98  }
99
100 void Tercero(char *Buff) { //Función del hilo 3
101     int i = 0; //Se define una variable para el ciclo for
102     Buff = static_cast<char *>(Buff); //Casteo de la variable Buff
103
104     struct sched_param param; //Se define una estructura para la planificación
105     param.sched_priority = MI_PRIORIDAD;
106     if (sched_setscheduler(0, SCHED_FIFO, &param) == -1) { //Se asigna la prioridad
            y la política de escalonamiento
107         perror("sched_setscheduler falló, thread 3");
108         exit(20);
109     }
110
111     int timer_fd = timer_config(PERIODO_3, INIT_3); //Se configura el timer
112     wait_period(timer_fd);
113
114     for (i = 0; i < MAX_CADENAS; i++) { //Se recorre el arreglo de cadenas
115
116         std::strcpy(StringArray[i], Buff); //Se copia la cadena en el arreglo
```

```
117         wait_period(timer_fd); //Se espera el periodo
118     }
119 }
120 
121 int main(void) { //Función principal
122     std::thread thrd1, thrd2, thrd3; //Se definen los hilos
123     char Buffer[MAX_LETRAS]; //Se define el buffer
124 
125     thrd1 = std::thread(Primero, Buffer); //Se crean los hilos
126     thrd2 = std::thread(Segundo, Buffer);
127     thrd3 = std::thread(Tercero, Buffer);
128 
129     thrd1.join();
130     thrd2.join();
131     thrd3.join();
132 
133     imprimir_y_guardar(); //Se imprime y guarda el archivo reconstruido
134 
135     std::cout << "\n\nFin del programa...\n";
136 
137     return EXIT_SUCCESS;
138 }
139 
140 int timer_config(int period, int inittime) {
141     // Crea un timer y verifica si se creó correctamente
142     int fd = timerfd_create(CLOCK_MONOTONIC, 0);
143     if (fd == -1) {
144         perror("Error al crear el timer.");
145         exit(1);
146     }
147 
148     // Asegura que el periodo esté en un rango válido
149     if (period > 999)
150         period = 999;
151     if (period < 1)
152         period = 1;
153 
154     struct itimerspec itval;
155     // Configura el intervalo y el tiempo inicial del timer
156     itval.it_interval.tv_sec = 0;
157     itval.it_interval.tv_nsec = static_cast<long>(period) * MILI_A_NANO;
158     itval.it_value.tv_sec = 0;
159     itval.it_value.tv_nsec = static_cast<long>(inittime) * MILI_A_NANO;
160 
161     // Inicia el timer
162     if (timerfd_settime(fd, 0, &itval, NULL) == -1) {
163         perror("Error al iniciar el timer.");
164         exit(1);
165     }
166     return fd; // Retorna el file descriptor del timer
167 }
168 
169 void wait_period(const int fd) {
170     uint64_t num_periods = 0;
171     // Espera a que el timer expire y obtiene el número de períodos
172     if (read(fd, &num_periods, sizeof(num_periods)) == -1) {
173         perror("Error al leer el timer.");
174         exit(1);
175     }
176 
177     if (num_periods > 1) {
178         std::cout << "Se pasó de un período." << std::endl;
179         exit(1);
180     }
181 }
182 
183 void imprimir_y_guardar() {
184     FILE *fp_reconstruido;
```

```
185     int i;
186     fp_reconstruido = fopen(RECONSTRUIDO, "w");
187 
188     // Escribe en el archivo reconstruido y muestra en la consola
189     for (i = 0; i < (cont1 + cont2); i++) {
190         fputs(StringArray[i], fp_reconstruido);
191         std::cout << StringArray[i];
192         fflush(stdout);
193     }
194 
195     fclose(fp_reconstruido); // Cierra el archivo reconstruido
196 }
```

Cuadro 14: Código Lab6.cpp

7.4. Laboratorio núm. 7

El laboratorio núm.7 tiene como objetivo aprender a utilizar *semaphores* para la sincronización de tareas, observar los efectos que puedan surgir al utilizar distintas prioridades en los hilos y comprender sobre los problemas y limitaciones de algunos algoritmos de escalonamiento.

Este laboratorio consiste en recrear un semáforo de tránsito usando la Raspberry Pi, esto se logrará utilizando los puertos GPIO y adicionalmente se usa un botón para simular los botones peatonales que se encuentran en los semáforos. La configuración del semáforo variará utilizando escalonamiento por prioridades y sincronización por *semaphores*.

En la primera parte se espera realizar el funcionamiento del semáforo utilizando un algoritmo de escalonamiento por poleo. En el caso de este laboratorio se decidió utilizar una librería estática como apoyo para el programa para la lectura del estatus del botón a utilizar al trabajar con el lenguaje de programación de C. Por lo que en lugar de utilizar una librería estática se decidió implementar el uso de una clase ya que C++ nos permite el uso de estas para facilitar el programa y no tener que depender de una librería estática. De esta forma se le puede dar un enfoque en la programación orientada de objetos y así mostrar una de las diferencias claves que existen entre C y C++ al utilizar clases. Las librerías que se utilizaron en esta primera parte del laboratorio son iostream, thread, chrono y WiringPi.h.

El programa Lab7_parte1.cpp quedo de la siguiente manera:

```
1 //============================================================================
2 // Nombre: Lab7_parte1.cpp
3 // Autor: Rodrigo José García Ambrosy
4 //============================================================================
```

```
5
6  #include <iostream> // Librería estándar de entrada/salida
7  #include <thread>   // Librería para utilizar hilos
8  #include <chrono>   // Librería para utilizar la función sleep_for
9  #include <wiringPi.h> // Librería para controlar pines GPIO
10
11 #define LUZ_1  3 // Pin para la luz 1
12 #define LUZ_2  4 // Pin para la luz 2
13 #define LUZ_P  5 // Pin para la luz de parada
14 #define BTN1   16 // Pin para el botón
15
16 class ButtonHandler { // Clase para el control del botón
17 public:
18     ButtonHandler() {
19         pinMode(BTN1, INPUT);
20         pullUpDnControl(BTN1, PUD_DOWN); // Configura pull down para el botón
21     }
22
23     bool isButtonPressed() { // Verifica si el botón está presionado
24         return digitalRead(BTN1) == HIGH;
25     }
26
27     void waitForButtonRelease() {
28         while (isButtonPressed()) {
29             // Espera a que el botón se suelte
30         }
31     }
32 };
33
34 // Hilo a tiempo real para el parpadeo de los LEDs con un algoritmo de
       escalonamiento por Poleo
35 void polled_scheduling(ButtonHandler &buttonHandler) {
36     while (true) {
37         digitalWrite(LUZ_1, HIGH);
38         std::this_thread::sleep_for(std::chrono::seconds(1));
39         digitalWrite(LUZ_1, LOW);
40
41         digitalWrite(LUZ_2, HIGH);
42         std::this_thread::sleep_for(std::chrono::seconds(1));
43         digitalWrite(LUZ_2, LOW);
44
45         if (buttonHandler.isButtonPressed()) {
46             digitalWrite(LUZ_P, HIGH);
47             std::this_thread::sleep_for(std::chrono::seconds(1));
48             digitalWrite(LUZ_P, LOW);
49             buttonHandler.waitForButtonRelease();
50         }
51     }
52 }
53
54 int main() {
55     wiringPiSetup(); // Inicializa la librería wiringPi
56
57     pinMode(LUZ_1, OUTPUT);
58     pinMode(LUZ_2, OUTPUT);
59     pinMode(LUZ_P, OUTPUT);
60
61     digitalWrite(LUZ_1, LOW);
62     digitalWrite(LUZ_2, LOW);
63     digitalWrite(LUZ_P, LOW);
64
65     ButtonHandler buttonHandler; // Crea un objeto de la clase ButtonHandler
66     std::thread polled_thread(polled_scheduling, std::ref(buttonHandler)); // Crea
           un hilo para el algoritmo de escalonamiento por Poleo
67
68     polled_thread.join();
69
70     return EXIT_SUCCESS;
```

```
}
```

Cuadro 15: Código Lab7_parte1.cpp

En la segunda parte, se espera realizar el funcionamiento del semáforo utilizando un escalonamiento por prioridades y mediante una sincronización a través de *semaphores*. Se planea crear un hilo por cada LED para que sea responsable de encender y apagar una de las luces, y estos deben ser capaces de sincronizarse. El programa también puede ajustar la prioridad y la política de escalonamiento de los hilos. Al desarrollar el programa, de la misma manera, se creó la clase para el botón, y adicionalmente, se creó una clase para el manejo de los LEDs, aparte del control de los hilos para gestionar las prioridades de los LEDs. La librería para utilizar *semaphores* en C++ se conoce como "*semaphore*", y esta librería solo es compatible con las versiones de C++20. Debido a las limitaciones de la Raspberry Pi 3, se utilizó la librería *semaphore.h*, que es estándar en C, para el uso de sus funciones. Es importante tener en cuenta que C++ nos permite utilizar librerías de C sin problemas al momento de compilar. El programa Lab7_parte2.cpp queda de la siguiente manera:

```
/*
 ============================================================================
 Nombre: Lab7_parte2.cpp
 Autor:  Rodrigo José García Ambrosy
 ============================================================================
 */
#include <iostream> // Librería estándar de entrada/salida
#include <thread>   // Librería para utilizar hilos
#include <chrono>   // Librería para utilizar la función sleep_for
#include <wiringPi.h> // Librería para controlar pines GPIO
#include <semaphore.h>   // Librería para utilizar semáforos

#define LUZ_1  3 // Pin para la luz 1
#define LUZ_2  4 // Pin para la luz 2
#define LUZ_P  5 // Pin para la luz de parada
#define BTN1  16 // Pin para el botón

#define LUZ_1_PRI 10 // Prioridad de la luz 1
#define LUZ_2_PRI 10 // Prioridad de la luz 2
#define LUZ_P_PRI 10 // Prioridad de la luz de parada

#define PERIOD 750 // Periodo de las luces

sem_t sem; // Declaración del semáforo

class LightController { // Clase para el control de las luces
public:
    LightController(int pin) : pin_(pin) { // Constructor
        pinMode(pin_, OUTPUT);
    }

    void turnOn() { // Enciende la luz
        digitalWrite(pin_, HIGH);
    }

    void turnOff() { // Apaga la luz
        digitalWrite(pin_, LOW);
    }

private:  // Atributos
```

```
41      int pin_;
42  };
43
44  class ButtonHandler { // Clase para el control del botón
45  public:
46      ButtonHandler(int pin) : pin_(pin) {
47          pinMode(pin_, INPUT); // Configura el pin como entrada
48          pullUpDnControl(pin_, PUD_DOWN); // Configura pull down para el botón
49      }
50
51      bool isButtonPressed() { // Verifica si el botón está presionado
52          return digitalRead(pin_) == HIGH;
53      }
54
55      void waitForButtonRelease() { // Espera a que el botón se suelte
56          while (isButtonPressed()) {
57              std::this_thread::sleep_for(std::chrono::milliseconds(1));
58          }
59      }
60
61  private:
62      int pin_;
63  };
64
65
66  void luz(LightController &light, int priority) { // Función para el control de las
        luces
67      struct sched_param param; //Se define una estructura para la planificación
68      param.sched_priority = priority; //Se asigna la prioridad y la política de
            escalonamiento
69      if (sched_setscheduler(0, SCHED_RR, &param) == -1) { //Se asigna la prioridad y
            la política de escalonamiento
70          perror("sched_setscheduler falló");
71          exit(20);
72      }
73
74      while (true) {
75          sem_wait(&sem); //Se espera a que el semáforo esté disponible
76          light.turnOn(); //Se enciende la luz
77          std::this_thread::sleep_for(std::chrono::milliseconds(PERIOD));
78          light.turnOff(); //Se apaga la luz
79          sem_post(&sem);
80          std::this_thread::sleep_for(std::chrono::milliseconds(1));
81      }
82  }
83
84  void peatonal(LightController &light, ButtonHandler &buttonHandler, int priority) {
        // Función para el control de la luz peatonal
85      struct sched_param param;
86      param.sched_priority = priority;
87      if (sched_setscheduler(0, SCHED_RR, &param) == -1) {
88          perror("sched_setscheduler falló");
89          exit(20);
90      }
91
92      while (true) {
93          sem_wait(&sem);
94          if (buttonHandler.isButtonPressed()) { //Se verifica si el botón está
                presionado
95              light.turnOn();
96              std::this_thread::sleep_for(std::chrono::milliseconds(PERIOD));
97              light.turnOff();
98              buttonHandler.waitForButtonRelease();
99          }
100         sem_post(&sem);
101         std::this_thread::sleep_for(std::chrono::milliseconds(1));
102     }
103 }
```

```
104
105 int main() { // Función principal
106     sem_init(&sem, 0, 1);
107     wiringPiSetupGpio();
108
109     LightController luz1Ctrl(LUZ_1); //Se crea un objeto de la clase LightController
110     LightController luz2Ctrl(LUZ_2); //Se crea un objeto de la clase LightController
111     LightController luzPCtrl(LUZ_P); //Se crea un objeto de la clase LightController
112     ButtonHandler buttonHandler(BTN1); //Se crea un objeto de la clase ButtonHandler
113
114     std::thread hilo_luz1(luz, std::ref(luz1Ctrl), LUZ_1_PRI); //Se crea un hilo
            para el control de la luz 1
115     std::thread hilo_luz2(luz, std::ref(luz2Ctrl), LUZ_2_PRI); //Se crea un hilo
            para el control de la luz 2
116     std::thread hilo_luzP(peatonal, std::ref(luzPCtrl), std::ref(buttonHandler),
            LUZ_P_PRI); //Se crea un hilo para el control de la luz peatonal
117
118
119     hilo_luz1.join();
120     hilo_luz2.join();
121     hilo_luzP.join();
122
123     return EXIT_SUCCESS;
124 }
```

Cuadro 16: Código Lab7_parte2.cpp

7.5. Laboratorio núm. 8

El laboratorio núm.8 tiene como objetivo aprender a comunicar y sincronizar procesos en un ambiente distribuido mediante *sockets* y una configuración maestro/esclavo.

Este laboratorio consiste en una configuración maestro/esclavo por lo cual se debe configurar una, dos o más Raspberry Pi como servidores y una computadora u otra Raspberry Pi como cliente. El protocolo de comunicación que se utilizará en este laboratorio es UDP. Ya que la idea es que los servidores le manden mensajes al cliente cuando este ingresa ciertos comandos.

Las librerías propias de C++ utilizadas para el código del maestro fueron las siguientes: iostream, cstdlib, ctime, cstring, cstdio y sstream. Por otro lado, las librerías propias de Linux para la programación por sockets y configuraciones de protocolos de red fueron las siguientes: types.h, netdb.h, socket.h y inet.h. Por lo que el código del maestro quedá de la siguiente manera:

```
1  /*
2   ============================================================================
3   Nombre: Lab8_servidor.cpp
4   Autor:  Rodrigo José García Ambrosy
5   ============================================================================
6   */
7  #include <iostream> // Librería estándar de entrada/salida
8  #include <cstdlib>  // Librería estándar para funciones generales
9  #include <ctime>    // Librería para utilizar la función time
10 #include <cstring>  // Librería para la manipulación de cadenas
11 #include <cstdio>   // Librería para utilizar la función sprintf
```

```
12 #include <unistd.h> // Librería que proporciona funciones y constantes específicas
       de sistemas Unix.
13 #include <fcntl.h>  // Librería para utilizar la función open
14 #include <sys/types.h>  // Librería para definir tipos de datos
15 #include <netdb.h>  // Librería para definir estructuras que almacenan información
       sobre hosts
16 #include <sys/socket.h> // Librería para utilizar sockets
17 #include <arpa/inet.h>  // Librería para manipular direcciones IP
18 #include <sstream>      // Librería para utilizar la función stringstream
19
20 #define OPCION_IP 1 // 0 - hard coded
21                     // 1 - Raspberry Pi / CentOS
22                     // 2 - CentOS J-305/306
23 #define TRUE 1  // Definición de TRUE
24 #define FALSE 0 // Definición de FALSE
25 #define MSG_SIZE 60 // Tamaño del mensaje
26 #define VOTO_MAX 15 // Voto máximo
27 #define IP_LENGTH 15    // Tamaño de la dirección IP
28 #define PORT 2044   // Puerto de comunicación
29
30 int getIP(char *);  // Función para obtener la dirección IP
31
32 void error(const char *msg) // Función para imprimir errores
33 {
34     perror(msg);
35     exit(0);
36 }
37
38 int main(int argc, char *argv[]){ // Función principal
39     int sockfd, n; // Declaración de variables
40     unsigned int length;
41     struct sockaddr_in addr, broadcast_addr; // Estructuras para almacenar las
           direcciones
42     char messg[MSG_SIZE], messg_temp[MSG_SIZE]; // Variables para almacenar el
           mensaje
43     int myvote = 0, myIP, incomingIP, in_vote = 0; // Variables para almacenar el
           voto y la dirección IP
44     int boolval = TRUE, master = FALSE; // Variables para configurar el socket
45     char *vote_tokens;  // Variable para almacenar el voto
46     char IP_buffer[IP_LENGTH], IP_temp[IP_LENGTH], IP_broadcast[IP_LENGTH]; //
           Variables para almacenar la dirección IP
47     int puerto = PORT; // Variable para almacenar el puerto
48     FILE *file; // Declaración del archivo
49
50     if (argc > 2) // Verifica si se ingresó el puerto
51     {
52         std::cout << "Usage: " << argv[0] << " [port]" << std::endl;
53         exit(1);
54     }
55
56     if (argc == 2)
57         puerto = std::atoi(argv[1]);
58
59 #if OPCION_IP == 0 //
60 // --- hard coded ----------------------------------------------------------
61     strcpy(IP_buffer, "192.168.20.227");
62     strcpy(IP_broadcast, "192.168.20.255");
63 #endif
64
65 #if (OPCION_IP == 1) || (OPCION_IP == 2)
66 // ----- Obtener la dirección IP y guardarla en un archivo (ipaddr) usando ifconfig
       --------
67
68 #if OPCION_IP == 1 // Se obtiene la IP de wlan0
69     system("ifconfig wlan0 | grep 'inet ' | awk '{ print $2 }' > ipaddr");
70 #else
71     system("ifconfig enp0s31f6 | grep 'inet ' | awk '{ print $2 }' > ipaddr");
72 #endif
```

```
73
74      file = fopen("ipaddr", "r");     // Abre el archivo con la IP
75      if(file == NULL) // Verifica si se abrio el archivo
76      {
77          std::cout << "Error opening file" << std::endl;
78          exit(-1);
79      }
80      else
81      {
82          fscanf(file, "%s", IP_buffer);
83      }
84
85      fclose(file);                    // Cierra el archivo
86
87  // --- Obtener la dirección de broadcast y guardarla en el archivo ipaddr ---
88  #if OPCION_IP == 1
89      system("ifconfig wlan0 | grep 'inet ' | awk '{ print $6 }' > ipaddr"); //
            Ejecutar comando para obtener la dirección de broadcast
90  #else
91      system("ifconfig enp0s31f6 | grep 'inet ' | awk '{ print $6 }' > ipaddr");
92  #endif
93
94      file = fopen("ipaddr", "r");     // Abrir archivo con la dirección de broadcast
95      if(file == NULL)
96      {
97          std::cout << "Error al abrir el archivo" << std::endl;;
98          exit(-1);
99      }
100     else
101     {
102         fscanf(file, "%s", IP_broadcast); // Leer la dirección de broadcast del
                archivo
103     }
104
105     fclose(file);                    // Cerrar archivo
106     system("rm ipaddr");             // Asegurarse de eliminar el archivo cuando se
            termine
107
108 #endif
109
110     std::cout << "Mi IP es: " << IP_buffer << std::endl; // Imprime la dirección IP
111     std::cout << "La dirección de broadcast es: " << IP_broadcast << std::endl; //
            Imprime la dirección de boradcast
112
113     strcpy(IP_temp, IP_buffer);      // Necesario porque getIP cambia el argumento
114     myIP = getIP(IP_temp); // Obtener el último número de la IP
115
116     srand(time(NULL));               // Inicializa el generador de números aleatorios
117
118     sockfd = socket(AF_INET, SOCK_DGRAM, 0); // Crea el socket
119     if(sockfd < 0) // Verifica si hubo error
120         error("Opening socket");
121
122     addr.sin_family = AF_INET; // Configura la familia de direcciones
123     addr.sin_port = htons(puerto); // Configura el puerto
124     addr.sin_addr.s_addr = htonl(INADDR_ANY); // Configura la dirección IP
125
126     broadcast_addr.sin_family = AF_INET;
127     broadcast_addr.sin_port = htons(puerto);
128     broadcast_addr.sin_addr.s_addr = inet_addr(IP_broadcast);
129
130     length = sizeof(addr);           // Tamaño de la dirección
131
132     // Vincula el socket a la dirección del host y al número de puerto
133     if(bind(sockfd, (struct sockaddr *)&addr, length) < 0)
134         error("Error al vincular el socket.");
135
136     // Cambia los permisos del socket para permitir el broadcast
```

```
137     if(setsockopt(sockfd, SOL_SOCKET, SO_BROADCAST, &boolval, sizeof(boolval)) < 0)
138         error("Error al configurar las opciones del socket\n");
139 
140     while(1)
141     {
142         memset(messg, 0, MSG_SIZE); // Limpia el buffer
143 
144         // Recibe un mensaje del cliente
145         n = recvfrom(sockfd, messg, MSG_SIZE, 0, (struct sockaddr *)&addr, &length);
146         if(n < 0)
147             error("recvfrom");
148 
149         if(strncmp(messg, "QUIEN ES", 8) == 0) // Verifica si el mensaje es "QUIEN
                ES"
150         {
151             if(master == TRUE) // Verifica si es el Master
152             {
153                 std::cout << "A punto de informar que soy el master" << std::endl;
154 
155                 std::stringstream ss;
156                 ss << "Rodrigo RP4 en " << IP_buffer << " es el Master";
157                 std::string messg = ss.str();   // Convertir a std::string
158 
159                 sendto(sockfd, messg.c_str(), messg.length(), 0, (struct sockaddr
                        *)&addr, sizeof(struct sockaddr));
160 
161             }
162             else
163             {
164                 std::cout <<  "No soy el Master, así que no envío ningún
                        mensaje..."<< std::endl;
165             }
166         }
167         else if(strncmp(messg, "VOTE", 4) == 0) // Verifica si el mensaje es "VOTE"
168         {
169             memset(messg, 0, MSG_SIZE);     // Limpia el buffer
170             myvote = rand()%VOTO_MAX + 1;   // El voto aleatorio entre 1 y VOTO_MAX.
171             //myvote = 5;
172             master = TRUE;          // En caso de que nadie más vote, yo seré el
                    Master
173 
174             std::stringstream ss; // Convertir a std::string
175             ss << "#" << IP_buffer << " " << myvote;
176             std::string messg = ss.str();
177 
178             std::cout << "A punto de mandar mi IP y mi voto: " << messg <<
                    std::endl; // Imprime la dirección IP y el voto
179             sendto(sockfd, messg.c_str(), messg.length(), 0, (struct sockaddr
                    *)&broadcast_addr, sizeof(struct sockaddr)); // Envía el mensaje
180 
181         }
182         else if(messg[0] == '#') // Comprueba si el mensaje es un voto
183         {
184             // Extrae el voto entrante y la dirección IP del mensaje...
185             strcpy(messg_temp, messg); // necesario porque getIP cambia el argumento
186             incomingIP = getIP(messg_temp); // obtiene el último número de la IP
187 
188             if(incomingIP != myIP) // Si es mi propio voto, lo ignoro.
189             {
190                 std::cout << "Alguien más votó: " << messg << std::endl;
191                 // Obtén el voto del mensaje entrante
192                 vote_tokens = strtok(messg," #.");
193                 while(vote_tokens != NULL)
194                 {
195                     vote_tokens = strtok(NULL, " #.");
196                     if(vote_tokens != NULL)
197                         in_vote = atoi(vote_tokens);
198                 }
```

```
199
200                 if(in_vote > myvote)    // Recibí un voto mayor
201                 {
202                     if(master == FALSE)
203                     {
204                         std::cout << "Voto mayor detectado. No importa, no soy el
                                Master. " << std::endl;
205                     }
206                     else
207                     {
208                         std::cout << "Voto mayor detectado. Dejaré de ser el
                                Master." << std::endl;
209                         master = FALSE;
210                     }
211                 }
212                 else if(in_vote == myvote)  // Recibí el mismo voto (empate)
213                 {
214                     if(master == FALSE)
215                     {
216                         std::cout << "Voto igual detectado. No importa, no soy el
                                Master." << std::endl;
217                     }
218                     else
219                     {
220                         if(incomingIP < myIP)   // Era el Master pero pierdo el
                                tie-break
221                         {
222                             master = FALSE;
223                             std::cout << "Perdí el desempate. Dejaré de ser el
                                    Master." << std::endl;
224                         }
225                         else    // Era el Master y gano el tie-break
226                         {
227                             std::cout << "Gané el desempate. Sigo siendo el
                                    Master."<< std::endl;
228                         }
229                     }
230                 }
231                 else    // Recibí un voto menor
232                 {
233                     if(master == FALSE)
234                         std::cout << "Voto menor detectado. No importa, no soy el
                                Master."<< std::endl;
235                     else
236                         std::cout << "Voto menor detectado. Sigo siendo el Master."
                                << std::endl;
237                 }
238             }
239         }
240         else
241         {
242             std::cout << "Recibí un mensaje inválido: " << messg << std::endl;
243         }
244     }
245
246     return 0;
247 }
248
249 int getIP(char *IP_buffer) // Función para obtener la dirección IP
250
251 {
252
253     char *IPptr;
254     IPptr = strtok(IP_buffer, ".");
255     IPptr = strtok(NULL, ".");
256     IPptr = strtok(NULL, ".");
257     IPptr = strtok(NULL, ". ");
258     return (atoi(IPptr));
```

```
259
260 }
```

Cuadro 17: Código Lab8_servidor.cpp

Para el código del cliente se utilizaron las mismas librerías propias de C++ que en el código del maestro incluso no se incluyeron tantas, además las librerías propias del sistema operativo de Linux se mantuvieron iguales solo se agregó la librería que se llama in.h. El código del cliente es el siguiente:

```
1  /*
2   ================================================================================
3   Nombre: Lab8_cliente.cpp
4   Autor:  Rodrigo José García Ambrosy
5   ================================================================================
6   */
7  #include <iostream> // Librería estándar de entrada/salida
8  #include <cstdlib>  // Librería estándar para funciones generales
9  #include <unistd.h> // Librería que proporciona funciones y constantes específicas
       de sistemas Unix.
10 #include <cstring>  // Librería para la manipulación de cadenas
11 #include <sys/types.h> // Librería para definir tipos de datos
12 #include <sys/socket.h> // Librería para utilizar sockets
13 #include <netinet/in.h> // Librería para utilizar direcciones de internet
14 #include <netdb.h>  // Librería para definir estructuras que almacenan información
       sobre hosts
15 #include <arpa/inet.h> // Librería para manipular direcciones IP
16 #include <thread>  // Librería para utilizar hilos
17 #include <chrono>  // Librería para utilizar la función sleep_for
18
19 #define OPCION_IP 1 /// 0 - hard coded
20                     // 1 - Raspberry Pi
21 #define MSG_SIZE 60
22 #define IP_LENGTH 15 // Tamaño de la dirección IP
23
24 void error(const char *msg)
25 {
26     perror(msg);
27     exit(0);
28 }
29
30 void receiving(int sock) // Función para recibir mensajes
31 {
32     unsigned int length = sizeof(struct sockaddr_in); // Tamaño de la estructura
           sockaddr_in
33     char buffer[MSG_SIZE]; // Buffer para almacenar el mensaje
34     struct sockaddr_in from; // Estructura para almacenar la dirección del emisor
35
36     while (true)
37     {
38         memset(buffer, 0, MSG_SIZE);
39         int n = recvfrom(sock, buffer, MSG_SIZE, 0, reinterpret_cast<struct sockaddr
               *>(&from), &length); // Recibe el mensaje
40         if (n < 0) // Verifica si hubo error
41             error("Error: recvfrom");
42
43         std::cout << "Esto se recibió: " << buffer << std::endl; // Imprime el
               mensaje recibido
44     }
45 }
```

```
46
47 int main(int argc, char *argv[]) // Función principal
48 {
49     int sock, n;
50     unsigned int length = sizeof(struct sockaddr_in);
51     char buffer[MSG_SIZE];
52     struct sockaddr_in anybody; // Estructura para almacenar la dirección del
           receptor
53     int boolval = 1; // Variable para configurar el socket
54     std::thread thread_rec; // Declaración del hilo
55     char IP_broadcast[IP_LENGTH]; // Variable para almacenar la dirección IP
56     FILE *file; // Declaración del archivo
57
58     if (argc != 2) // Verifica si se ingresó el puerto
59     {
60         std::cout << "Uso: " << argv[0] << " puerto" << std::endl;
61         exit(1);
62     }
63
64 #if OPCION_IP == 0 // Configuración de la dirección IP
65     std::strcpy(IP_broadcast, "192.168.90.255");
66 #endif
67
68 #if (OPCION_IP == 1) || (OPCION_IP == 2) // Configuración de la dirección IP
69
70 #if OPCION_IP == 1
71     system("ifconfig wlan0 | grep 'inet ' | awk '{ print $6 }' > ipaddr"); //
           Ejecutar comando para obtener la dirección de broadcast
72 #else
73     system("ifconfig enp0s31f6 | grep 'inet ' | awk '{ print $6 }' > ipaddr");
74 #endif
75
76     file = fopen("ipaddr", "r");    // Abrir archivo con la dirección de broadcast
77     if(file == NULL)
78     {
79         std::cout << "Error al abrir el archivo" << std::endl;;
80         exit(-1);
81     }
82     else
83     {
84         fscanf(file, "%s", IP_broadcast); // Leer la dirección de broadcast del
               archivo
85     }
86
87     fclose(file);                   // Cerrar archivo
88     system("rm ipaddr");            // Asegurarse de eliminar el archivo cuando se
           termine
89
90 #endif
91
92     std::cout << "La dirección de broadcast es: " << IP_broadcast << std::endl; //
           Imprime la dirección IP
93
94     anybody.sin_family = AF_INET; // Configura la familia de direcciones
95     anybody.sin_port = htons(atoi(argv[1])); // Configura el puerto
96     anybody.sin_addr.s_addr = htonl(INADDR_ANY); // Configura la dirección IP
97
98     sock = socket(AF_INET, SOCK_DGRAM, 0); // Crea el socket
99     if (sock < 0) // Verifica si hubo error
100        error("Error: socket");
101
102    if (bind(sock, reinterpret_cast<struct sockaddr *>(&anybody), sizeof(struct
           sockaddr_in)) < 0) // Asigna el socket
103    {
104        std::cout << "Error binding socket." << std::endl;
105        exit(-1);
106    }
107
```

```
108     if (setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &boolval, sizeof(boolval)) < 0)
            // Configura el socket
109     {
110         std::cout << "Error setting socket options." << std::endl;
111         exit(-1);
112     }
113
114     anybody.sin_addr.s_addr = inet_addr(IP_broadcast); // Configura la dirección IP
115
116     thread_rec = std::thread(receiving, sock); // Crea el hilo
117
118     std::cout << "Este programa despliega lo que sea que reciba." << std::endl;
119     std::cout << "También transmite lo que el usuario ingrese, máx. " << MSG_SIZE <<
            " caracteres. (! para salir):" << std::endl;
120
121     do // Ciclo para enviar mensajes
122     {
123         memset(buffer, 0, MSG_SIZE); // Limpia el buffer
124         std::cin.getline(buffer, MSG_SIZE - 1); // Lee la entrada del usuario
125
126         if (buffer[0] != '!') // Verifica si el usuario quiere salir
127         {
128             n = sendto(sock, buffer, strlen(buffer), 0, reinterpret_cast<const
                    struct sockaddr *>(&anybody), length); // Envía el mensaje
129             if (n < 0) // Verifica si hubo error
130                 error("Error: sendto");
131         }
132     } while (buffer[0] != '!');
133
134     close(sock); // Cierra el socket
135     thread_rec.join(); // Espera a que el hilo termine
136
137     return 0;
138 }
```

Cuadro 18: Código Lab8_cliente.cpp

CAPÍTULO 8

Traducción de proyecto de lenguaje C a C++

8.1. Objetivos y descripción del proyecto

El objetivo principal del proyecto consiste en el uso de sistemas de Supervisión, Control y Adquisición de Datos (SCADA).

El proyecto consiste en la implementación y verificación de una arquitectura de *hardware* y *software* para el uso de un sistema SCADA simplificado. Este sistema SCADA se debe realizar con múltiples UTRs (Unidad de Terminal Remota), líneas de transmisión y mediciones. Por lo tanto, el sistema debe construirse de la siguiente manera: N UTRs con entradas/salidas digitales y entradas analógicas para monitorear interruptores o botones, encender/apagar componentes y obtener información de sensores, entre otros. Estos se conectarán con el historiador y le proporcionarán actualizaciones periódicas del estado de sus subsistemas correspondientes. De igual manera, los UTR deben ser capaces de recibir comandos del historiador. En el historiador se deben registrar todos los eventos en la secuencia en que sucedan. Además del uso de las Raspberry Pi para este proyecto, se utilizará adicionalmente un Arduino IoT, el cual contará con un programa cuya función es simplemente encender un LED y enviar la información al UTR mediante I^2C.

El primer código que se desarrolló fue el del historiador, el cual es capaz de enviar comandos a los UTR mediante una conexión de red y, además, debe recibir información de eventos de los UTR. En este proyecto se utilizan 2 o 1 UTRs, en caso de que se llegaran a utilizar más, es necesario configurar este código para que pueda leer N UTRs extras que se agreguen al sistema. Las librerías utilizadas en este código son las mismas que se han estado utilizando en los laboratorios mencionados en el capítulo 7. El código del historiador quedó de la siguiente manera:

```
1 /*
```

```
 ============================================================================
 Nombre: Historiador.cpp
 Autor:  Rodrigo José García Ambrosy
 Proyecto
 ============================================================================
 */
#include <iostream> // Librería estándar de entrada/salida
#include <thread>   // Librería para utilizar hilos
#include <chrono>   // Librería para utilizar la función sleep_for
#include <cstdio>  //Librería estándar de entrada/salida
#include <cstdlib> //Librería estándar para funciones generales
#include <cstring> //Librería para la manipulación de cadenas
#include <unistd.h> // Librería que proporciona funciones y constantes específicas de sistemas Unix.
#include <sys/types.h> // Librería para definir tipos de datos
#include <sys/socket.h> // Librería para utilizar sockets
#include <netinet/in.h> // Librería para utilizar direcciones de internet
#include <arpa/inet.h>  // Librería para manipular direcciones IP
#include <fstream>      //Librería para utilizar archivos


#define MSG_SIZE 3000 // Tamaño del mensaje
#define IP "192.168.1.255" // Dirección IP
#define OPCION_IP 0 /// 0 - hard coded
#define IP_LENGTH 15 // Tamaño de la dirección IP
int sockfd, n; // Descriptores de archivo
unsigned int length = sizeof(struct sockaddr_in);// Tamaño de la estructura sockaddr_in
struct sockaddr_in RTU, from1; // Estructuras para almacenar las direcciones de los sockets
char buffer[MSG_SIZE];  // Buffer para almacenar el mensaje
int boolval = 1;        // Opción de socket
char IP_broadcast[IP_LENGTH];

bool shouldExit = false;

void enviar(void*ptr); // Declaración de la función enviar

void error(const char *msg) // Función para imprimir errores
{
    perror(msg);
    exit(0);
}


void receiving(int sock, std::ofstream& outputFile) // Función para recibir mensajes
{
    char buffer[MSG_SIZE]; // Buffer para almacenar el mensaje
    struct sockaddr_in from; // Estructura para almacenar la dirección del emisor

    while (!shouldExit)
    {
        memset(buffer, 0, MSG_SIZE);
        int n = recvfrom(sock, buffer, MSG_SIZE, 0, reinterpret_cast<struct sockaddr *>(&from), &length); // Recibe el mensaje
        if (n < 0) // Verifica si hubo error
            error("Error: recvfrom");

        std::cout << "Esto se recibió: " << buffer << std::endl; // Imprime el mensaje recibido
        outputFile << buffer << std::endl; // Escribe el mensaje en el archivo de salida
        if (std::strcmp(buffer, "!") == 0) // Verifica si el mensaje es "!"
        {
            shouldExit = true; // Establece shouldExit en true para salir del programa
            break;
        }
```

```
63      }
64  }
65
66
67  int main(int argc, char *argv[]) // Función principal
68  {
69      std::ofstream outputFile("Lista de Eventos.txt");
70
71      // Revisa si el archivo se abrió correctamente
72      if (!outputFile.is_open()) {
73          std::cerr << "Error opening the file." << std::endl; // Imprime un mensaje
                de error en caso de que suceda
74          return 1;
75      }
76
77      std::thread hilo1; // Declaración del hilo
78
79      if (argc != 2) // Verifica si se ingresó el puerto
80      {
81          std::cout << "Uso: " << argv[0] << " <puerto>" << std::endl;
82          return 0;
83      }
84  #if OPCION_IP == 0 // Configuración de la dirección IP
85      std::strcpy(IP_broadcast, "192.168.1.255");
86  #endif
87
88      std::cout << "La dirección de broadcast es: " << IP_broadcast << std::endl;
89
90      sockfd = socket(AF_INET, SOCK_DGRAM, 0); // Crea el socket. Es sin conexión.
91      if (sockfd < 0) error("ERROR al abrir el socket");
92
93      RTU.sin_family = AF_INET; // constante de símbolo para dominio de Internet
94      RTU.sin_port = htons(atoi(argv[1])); // campo de puerto
95      //RTU.sin_addr.s_addr = inet_addr(IP); // campo de dirección IP. Para el
            servidor, será la dirección IP de la máquina en la que se ejecuta este
            programa.
96
97
98      if (bind(sockfd, reinterpret_cast<struct sockaddr *>(&RTU), length) < 0) //
            Asigna el socket
99      {
100         std::cout << "Error binding socket." << std::endl;
101         exit(-1);
102     }
103
104     if (setsockopt(sockfd, SOL_SOCKET, SO_BROADCAST, &boolval, sizeof(boolval)) < 0)
            // Configura el socket
105     {
106         std::cout << "error al configurar opciones del socket" << std::endl;
107         exit(-1);
108     }
109     RTU.sin_addr.s_addr = inet_addr(IP);
110
111     hilo1 = std::thread(enviar, (void *)0); // Crea el hilo
112
113     std::cout << "Los comandos son los siguientes:" << std::endl; // Imprime los
            comandos
114     std::cout << "RTU# LED# 0 o 1" << std::endl;
115     std::cout << "RTU# LEDIoT 0 o 1" << std::endl;
116     std::cout << "Utilizar ! para salir: " << std::endl;
117     //RTU.sin_addr.s_addr = inet_addr(IP);
118     char CONEXION[128]; // Variable para almacenar el mensaje
119
120     std::strcpy(CONEXION, "Conexión"); // Copia el mensaje en la variable
121     n = sendto(sockfd, CONEXION, strlen(CONEXION), 0, (const struct sockaddr *)&RTU,
            length); // Envia el mensaje
122     if (n < 0){
123         std::cerr << "ERROR al enviar" << std::endl; // Verifica si hubo error
```

```
124     }
125     while (!shouldExit)
126     {
127         memset(buffer, 0, MSG_SIZE);
128         receiving(sockfd, outputFile);
129     }
130     hilo1.join();
131     outputFile.close(); // Cierra el archivo de salida
132     std::cout << "Saliendo del Programa" << std::endl;
133     return 0;
134 }
135 
136 void enviar(void*ptr) // Función para enviar mensajes
137 {
138     while(!shouldExit){
139         memset(buffer, 0, MSG_SIZE); // Limpia el buffer
140         std::cin.getline(buffer, MSG_SIZE - 1); // Lee el mensaje
141 
142         std::cout << "Mandando mensaje: " << buffer << std::endl;
143         n = sendto(sockfd, buffer, strlen(buffer), 0,
144         (const struct sockaddr *)&RTU, length);
145         if (n < 0) error("ERROR sendto");
146 
147         if((std::strcmp(buffer, "RTU1 LED1 1\n")) == 0){ // Verifica si se ingresó
                el comando para encender el LED1 de la RTU1
148             n = sendto(sockfd, buffer, strlen(buffer), 0,
149             (const struct sockaddr *)&RTU, length);
150             if (n < 0) error("ERROR sendto");
151         }
152         if((std::strcmp(buffer, "RTU1 LED1 0\n")) == 0){ // Verifica si se ingresó
                el comando para apagar el LED1 de la RTU1
153             n = sendto(sockfd, buffer, strlen(buffer), 0,
154             (const struct sockaddr *)&RTU, length);
155             if (n < 0) error("ERROR sendto");
156         }
157         if((std::strcmp(buffer, "RTU1 LED2 1\n")) == 0){ // Verifica si se ingresó
                el comando para encender el LED2 de la RTU1
158             n = sendto(sockfd, buffer, strlen(buffer), 0,
159             (const struct sockaddr *)&RTU, length);
160             if (n < 0) error("ERROR sendto");
161         }
162         if((std::strcmp(buffer, "RTU1 LED2 0\n")) == 0){ // Verifica si se ingresó
                el comando para apagar el LED2 de la RTU1
163             n = sendto(sockfd, buffer, strlen(buffer), 0,
164             (const struct sockaddr *)&RTU, length);
165             if (n < 0) error("ERROR sendto");
166         }
167         if((std::strcmp(buffer, "RTU2 LED1 1\n")) == 0){ // Verifica si se ingresó
                el comando para encender el LED1 de la RTU2
168             n = sendto(sockfd, buffer, strlen(buffer), 0,
169             (const struct sockaddr *)&RTU, length);
170             if (n < 0) error("ERROR sendto");
171         }
172         if((std::strcmp(buffer, "RTU2 LED1 0\n")) == 0){ // Verifica si se ingresó
                el comando para apagar el LED1 de la RTU2
173             n = sendto(sockfd, buffer, strlen(buffer), 0,
174             (const struct sockaddr *)&RTU, length);
175             if (n < 0) error("ERROR sendto");
176         }
177         if((std::strcmp(buffer, "RTU2 LED2 1\n")) == 0){ // Verifica si se ingresó
                el comando para encender el LED2 de la RTU2
178             n = sendto(sockfd, buffer, strlen(buffer), 0,
179             (const struct sockaddr *)&RTU, length);
180             if (n < 0) error("ERROR sendto");
181         }
182         if((std::strcmp(buffer, "RTU2 LED2 0\n")) == 0){ // Verifica si se ingresó
                el comando para apagar el LED2 de la RTU2
183             n = sendto(sockfd, buffer, strlen(buffer), 0,
```

```
184             (const struct sockaddr *)&RTU, length);
185             if (n < 0) error("ERROR sendto");
186         }
187         if((std::strcmp(buffer, "RTU1 LEDIoT 1\n")) == 0){ // Verifica si se ingresó
                el comando para apagar el LEDIOT de la RTU1
188             n = sendto(sockfd, buffer, strlen(buffer), 0,
189             (const struct sockaddr *)&RTU, length);
190             if (n < 0) error("ERROR sendto");
191         }
192         if((std::strcmp(buffer, "RTU1 LEDIoT 0\n")) == 0){ // Verifica si se ingresó
                el comando para apagar el LEDIOT de la RTU1
193             n = sendto(sockfd, buffer, strlen(buffer), 0,
194             (const struct sockaddr *)&RTU, length);
195             if (n < 0) error("ERROR sendto");
196         }
197         if((std::strcmp(buffer, "RTU2 LEDIoT 1\n")) == 0){ // Verifica si se ingresó
                el comando para apagar el LEDIOT de la RTU2
198             n = sendto(sockfd, buffer, strlen(buffer), 0,
199             (const struct sockaddr *)&RTU, length);
200             if (n < 0) error("ERROR sendto");
201         }
202         if((std::strcmp(buffer, "RTU2 LEDIoT 0\n")) == 0){ // Verifica si se ingresó
                el comando para apagar el LEDIOT de la RTU2
203             n = sendto(sockfd, buffer, strlen(buffer), 0,
204             (const struct sockaddr *)&RTU, length);
205             if (n < 0) error("ERROR sendto");
206         }
207     }
208 }
```

Cuadro 19: Código Historiador.cpp

El código desarrollado para los UTR es capaz de recibir comandos del historiador y enviar, de forma periódica o constante, la información de los eventos que ocurran, por ejemplo: el encendido o apagado de un LED desde el historiador o la pulsación de un botón conectado al UTR. En el UTR, se configuraron interrupciones específicas de la librería wiringPi para manejar los botones. Para establecer la conexión entre el Arduino IoT y el UTR, fue necesario configurar I^2C mediante wiringPi. Luego, para utilizar el ADC y obtener el valor del voltaje, se configuró SPI en el UTR para que se conecte al integrado MCP3002, el cual funciona como un convertidor de analógico a digital. De esta manera, el UTR puede procesar los datos y leerlos. Además, se crearon tres clases que manejan el funcionamiento de los LED, los botones y los switches. Cada evento debe ser clasificado para que el historiador conozca qué sucedió y cuándo. Esto se logró de tal manera que, por ejemplo, el evento 0 indica la ausencia de actividad mientras que el evento 1 señala que se presionó el botón 1. El historiador recibirá líneas que describen todo lo que sucede en el UTR siguiendo este formato: 'Número de UTR, evento, fecha hora, estado inicial de cada evento, voltaje ADC'. El código de los UTR es el mismo, la única modificación depende del número de UTR para identificar correctamente cuál envía la información al historiador. Todos estos eventos, al momento de que se cierre el programa del historiador, quedan guardados en un archivo de texto en donde se documenta lo que sucedió. La Figura 6 muestra la información de eventos que se encuentra en el archivo de texto generado al finalizar una sesión.

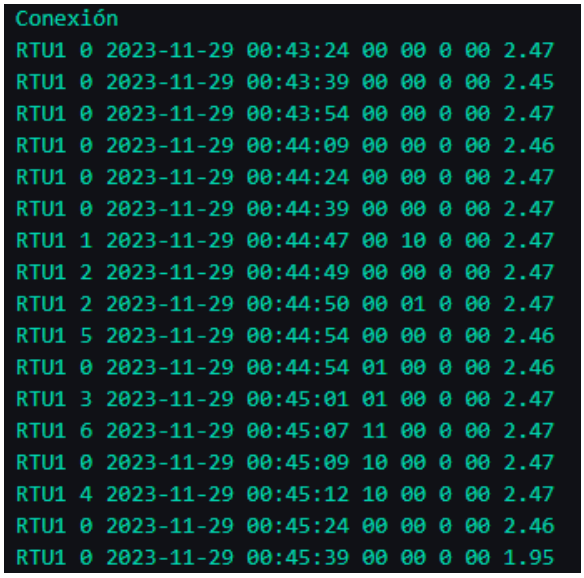


Figura 6: Listado de eventos guardados

Como se menciona anteriormente cada acción que lea el UTR va a documentar un evento, así como se mira en la figura 5. La siguiente lista describe que significa cada evento:

- Evento 0: No sucede nada en el sistema.
- Evento 1: Botón 1 presionado.
- Evento 2: Botón 2 presionado.
- Evento 3: Switch 1 cambia a posición 'ON'.
- Evento 4: Switch 1 cambia a posición 'OFF'.
- Evento 5: Switch 2 cambia a posición 'ON'
- Evento 6: Switch 2 cambia a posición 'OFF'
- Evento 7: Se enciende alarma detectando voltaje menor a 0.5V.
- Evento 8: Se enciende alarma detectando voltaje mayor a 2.5V.
- Evento 9: Led 1 se enciende.
- Evento 10: Led 1 se apaga.
- Evento 11: Led 2 se enciende.
- Evento 12: Led 2 se apaga.
- Evento 13: Led integrado del IoT se enciende.
- Evento 14: Led integrado del IoT se apaga.

El código del UTR queda de la siguiente manera:

```
#include <wiringPi.h> // Librería para controlar pines GPIO
#include <iostream> // Librería estándar de entrada/salida
#include <thread>   // Librería para utilizar hilos
#include <chrono>   // Librería para utilizar la función sleep_for
#include <cstdio>   // Librería estándar de entrada/salida
#include <cstdlib>  // Librería estándar para funciones generales
#include <ctime>    // Librería para utilizar la función time
#include <cstring>  // Librería para la manipulación de cadenas
#include <cstdint>  // Librería para utilizar tipos de datos enteros
#include <wiringPiSPI.h> // Librería para utilizar el bus SPI
#include <wiringPiI2C.h> // Librería para utilizar el bus I2C
#include <string>   // Librería para utilizar cadenas
#include <sys/socket.h>  // Librería para utilizar sockets
#include <arpa/inet.h>   // Librería para manipular direcciones IP
#include <sys/time.h>   // Librería para utilizar la función gettimeofday
#include <unistd.h>    // Librería que proporciona funciones y constantes
    específicas de sistemas Unix.
#include <vector>   // Librería para utilizar vectores

#define SPI_CHANNEL       0 // Canal SPI de la Raspberry Pi, 0 ó 1
#define SPI_SPEED   1500000 // Velocidad de la comunicación SPI (reloj, en HZ)
                            // Máxima de 3.6 MHz con VDD = 5V, 1.2 MHz con VDD = 2.7V
#define ADC_CHANNEL       0 // Canal A/D del MCP3002 a usar, 0 ó 1
#define MSG_SIZE 301     // Tamaño del mensaje
#define IP "192.168.1.255" // Dirección IP

#define LUZ_1   24  // Pin para la luz 1
#define LUZ_2   25  // Pin para la luz 2
#define BTN1    17   // Pin para el botón 1
#define BTN2    26  // Pin para el botón 2
#define Swtch1  22  // Pin para el switch 1
#define Swtch2  23  // Pin para el switch 2
#define Alarm   16  // Pin para la alarma/buzzer

char timestamp_str[20];

// Variables globales
int switch1, switch2, PUSH1, PUSH2, status_led_1, LD1, LD2;
float voltajeadc = 0;

const int MAX_NEVENTOS = 100; // Número máximo de eventos
char eventos_pendientes[MAX_NEVENTOS][256]; // Número de eventos pendientes
int n_eventos_pendientes_envio = 0; // Número de eventos pendientes de envío
uint32_t numero_eventos=0; // Número de eventos
uint32_t time_on=0; // Tiempo encendido

int sockfd, n; // Variables para el socket
unsigned int length;
struct sockaddr_in addr, broadcast_addr; // Estructuras para almacenar las
    direcciones de los sockets
char buffer[MSG_SIZE]; // Buffer para almacenar el mensaje
int boolval = 1;

char message[128];

uint16_t get_ADC(int channel); //Función para obtener el valor del ADC

void RTU1(int sockfd, sockaddr_in& addr, socklen_t& length); // Declaración de la
    función RTU1
void agregar_evento(uint8_t evento_id); // Declaración de la función agregar_evento

void error(const char *msg) // Función para imprimir errores
{
    perror(msg);
    exit(0);
```

```
}


class LED { // Clase para controlar LEDs
    public:
        LED(int pin) {
            this->pin = pin;
            pinMode(pin, OUTPUT);
        }

        void on() {
            digitalWrite(pin, HIGH);
        }

        void off() {
            digitalWrite(pin, LOW);
        }

    private:
        int pin;
    };

class Button { // Clase para controlar botones
    public:
        Button(int pin, int eventID) : pin(pin), eventID(eventID) {
            pinMode(pin, INPUT);
            pullUpDnControl(pin, PUD_DOWN);
            wiringPiISR(pin, INT_EDGE_FALLING, &Button::isrWrapper); // Configura la
                interrupción
            instance = this;
        }

        bool isPressed() {
            return digitalRead(pin) == HIGH;
        }

        void waitForButtonRelease() {
            while (isPressed()) {
                std::this_thread::sleep_for(std::chrono::milliseconds(1));
            }
        }

        void isr() { // Función para la interrupción
            agregar_evento(eventID);
        }

    private:
        int pin;
        int eventID;
        static Button* instance;

        // Wrapper para llamar a la función de interrupción de la clase
        static void isrWrapper() {
            if (instance != nullptr) {
                instance->isr();
            }
        }
};

Button* Button::instance = nullptr; // Inicio de la variable estática

class Button2 { // Clase para controlar botones
    public:
        Button2(int pin, int eventID) : pin(pin), eventID(eventID) {
            pinMode(pin, INPUT);
            pullUpDnControl(pin, PUD_DOWN);
            wiringPiISR(pin, INT_EDGE_FALLING, &Button2::isrWrapper); // Configura
                la interrupción
```

```
129             instance = this;
130         }
131 
132         bool isPressed() {
133             return digitalRead(pin) == HIGH;
134         }
135 
136         void waitForButtonRelease() {
137             while (isPressed()) {
138                 std::this_thread::sleep_for(std::chrono::milliseconds(1));
139             }
140         }
141 
142         void isr() { // Función para la interrupción
143             agregar_evento(eventID);
144         }
145 
146     private:
147         int pin;
148         int eventID;
149         static Button2* instance;
150 
151         // Wrapper para llamar a la función de interrupción de la clase
152         static void isrWrapper() {
153             if (instance != nullptr) {
154                 instance->isr();
155             }
156         }
157 };
158 
159 Button2* Button2::instance = nullptr; // Inicio de la variable estática
160 
161 class Switch { // Clase para controlar switches
162     public:
163         Switch(int pin, int eventID, int& globalSwitch) : pin(pin),
            eventID(eventID), globalSwitch(globalSwitch) { // Constructor
164             pinMode(pin, INPUT);
165             pullUpDnControl(pin, PUD_UP);
166         }
167 
168         void checkState() { // Función para verificar el estado del switch
169             int currentState = digitalRead(pin);
170             if (currentState != previousState) {
171 
172                 switch (currentState) { // Verifica si el switch está encendido o
                        apagado
173                     case HIGH:
174                         agregar_evento(eventID + 1);
175                         globalSwitch = 0;
176                         break;
177                     case LOW:
178                         agregar_evento(eventID);
179                         globalSwitch = 1;
180                         break;
181                 }
182                 previousState = currentState; // Actualiza el estado anterior
183             }
184         }
185 
186     private:
187         int pin;
188         int eventID;
189         int& globalSwitch;
190         int previousState = HIGH;  // Assuming the initial state is HIGH
191 };
192 
193 //Funciones para orden de eventos
194 void listar_eventos(){
```

```
195     int i;
196     printf("listado eventos\n");
197     for (i=0; i< n_eventos_pendientes_envio; i++){
198 
199         std::cout << eventos_pendientes[i] << std::endl; // Imprime los eventos
                pendientes
200 
201         //Manda los eventos pendientes
202         n = sendto(sockfd, eventos_pendientes[i], MAX_NEVENTOS, 0,
203         (struct sockaddr *)&addr, sizeof(struct sockaddr));
204         if(n < 0)
205             std::cerr << "Error en sendto" << std::endl; //Verifica si hubo error
206         }
207     n_eventos_pendientes_envio = 0; // Reinicia el número de eventos pendientes de
            envío
208 }
209 
210 
211 void agregar_evento(uint8_t evento_id) { // Función para agregar eventos
212     struct timeval current_time; // Estructura para almacenar el tiempo actual
213     gettimeofday(&current_time, NULL); // Obtiene el tiempo actual
214 
215     struct timeval tv;
216     gettimeofday(&tv, NULL);
217     time_t now = tv.tv_sec;
218     struct tm* timeinfo = localtime(&now); // Obtiene la fecha y hora actual
219     char timestamp[80];
220     strftime(timestamp, 80, "%Y-%m-%d %H:%M:%S", timeinfo); // Formato de la fecha y
            hora
221 
222     numero_eventos++;
223 
224     // Formato del mensaje
225     std::sprintf(message, "RTU1 %d %s %d%d %d%d %d %d%d %0.2f",
226              evento_id, timestamp,
227              switch1, switch2, PUSH1, PUSH2, status_led_1, LD1, LD2, voltajeadc);
228 
229     std::cout << "Mandando Mensaje: " << message << std::endl; // Imprime el mensaje
230 
231     // Se copia el mensaje en el arreglo de eventos pendientes
232     if (n_eventos_pendientes_envio < MAX_NEVENTOS) {
233         std::strcpy(eventos_pendientes[n_eventos_pendientes_envio], message);
234         n_eventos_pendientes_envio++;
235     }
236 
237 };
238 
239 int main(int argc, char *argv[]) { // Función principal
240     wiringPiSetupGpio(); // Inicializa la librería wiringPi
241     pinMode(Alarm, OUTPUT); // Configura el pin de la alarma como salida
242     uint16_t ADCvalue; // Variable para almacenar el valor del ADC
243     int last_alarm_statusH=0; // Variable para almacenar el estado de la alarma
244     int last_alarm_statusL=0;
245 
246     // Configuración del bus SPI
247     if (wiringPiSPISetup(SPI_CHANNEL, SPI_SPEED) < 0) {
248         std::cerr << "Error setting up SPI" << std::endl;
249         return 1;  // Verifica si hubo error
250     }
251     //Se crean los objetos de las clases
252     LED luz1(LUZ_1);
253     LED luz2(LUZ_2);
254     Button btn1(BTN1, 1);
255     Button2 btn2(BTN2, 2);
256     Switch swtch1(Swtch1,3,switch1);
257     Switch swtch2(Swtch2,5, switch2);
258     digitalWrite(Alarm, LOW);
259 
```

```
260     if(argc != 2) // Verifica si se ingresó el puerto
261      {
262          std::cout << "Usage: " << argv[0] << " [port]" << std::endl;
263          exit(1);
264      }
265
266
267      sockfd = socket(AF_INET, SOCK_DGRAM, 0); // Crea socket UDP sin conexión
268      if(sockfd < 0)
269          std::cerr << "Opening socket" << std::endl; // Verifica si hubo error al
                 crear el socket
270
271      addr.sin_family = AF_INET; // Configura la familia de direcciones
272      addr.sin_port = htons(atoi(argv[1]));    // Configura el puerto
273
274      length = sizeof(addr); // Tamaño de la dirección
275
276      if(bind(sockfd, (struct sockaddr *)&addr, length) < 0) // Asigna el socket
277          std::cerr << "Error binding socket." << std::endl;
278
279      // Cambia los permisos del socket para permitir el broadcast
280      if(setsockopt(sockfd, SOL_SOCKET, SO_BROADCAST, &boolval, sizeof(boolval)) < 0)
281          std::cerr << "Error setting socket options\n" << std::endl;
282
283      addr.sin_addr.s_addr = inet_addr(IP); // Configura la dirección IP
284
285      std::thread rtuThread(RTU1, sockfd, std::ref(addr), std::ref(length)); // Crea
             el hilo de la función RTU1
286
287      while (true) { // Ciclo infinito
288          time_on++; // Incrementa el tiempo encendido
289          ADCvalue =get_ADC(ADC_CHANNEL); // Obtiene el valor del ADC
290          PUSH1 = 0;
291          PUSH2 = 0;
292          if (btn1.isPressed()) { // Verifica si el botón 1 está presionado
293              PUSH1 = 1;
294          }
295          if (btn2.isPressed()) { // Verifica si el botón 2 está presionado
296              PUSH2 = 1;
297          }
298          swtch1.checkState(); // Verifica el estado del switch 1
299          swtch2.checkState(); // Verifica el estado del switch 2
300
301          //Condicionales para la alarma con el ADC
302          if (voltajeadc < 0.5) {
303              if(last_alarm_statusL == 0){
304                  agregar_evento(7);
305                  last_alarm_statusL = 1;
306                  digitalWrite(Alarm, HIGH);
307              }
308          } else if (voltajeadc > 2.5) {
309              if(last_alarm_statusH == 0){
310                  agregar_evento(8);
311                  last_alarm_statusH = 1;
312                  digitalWrite(Alarm, HIGH);
313              }
314          }else if ((voltajeadc < 2.5)&&(voltajeadc > 0.5)){
315              digitalWrite(Alarm, LOW);
316              last_alarm_statusL = 0;
317              last_alarm_statusH = 0;
318          }
319
320          if ((time_on %15) == 0){ //Cada 15 segundos se agrega un evento
321              agregar_evento (0);
322          }
323          fflush(stdout);
324          if ((time_on % 15)==0){ //Cada 15 segundos se imprime el número de eventos
325             listar_eventos(); // Llama a la función para listar los eventos
```

```
326             if (n_eventos_pendientes_envio >0){ // Verifica si hay eventos
                    pendientes de envío
327                 std::cout << "Hay eventos pendientes de envío" << std::endl;
328             }
329         }
330 
331         std::this_thread::sleep_for(std::chrono::seconds(1)); // Espera 1 segundo
332     }
333 
334     rtuThread.join(); // Espera a que termine el hilo
335     return 0;
336 }
337 
338 void RTU1(int sock, sockaddr_in& addres, socklen_t& leng){ // Función para la RTU1
339     LED luz1(LUZ_1);
340     LED luz2(LUZ_2);
341     int i2cAddress = 0x08; // Dirección del dispositivo I2C
342 
343     // Comunicación I2C abierta
344     int i2cHandle = wiringPiI2CSetup(i2cAddress);
345     if (i2cHandle == -1) {
346         std::cerr << "Error opening I2C communication." << std::endl; // Verifica si
                hubo error
347         return;
348     }
349 
350     while (1){
351 
352         memset(buffer, 0, MSG_SIZE); // Limpia el buffer
353         n = recvfrom(sock, buffer, MSG_SIZE, 0, (struct sockaddr*)&addres, &leng);
                // Recibe el mensaje
354         if (n < 0){
355             perror("ERROR reading from socket (revfrom)"); // Verifica si hubo error
356         }
357         std::cout << "Message recibido (" << n << " bytes): " << buffer <<
                std::endl; // Imprime el mensaje recibido
358 
359 
360         if((std::strcmp(buffer, "RTU1 LED1 1") == 0)){ // Verifica si el mensaje es
                para encender el LED 1
361             char mensaje[60];
362             luz1.on();
363             LD1 = 1;
364             agregar_evento(9); // Agrega el evento 9
365         }
366         if((std::strcmp(buffer, "RTU1 LED1 0") == 0)){ // Verifica si el mensaje es
                para apagar el LED 1
367             char mensaje[60];
368             luz1.off();
369             LD1 = 0;
370             agregar_evento(10);
371         }
372         if((std::strcmp(buffer, "RTU1 LED2 1") == 0)){ // Verifica si el mensaje es
                para encender el LED 2
373             char mensaje[60];
374             luz2.on();
375             LD2 = 1;
376             agregar_evento(11);
377         }
378         if((std::strcmp(buffer, "RTU1 LED2 0") == 0)){ // Verifica si el mensaje es
                para apagar el LED 2
379             char mensaje[60];
380             luz2.off();
381             LD2 = 0;
382             agregar_evento(12);
383         }
384         if((std::strcmp(buffer, "RTU1 LEDIoT 1") == 0)){ // Verifica si el mensaje
                es para encender el LED IoT
```

```
385             char mensaje[60];
386             char data1[] = "A";
387             status_led_1 = 1;
388             agregar_evento(13);
389             write(i2cHandle, data1, sizeof(data1)); // Escribe en el bus I2C para
                    mandar el mensaje al Arduino
390         }
391         if((std::strcmp(buffer, "RTU1 LEDIoT 0") == 0)){ // Verifica si el mensaje
                es para apagar el LED IoT
392             char mensaje[60];
393             char data0[] = "B";
394             status_led_1 = 0;
395             agregar_evento(14);
396             write(i2cHandle, data0, sizeof(data0)); // Escribe en el bus I2C para
                    mandar el mensaje al Arduino
397         }
398     }
399     close(i2cHandle); // Cierra la comunicación I2C
400 }
401
402 uint16_t get_ADC(int ADC_chan)
403 {
404     uint8_t spiData[2]; // La comunicación usa dos bytes
405     uint16_t resultado;
406
407     // Asegurarse que el canal sea válido. Si lo que viene no es válido, usar canal
            0.
408     if((ADC_chan < 0) || (ADC_chan > 1))
409         ADC_chan = 0;
410
411     // Construimos el byte de configuración: 0, start bit, modo, canal, MSBF:
            01MC1000
412     spiData[0] = 0b01101000 | (ADC_chan << 4);  // M = 1 ==> "single ended"
413                                                 // C: canal: 0 ó 1
414     spiData[1] = 0; // "Don't care", este valor no importa.
415
416     // La siguiente función realiza la transacción de escritura/lectura sobre el bus
            SPI
417     // seleccionado. Los datos que estaban en el buffer spiData se sobreescriben por
418     // los datos que vienen por SPI.
419     wiringPiSPIDataRW(SPI_CHANNEL, spiData, 2); // 2 bytes
420
421     // spiData[0] y spiData[1] tienen el resultado (2 bits y 8 bits, respectivamente)
422     resultado = (spiData[0] << 8) | spiData[1];
423     voltajeadc=(resultado*3.3)/1023.0;
424     return(resultado);
425 }
```

Cuadro 20: Código UTR.cpp

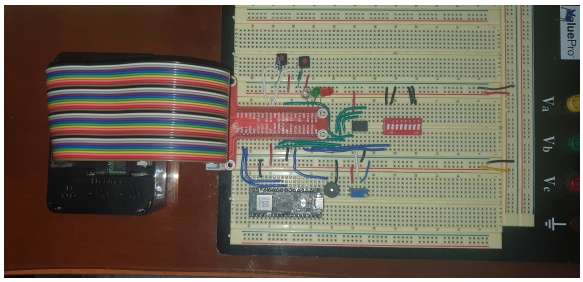


Figura 7: Raspberry Pi3 como UTR

El código del Arduino IoT es mucho más sencillo en comparación con el UTR y el historiador, ya que solo es necesario configurar I^2C por medio de la librería *Wire.h*. El código queda de la siguiente forma:

```
#include <Wire.h> // Incluye la librería Wire (I2C)

const int BUFFER_SIZE = 64; // Define el tamaño del buffer
char buffer[BUFFER_SIZE]; // Define el buffer
int bufferIndex = 0; // Define el índice del buffer

void receiveEvent(int howMany) { // Función para recibir datos
  while (Wire.available()) { // Mientras haya datos disponibles
    char receivedChar = Wire.read(); // Lee el dato recibido


    if (bufferIndex < BUFFER_SIZE - 1) { // Si el índice del buffer es menor al
        tamaño del buffer
      buffer[bufferIndex++] = receivedChar;
      buffer[bufferIndex] = '\0'; // Agrega el caracter nulo al final del buffer
    }
  }
}

void processReceivedData() { // Función para procesar los datos recibidos
  if (bufferIndex > 0) {
    // Imprime los datos recibidos
    Serial.print("Received data: ");
    Serial.println(buffer); // Imprime el buffer

    // Chequea si el dato recibido es 'A' o 'B'
    if (buffer[0] == 'A') {
      Serial.println("Turning on the built-in LED."); // Imprime el mensaje
      digitalWrite(LED_BUILTIN, HIGH); // Enciende el LED
    } else if (buffer[0] == 'B') {
      Serial.println("Turning off the built-in LED."); // Imprime el mensaje
      digitalWrite(LED_BUILTIN, LOW); // Apaga el LED
    }

    // Limpia el buffer
    bufferIndex = 0;
    buffer[0] = '\0';
  }
}

void setup() { // Función de configuración
  Wire.begin(0x08); // Inicializa el dispositivo como esclavo con la dirección 0x08
  Wire.onReceive(receiveEvent); // Registra la función para recibir datos
  Serial.begin(9600); // Inicializa el puerto serial

  pinMode(LED_BUILTIN, OUTPUT); // Configura el pin del LED como salida
}

void loop() {
  processReceivedData(); // Procesa los datos recibidos
  delay(100); // Espera 100 milisegundos
}
```

Cuadro 21: Código Arduino Nano IoT.cpp

La topología que utiliza este proyecto es la siguiente:

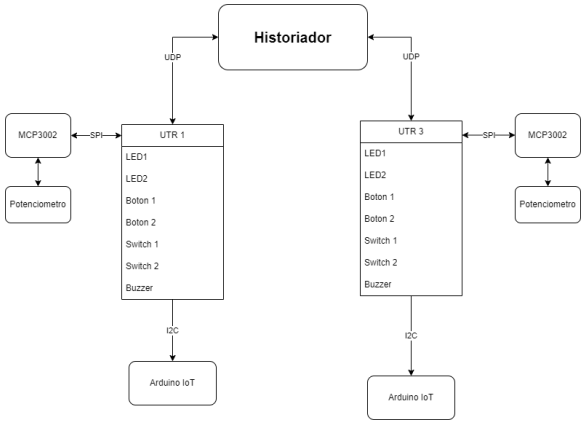


Figura 8: Topología del sistema SCADA

8.2. Interfaz gráfica

En el proyecto original, no era necesario realizar una interfaz gráfica, además de que las Raspberry Pi 3 no permiten su desarrollo debido a limitaciones del sistema operativo. En este caso, la interfaz gráfica se realizo en una máquina virtual con el sistema operativo Ubuntu. La interfaz se diseño utilizando Qt, la cual tiene un IDE llamado Qt Creator, que permite desarrollar interfaces gráficas en C++ y Python.

La única diferencia entre el uso de esta interfaz gráfica y el código del historiador es que la interfaz no puede generar un archivo .txt que contenga todos los eventos que ocurren, por lo que esto puede ser realizado en un trabajo futuro. Todas las capacidades del código del historiador se mantienen, inclusive se emplean bibliotecas diseñadas para gestionar clases un tanto más complejas, como una para el funcionamiento interno de la interfaz y otra para la gestión de la comunicación mediante *sockets* utilizando el protocolo UDP. La interfaz se ilustra en la Figura 9:

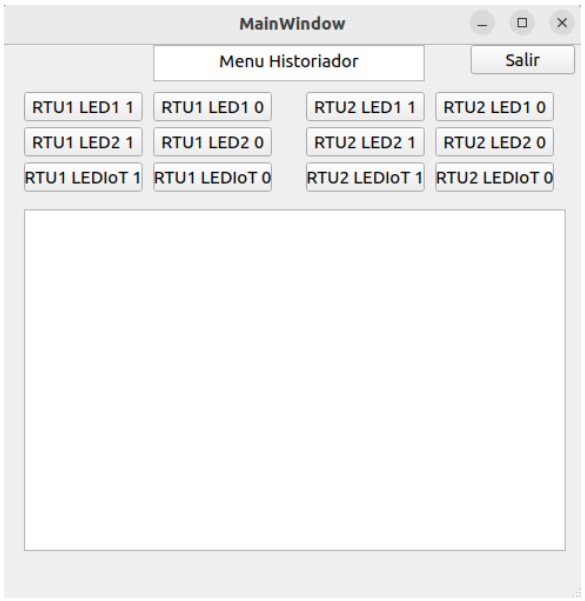


Figura 9: Interfaz gráfica

Como se menciona anteriormente la interfaz se diseño en la aplicación llamada Qt Creator, la siguiente imagen muestra el Entrono de Desarrollo Integrado en donde se diseña la interfaz gráfica con la que se trabajara.

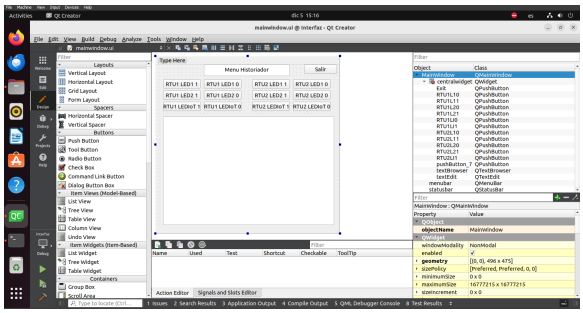


Figura 10: Diseño de interfaz en Qt Creator

La interfaz utiliza varios fragmentos de código distribuidos en diferentes componentes, tales como el código que define la estructura de la interfaz, el código de configuración necesario para compilar el programa, el código principal y los códigos creados para generar bibliotecas con las funciones de las clases correspondientes.

Qt Creator se encarga de compilar automáticamente todos los códigos utilizados en el proyecto, eliminando la necesidad de recurrir a la terminal para la compilación. No obstante, existe la opción de compilar el trabajo mediante la terminal. Para ello, es necesario emplear el comando “qmake” junto con el archivo de extensión “.pro” para generar el programa.

Otro punto importante al utilizar Qt Creator es que la aplicación utiliza sus propias clases fundamentales dentro del marco de trabajo de Qt. Esto significa que Qt Creator tiene acceso a una lista de librerías y herramientas que facilitan la aplicación de ciertas funciones.

Las clases de Qt que se utilizaron son las siguientes:

- QHostAddress: Habilita el uso y manejo direcciones de internet.
- QUdpSocket: Habilita la creación y manipulación de sockets UDP en Qt.
- QObject: Permite funcionalidades fundamentales para la programación de objetos.
- QMutex: Permite sincronizar el acceso a datos compartidos para evitar condiciones de carrera.
- QThread: Facilita la creación y gestión de hilos en Qt.

El siguiente código corresponde al archivo encargado de los ajustes y de compilar todo el programa para el funcionamiento de la interfaz, lleva por nombre "Interfaz.pro".

```
#Configuración del proyecto que se creo en QT Creator
#Se configura el core para las funciones de qmake y para habilitar el uso de sockets
QT       += core gui
QT       += network
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

#Se configura la version de C++ que se va a utilizar
CONFIG += c++11

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs
    deprecated before Qt 6.0.0

#Se definen los archivos fuente y los headers respectivos, y el código que contiene
    el diseño de la interfaz.
SOURCES += \
    main.cpp \
    mainwindow.cpp \
    socket_udp.cpp

HEADERS += \
    mainwindow.h \
    socket_udp.h

FORMS += \
    mainwindow.ui

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target
```

Cuadro 22: Archivo de configuración de Qt Creator Interfaz.pro

A continuación, encontramos el código generado automáticamente por Qt Creator al diseñar la interfaz. Esto implica que cualquier modificación realizada en la aplicación se refleja automáticamente en este código. Es importante destacar que este código no puede

modificarse a menos que se realicen cambios fuera de la aplicación de Qt Creator, como en otro entorno de trabajo sin utilizar la aplicación.

```
/********************************************************************************
** Form generated from reading UI file 'mainwindowQgfVBP.ui'
**
** Created by: Qt User Interface Compiler version 5.15.3
**
** WARNING! All changes made in this file will be lost when recompiling UI file!
** Codigo generado al diseñar la interfaz
********************************************************************************/

#ifndef MAINWINDOWQGFVBP_H
#define MAINWINDOWQGFVBP_H

#include <QtCore/QVariant>
#include <QtWidgets/QApplication>
#include <QtWidgets/QMainWindow>
#include <QtWidgets/QMenuBar>
#include <QtWidgets/QPushButton>
#include <QtWidgets/QStatusBar>
#include <QtWidgets/QTextBrowser>
#include <QtWidgets/QTextEdit>
#include <QtWidgets/QWidget>

QT_BEGIN_NAMESPACE

class Ui_MainWindow
{
public:
    QWidget *centralwidget;
    QPushButton *Exit;
    QTextBrowser *textBrowser;
    QPushButton *RTU1L11;
    QPushButton *RTU1L10;
    QPushButton *RTU1L21;
    QPushButton *RTU1L20;
    QPushButton *RTU1LI1;
    QPushButton *RTU1LI0;
    QPushButton *pushButton_7;
    QPushButton *RTU2L11;
    QPushButton *RTU2L20;
    QPushButton *RTU2LI1;
    QPushButton *RTU2L21;
    QPushButton *RTU2L10;
    QTextEdit *textEdit;
    QMenuBar *menubar;
    QStatusBar *statusbar;

    void setupUi(QMainWindow *MainWindow)
    {
        if (MainWindow->objectName().isEmpty())
            MainWindow->setObjectName(QString::fromUtf8("MainWindow"));
        MainWindow->resize(496, 475);
        centralwidget = new QWidget(MainWindow);
        centralwidget->setObjectName(QString::fromUtf8("centralwidget"));
        Exit = new QPushButton(centralwidget);
        Exit->setObjectName(QString::fromUtf8("Exit"));
        Exit->setGeometry(QRect(400, 0, 89, 25));
        textBrowser = new QTextBrowser(centralwidget);
        textBrowser->setObjectName(QString::fromUtf8("textBrowser"));
        textBrowser->setGeometry(QRect(130, 0, 231, 31));
        RTU1L11 = new QPushButton(centralwidget);
        RTU1L11->setObjectName(QString::fromUtf8("RTU1L11"));
```

```
62          RTU1L11->setGeometry(QRect(20, 40, 101, 25));
63          RTU1L10 = new QPushButton(centralwidget);
64          RTU1L10->setObjectName(QString::fromUtf8("RTU1L10"));
65          RTU1L10->setGeometry(QRect(130, 40, 101, 25));
66          RTU1L21 = new QPushButton(centralwidget);
67          RTU1L21->setObjectName(QString::fromUtf8("RTU1L21"));
68          RTU1L21->setGeometry(QRect(20, 70, 101, 25));
69          RTU1L20 = new QPushButton(centralwidget);
70          RTU1L20->setObjectName(QString::fromUtf8("RTU1L20"));
71          RTU1L20->setGeometry(QRect(130, 70, 101, 25));
72          RTU1LI1 = new QPushButton(centralwidget);
73          RTU1LI1->setObjectName(QString::fromUtf8("RTU1LI1"));
74          RTU1LI1->setGeometry(QRect(20, 100, 101, 25));
75          RTU1LI0 = new QPushButton(centralwidget);
76          RTU1LI0->setObjectName(QString::fromUtf8("RTU1LI0"));
77          RTU1LI0->setGeometry(QRect(130, 100, 101, 25));
78          pushButton_7 = new QPushButton(centralwidget);
79          pushButton_7->setObjectName(QString::fromUtf8("pushButton_7"));
80          pushButton_7->setGeometry(QRect(370, 100, 101, 25));
81          RTU2L11 = new QPushButton(centralwidget);
82          RTU2L11->setObjectName(QString::fromUtf8("RTU2L11"));
83          RTU2L11->setGeometry(QRect(260, 40, 101, 25));
84          RTU2L20 = new QPushButton(centralwidget);
85          RTU2L20->setObjectName(QString::fromUtf8("RTU2L20"));
86          RTU2L20->setGeometry(QRect(370, 70, 101, 25));
87          RTU2LI1 = new QPushButton(centralwidget);
88          RTU2LI1->setObjectName(QString::fromUtf8("RTU2LI1"));
89          RTU2LI1->setGeometry(QRect(260, 100, 101, 25));
90          RTU2L21 = new QPushButton(centralwidget);
91          RTU2L21->setObjectName(QString::fromUtf8("RTU2L21"));
92          RTU2L21->setGeometry(QRect(260, 70, 101, 25));
93          RTU2L10 = new QPushButton(centralwidget);
94          RTU2L10->setObjectName(QString::fromUtf8("RTU2L10"));
95          RTU2L10->setGeometry(QRect(370, 40, 101, 25));
96          textEdit = new QTextEdit(centralwidget);
97          textEdit->setObjectName(QString::fromUtf8("textEdit"));
98          textEdit->setGeometry(QRect(20, 140, 461, 291));
99          MainWindow->setCentralWidget(centralwidget);
100         menubar = new QMenuBar(MainWindow);
101         menubar->setObjectName(QString::fromUtf8("menubar"));
102         menubar->setGeometry(QRect(0, 0, 496, 22));
103         MainWindow->setMenuBar(menubar);
104         statusbar = new QStatusBar(MainWindow);
105         statusbar->setObjectName(QString::fromUtf8("statusbar"));
106         MainWindow->setStatusBar(statusbar);
107
108         retranslateUi(MainWindow);
109
110         QMetaObject::connectSlotsByName(MainWindow);
111     } // setupUi
112
113     void retranslateUi(QMainWindow *MainWindow)
114     {
115         MainWindow->setWindowTitle(QCoreApplication::translate("MainWindow",
                "MainWindow", nullptr));
116         Exit->setText(QCoreApplication::translate("MainWindow", "Salir", nullptr));
117         textBrowser->setHtml(QCoreApplication::translate("MainWindow", "<!DOCTYPE
                HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\"
                \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
"p, li { white-space: pre-wrap; }\n"
"</style></head><body style=\" font-family:'Ubuntu'; font-size:11pt;
    font-weight:400; font-style:normal;\">\n"
"<p align=\"center\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
    margin-right:0px; -qt-block-indent:0; text-indent:0px;\">Menu
    Historiador</p></body></html>", nullptr));
122         RTU1L11->setText(QCoreApplication::translate("MainWindow", "RTU1 LED1 1",
                nullptr));
```

```
123         RTU1L10->setText(QCoreApplication::translate("MainWindow", "RTU1 LED1 0",
                nullptr));
124         RTU1L21->setText(QCoreApplication::translate("MainWindow", "RTU1 LED2 1",
                nullptr));
125         RTU1L20->setText(QCoreApplication::translate("MainWindow", "RTU1 LED2 0",
                nullptr));
126         RTU1LI1->setText(QCoreApplication::translate("MainWindow", "RTU1 LEDIoT 1",
                nullptr));
127         RTU1LI0->setText(QCoreApplication::translate("MainWindow", "RTU1 LEDIoT 0",
                nullptr));
128         pushButton_7->setText(QCoreApplication::translate("MainWindow", "RTU2 LEDIoT
                0", nullptr));
129         RTU2L11->setText(QCoreApplication::translate("MainWindow", "RTU2 LED1 1",
                nullptr));
130         RTU2L20->setText(QCoreApplication::translate("MainWindow", "RTU2 LED2 0",
                nullptr));
131         RTU2LI1->setText(QCoreApplication::translate("MainWindow", "RTU2 LEDIoT 1",
                nullptr));
132         RTU2L21->setText(QCoreApplication::translate("MainWindow", "RTU2 LED2 1",
                nullptr));
133         RTU2L10->setText(QCoreApplication::translate("MainWindow", "RTU2 LED1 0",
                nullptr));
134     } // retranslateUi
135
136 };
137
138 namespace Ui {
139     class MainWindow: public Ui_MainWindow {};
140 } // namespace Ui
141
142 QT_END_NAMESPACE
143
144 #endif // MAINWINDOWQGFVBP_H
```

Cuadro 23: Código del diseño de la interfaz mainwindow.ui

Para el correcto funcionamiento de la interfaz y de las clases generadas, se requiere un código principal que llame a la interfaz diseñada y a todas sus funcionalidades. A continuación, se presenta el código necesario:

```
1  /*
2   ============================================================================
3   Nombre: main.cpp
4   Autor: Rodrigo José García Ambrosy
5   Proyecto
6   ============================================================================
7   */
8  //Este código es el principal para el funcionamiento general del proyecto creado en
       Qt Creator
9
10 #include "mainwindow.h" //Incluye las funciones de la interfaza gráfica
11 #include "ui_mainwindow.h" //Incluye la interfaz gráfica
12 #include <QThread>  //Clase para utilizar hilos especifica de Qt
13 #include "socket_udp.h" //Incluye las funciones del socket
14
15 #define MSG_SIZE 3000 // Tamaño del mensaje
16 #define IP "192.168.1.255" // Dirección IP
17 #define OPCION_IP 0 /// 0 - hard coded
18 #define IP_LENGTH 15 // Tamaño de la dirección IP
19 int sockfd, n; // Descriptores de archivo
```

```
socket_udp udpSocket; //Instancia del socket
int port = 2044; //Puerto de comunicación
bool shouldExit = false; //Variable para salir del programa

void receiving(socket_udp *udpSocket) { // Función para recibir mensajes
    udpSocket->receiveAndWrite(MSG_SIZE); // Recibe el mensaje y se llama a la clase de recivir y escribir del socket
}

int main(int argc, char *argv[]) { //Función principal
    QApplication a(argc, argv); //Instancia de la interfaz gráfica
    QHostAddress address("192.168.1.255"); //Dirección IP
    // Inicializa el socket y lo configura
    udpSocket.initializeSocket(port);
    udpSocket.bindSocket(); //Se venicula el socket
    udpSocket.setBroadcastOption(); //Se configura el socket para utilizar broadcast
    udpSocket.setBroadcastIP(); //Se configura la dirección IP
    udpSocket.conexion("Conexion"); //Se envía el mensaje de conexión a las UTR

    QThread receiveThread; //Se crea un hilo para recibir mensajes
    udpSocket.moveToThread(&receiveThread); //Se mueve el socket al hilo creado
    QObject::connect(&receiveThread, &QThread::started, [&]() { //Se conecta el hilo con la función de recibir mensajes
        receiving(&udpSocket); //Se llama a la función de recibir mensajes
        }
    );

    receiveThread.start(); //Se inicia el hilo
    MainWindow w(&udpSocket); //Se crea la interfaz gráfica y se vincula con los sockets
    w.show();  //Se muestra la interfaz gráfica

    // Inicia el bucle de eventos
    int result = a.exec(); //Se ejecuta la interfaz gráfica

    return result; //Se retorna el resultado
}
```

Cuadro 24: Código principal main.cpp

Ahora disponemos de los archivos utilizados para crear las bibliotecas que contienen el funcionamiento interno de la interfaz y el uso de sockets del programa. Estas bibliotecas constan de un archivo "*Header*" y un archivo fuente. El archivo principal, que es el "*Header*", se encarga de establecer la clase y todas sus funciones, así como las comunicaciones necesarias entre archivos. Por otro lado, el archivo fuente define las acciones específicas de cada función de la clase diseñada en el "*Header*". A continuación, se presentan las bibliotecas creadas junto con sus respectivos archivos.

Estos son los códigos que realizan el funcionamiento interno de la interfaz:

```
/*
 Nombre: mainwindow.h
 Autor: Rodrigo José García Ambrosy
 Proyecto
*/
//Este código es el header que se encarga de establecer las funciones de la interfaz gráfica y crear la clase del mismo.

#ifndef MAINWINDOW_H
```

```
9  #define MAINWINDOW_H
10
11 #include <QMainWindow> //Clase para el uso de la interfaz gráfica
12 #include "socket_udp.h" //Librería para los sockets
13
14 QT_BEGIN_NAMESPACE //Se inicia el namespace
15 namespace Ui { class MainWindow; }
16 QT_END_NAMESPACE //Se termina el namespace
17
18 class MainWindow : public QMainWindow //Se crea la clase de la interfaz gráfica
19 {
20     Q_OBJECT
21
22 public: //Se crea la función pública
23     MainWindow(socket_udp *udpSocket, QWidget *parent = nullptr);
24     ~MainWindow();
25
26 private slots: //Se crean las funciones privadas
27     void updateTextEdit(const QString &data); //Función para actualizar el texto de
           la interfaz gráfica
28     void on_Exit_clicked(); //Función para salir del programa
29     //Funciones para encender y apagar los LED de las UTR desde la interfaz gráfica
30     void on_RTU1L11_clicked();
31     void on_RTU1L10_clicked();
32     void on_RTU1L21_clicked();
33     void on_RTU1L20_clicked();
34     void on_RTU1LI1_clicked();
35     void on_RTU1LI0_clicked();
36     void on_RTU2L11_clicked();
37     void on_RTU2L10_clicked();
38     void on_RTU2L21_clicked();
39     void on_RTU2L20_clicked();
40     void on_RTU2LI1_clicked();
41     void on_pushButton_7_clicked();
42
43 private: //Atributos privados de la clase
44     Ui::MainWindow *ui;
45     bool shouldExit;
46     socket_udp *udpSocketInstance;
47 };
48 #endif // MAINWINDOW_H
```

Cuadro 25: Header mainwindow.h

```
1  /*
2   Nombre: mainwindow.cpp
3   Autor: Rodrigo José García Ambrosy
4   Proyecto
5  */
6  //Código que contiene las funciones para el funcionamiento de la interfaz gráfica y
       contenido para la librería mainwindow.h
7  #include "mainwindow.h" //Se incluye el header con las funciones declaradas para
       crear la librería
8  #include "ui_mainwindow.h" //Se incluye la interfaz gráfica
9  #include "socket_udp.h" //Se incluye la librería del socket
10
11 MainWindow::MainWindow(socket_udp *udpSocket, QWidget *parent) //Se crea el
       constructor de la interfaz gráfica
12     : QMainWindow(parent),
13       ui(new Ui::MainWindow),
14       udpSocketInstance(udpSocket)
15
16 {
```

```
17      ui->setupUi(this);
18      connect(udpSocketInstance, &socket_udp::receivedMessageSignal, this,
            &MainWindow::updateTextEdit);
19      //Se conecta el socket con la función de actualizar el texto de la interfaz
            gráfica
20  }
21
22  MainWindow::~MainWindow() //Se crea el destructor de la interfaz gráfica
23  {
24      delete ui; //Se elimina la interfaz gráfica
25  }
26
27  void MainWindow::updateTextEdit(const QString &data) //Se crea la función para
        actualizar el texto de la interfaz gráfica
28  {
29      ui->textEdit->append(data); //Se actualiza el texto de la interfaz gráfica
30  }
31
32  void MainWindow::on_Exit_clicked() //Se crea la función para salir del programa
33  {
34      udpSocketInstance->setShouldExit(true); //Se cambia el valor de la variable para
            salir del programa
35      close(); //Se cierra el programa
36  }
37
38  //Funciones de los botones para encender y apagar los LED de las UTR desde la
        interfaz gráfica
39  void MainWindow::on_RTU1L11_clicked()
40  {
41      udpSocketInstance->sendTo("RTU1 LED1 1");
42  }
43
44  void MainWindow::on_RTU1L10_clicked()
45  {
46      udpSocketInstance->sendTo("RTU1 LED1 0");
47  }
48
49  void MainWindow::on_RTU1L21_clicked()
50  {
51      udpSocketInstance->sendTo("RTU1 LED2 1");
52  }
53
54  void MainWindow::on_RTU1L20_clicked()
55  {
56      udpSocketInstance->sendTo("RTU1 LED2 0");
57  }
58
59  void MainWindow::on_RTU1LI1_clicked()
60  {
61      udpSocketInstance->sendTo("RTU1 LEDIoT 1");
62  }
63
64  void MainWindow::on_RTU1LI0_clicked()
65  {
66      udpSocketInstance->sendTo("RTU1 LEDIoT 0");
67  }
68
69  void MainWindow::on_RTU2L11_clicked()
70  {
71      udpSocketInstance->sendTo("RTU2 LED1 1");
72  }
73
74  void MainWindow::on_RTU2L10_clicked()
75  {
76     udpSocketInstance->sendTo("RTU2 LED1 0");
77  }
78
79  void MainWindow::on_RTU2L21_clicked()
```

```
{
    udpSocketInstance->sendTo("RTU2 LED2 1");
}

void MainWindow::on_RTU2L20_clicked()
{
    udpSocketInstance->sendTo("RTU2 LED2 0");
}

void MainWindow::on_RTU2LI1_clicked()
{
    udpSocketInstance->sendTo("RTU2 LEDIoT 1");
}

void MainWindow::on_pushButton_7_clicked()
{
    udpSocketInstance->sendTo("RTU2 LEDIoT 0");
}
```

Cuadro 26: Código fuente mainwindow.cpp

Estos son los códigos que se encargan de la comunicación por sockets:

```
/*
 Nombre: socket_udp.h
 Autor: Rodrigo José García Ambrosy
 Proyecto
*/

//Código para la librería del socket UDP que contiene la classe para utilizar sockets.
#ifndef SOCKET_UDP_H
#define SOCKET_UDP_H

#include <sys/socket.h> //Librería para utilizar sockets
#include <netinet/in.h> //Librería para utilizar direcciones de internet
#include <QHostAddress> //Clase propia de Qt para utilizar direcciones de internet
#include <QUdpSocket>   //Clase propia de Qt para utilizar sockets UDP
#include <QObject>    //Clase propia de Qt para utilizar objetos
#include <QMutex>   //Clase propia de Qt para utilizar mutex
#include <iostream> //Librería para utilizar entrada y salida de datos
#include <cstring> //Librería para utilizar cadenas de caracteres
#include <unistd.h> //Librería para utilizar funciones de sistema
#include <arpa/inet.h> //Librería para manipular direcciones de internet

class socket_udp : public QObject{ //Classe para utilizar sockets UDP
    Q_OBJECT
public:
    socket_udp(); //Constructor
    ~socket_udp(); //Destructor

    bool initializeSocket(int port); //Función para inicializar el socket
    bool bindSocket(); //Función para enlazar el socket
    bool setBroadcastOption(); //Función para establecer la opción de broadcast
    bool setBroadcastIP(); //Función para establecer la dirección de broadcast
    void sendTo(const char *message); //Función para enviar un mensaje
    void receiveAndWrite(size_t bufferSize); //Función para recibir y escribir un mensaje
    void setShouldExit(bool value); //Función para establecer el valor de shouldExit y salir del programa
    void conexion(const char *message); //Función para enviar mensaje que establece la conexión
```

```
36
37 signals: //Señales para enviar recibir mensajes
38     void receivedMessageSignal(const QString& message);
39
40 private slots: //Slots privados para procesar mensajes.
41     void processPendingDatagrams();
42
43 private: //Atributos privados de la clase
44     QMutex mutex;
45     bool shouldExit;
46     QUdpSocket *udpSocket;
47     int sockfd;
48     struct sockaddr_in socketAddress;
49
50
51 };
52
53
54
55 #endif // SOCKET_UDP_H
```

Cuadro 27: Header socket_udp.h

```
1  /*
2   Nombre: socket_udp.cpp
3   Autor: Rodrigo José García Ambrosy
4   Proyecto
5  */
6  //Este código contiene las funciones necesarias para el archivo header que contiene
       la clase.
7  #include "socket_udp.h" //Incluye las funciones del socket
8  #include "mainwindow.h" //Librería de las funciones de la interfaz gráfica
9
10 #define IP "192.168.1.255" // Dirección IP
11
12
13 socket_udp::socket_udp() : QObject(), shouldExit(false), udpSocket(new
       QUdpSocket(this)) { //Constructor de la clase
14     shouldExit = false; //Variable para salir del programa
15     connect(udpSocket, &QUdpSocket::readyRead, this,
           &socket_udp::processPendingDatagrams);
16     //Se conecta el socket con la función de procesar mensajes
17 }
18
19 socket_udp::~socket_udp() { //Destructor de la clase
20     if (sockfd != -1) {
21         close(sockfd);
22     }
23 }
24
25 bool socket_udp::initializeSocket(int port) { //Función para inicializar el socket
26     sockfd = socket(AF_INET, SOCK_DGRAM, 0); //Se crea el socket, sin conexión
27     if (sockfd < 0) {   //Verifica si hubo error
28         std::cerr << "ERROR opening socket" << std::endl;
29         return false;
30     }
31     //Se configura el socket y su estructura
32     memset(&socketAddress, 0, sizeof(socketAddress));
33     socketAddress.sin_family = AF_INET;
34     socketAddress.sin_port = htons(port); //Se configura el puerto
35     return true;
36 }
```

```
37
38 bool socket_udp::bindSocket() { //Función para vincular el socket
39     if (bind(sockfd, reinterpret_cast<struct sockaddr *>(&socketAddress),
           sizeof(socketAddress)) < 0) { //Verifica si hubo error
40         std::cerr << "ERROR binding socket" << std::endl;
41         return false;
42     }
43
44     return true;
45 }
46
47 bool socket_udp::setBroadcastOption() { //Función para configurar el socket para
       utilizar broadcast
48     int boolval = 1;
49     if (setsockopt(sockfd, SOL_SOCKET, SO_BROADCAST, &boolval, sizeof(boolval)) < 0)
           { //Verifica si hubo error
50         std::cerr << "ERROR setting socket options" << std::endl;
51         return false;
52     }
53
54     return true;
55 }
56
57 bool socket_udp::setBroadcastIP() { //Función para configurar la dirección IP
58     socketAddress.sin_addr.s_addr = inet_addr(IP); //Se configura la dirección IP
59     return true;
60 }
61
62 void socket_udp::conexion(const char *message){ //Función para enviar el mensaje de
       conexión
63     char CONEXION[128]; // Variable para almacenar el mensaje
64     int n; // Variable para almacenar el número de bytes enviados
65     std::strcpy(CONEXION, message); // Copia el mensaje en la variable
66     n = sendto(sockfd, CONEXION, strlen(CONEXION), 0, (const struct sockaddr
           *)&socketAddress, sizeof(socketAddress)); // Envia el mensaje
67     if (n < 0){
68         std::cerr << "ERROR al enviar" << std::endl; // Verifica si hubo error
69     }
70     std::cout << "Esto se envió: " << message << std::endl; // Imprime el mensaje
           enviado
71 }
72
73 void socket_udp::sendTo(const char *message) { //Función para enviar mensajes
74     int n; // Variable para almacenar el número de bytes enviados
75     n = sendto(sockfd, message, strlen(message), 0, (const struct sockaddr
           *)&socketAddress, sizeof(socketAddress)); // Envia el mensaje
76     if (n < 0) {
77         perror("Error sending message"); // Verifica si hubo error
78     } else {
79         std::cout << "Esto se envió: " << message << std::endl; // Imprime el
               mensaje enviado
80     }
81 }
82
83 void socket_udp::receiveAndWrite(size_t bufferSize) { //Función para recibir y
       escribir mensajes
84     char buffer[bufferSize]; // Variable para almacenar el mensaje
85     while (true) { // Ciclo infinito
86         {
87             QMutexLocker locker(&mutex);  //Se bloquea el mutex
88             if (shouldExit) { //Verifica si debe salir del ciclo
89                 break;  //Sale del ciclo
90             }
91         }
92     ssize_t bytesRead = recvfrom(sockfd, buffer, bufferSize, 0, nullptr, nullptr);
           //Recibe el mensaje y lee el número de bytes recibidos
93     std::cout << "Esto se recibió: " << buffer << std::endl; // Imprime el mensaje
           recibido
```

```
94      if (bytesRead == -1){ //Verifica si hubo error
95          std::cerr << "Error reciviendo data" << std::endl;
96      } else {
97          QString receivedMessage = QString::fromUtf8(buffer, bytesRead); //Convierte
                el mensaje a QString
98          emit receivedMessageSignal(receivedMessage); //Emite la señal de mensaje
                recibido
99      }
100     }
101 }
102
103 void socket_udp::processPendingDatagrams() { //Función para procesar mensajes
104     while (udpSocket->hasPendingDatagrams()) { //Verifica si hay mensajes pendientes
105         QByteArray datagram; // Variable para almacenar el mensaje
106         datagram.resize(udpSocket->pendingDatagramSize()); //Se redimensiona la
                variable
107         udpSocket->readDatagram(datagram.data(), datagram.size()); //Se lee el
                mensaje
108
109         QString receivedMessage = QString::fromUtf8(datagram); //Convierte el
                mensaje a QString
110         emit receivedMessageSignal(receivedMessage); //Emite la señal de mensaje
                recibido
111     }
112 }
113
114 void socket_udp::setShouldExit(bool value) { //Función para salir del programa
115     shouldExit = value; //Se asigna el valor
116 }
```

Cuadro 28: Código fuente socket_udp.cpp

Estos códigos que se presentaron son indispensables para el correcto funcionamiento de la interfaz diseñada. Como se mencionaba, la única diferencia entre esta interfaz y el código utilizado para Historiador.cpp es que la interfaz no tiene la capacidad de generar un archivo .txt. Además, es necesario indicar el puerto que se va a utilizar dentro del código en lugar de hacerlo al momento de ejecutar el programa, como ocurre en Historiador.cpp.

CAPÍTULO 9

Rendimiento de programas en distintos sistemas operativos de Linux

Este capítulo tiene como objetivo comparar el rendimiento de los programas de C con los de C++ para evaluar qué diferencias existen con respecto al uso de la memoria y el porcentaje de uso del CPU. Esto se realizó en distintos sistemas operativos (RockyLinux, Ubuntu, Debian y Oracle).

Para realizar estas pruebas de rendimiento, se utilizó el código L3_Hilos_Ej2.cpp modificado y se creó su versión respectiva en C. El código de C++ es el siguiente:

```
1  /*
2   ============================================================================
3   Nombre: L3HilosEj2.cpp
4   Autor:  Rodrigo José García Ambrosy
5   Este código se utilizo para obtener el uso de CPU y Memoria del programa
6   ============================================================================
7   */
8
9  #include <iostream>
10 #include <fstream>
11 #include <thread>
12 #include <chrono>
13 #include <unistd.h>
14 #include <iomanip>
15 #include <sys/resource.h>
16
17 void My_Thread(void *ptr)
18 {
19     char *message;
20     message = static_cast<char *>(ptr);
21
22     while (true)
23     {
24         std::cout << message;
25         std::cout.flush();
26         std::this_thread::sleep_for(std::chrono::microseconds(1100000));
27
28         // Obtener el PID del programa actual
```

```
        pid_t pid = getpid();

        // Obtener el consumo de CPU y Memoria del programa
        struct rusage usage;
        getrusage(RUSAGE_SELF, &usage);

        // Información de CPU
        double cpu_usage = (usage.ru_utime.tv_sec + usage.ru_utime.tv_usec /
            1000000.0) * 100.0;

        // Información de memoria (en kilobytes)
        long mem_usage = usage.ru_maxrss;

        std::cout << "Uso de CPU del programa: " << std::fixed <<
            std::setprecision(3) << cpu_usage << " %\n";
        std::cout << "Uso de Memoria del programa: " << mem_usage << " KB\n";
    }
}

int main()
{
    std::thread thread2;
    const char *message1 = "Hello ";
    const char *message2 = "World\n";

    thread2 = std::thread(My_Thread, const_cast<char *>(message1));

    while (true)
    {
        std::cout << message2;
        std::cout.flush();
        std::this_thread::sleep_for(std::chrono::microseconds(1000000));
    }

    return 0;
}
```

Cuadro 29: Código L3_Hilos_Ej2.cpp

El código en C es el siguiente:

```
/*
 ============================================================================
 Nombre: L2HilosEj2.cpp
 Autor:  Rodrigo José García Ambrosy
 Este código se utilizo para obtener el uso de CPU y Memoria del programa
 ============================================================================
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/resource.h>

// Código a ejecutar por el segundo hilo
void My_Thread(void *ptr)
{
    char *message; // puntero, para la cadena de caracteres (string)
    message = (char *)ptr; // recupera la información pasada a este hilo

    while (1)
```

```c
    {
        printf("%s", message);
        fflush(stdout);
        usleep(1100000);

        // Obtener el consumo de CPU del programa
        struct rusage usage;
        getrusage(RUSAGE_SELF, &usage);

        // Información de CPU
        double cpu_usage = (usage.ru_utime.tv_sec + usage.ru_utime.tv_usec /
            1000000.0) * 100.0;
        printf("Uso de CPU del programa: %.3f %%\n", cpu_usage);

        // Información de memoria (en kilobytes)
        printf("Uso de Memoria del programa: %ld KB\n", usage.ru_maxrss);
    }

    // Lo siguiente no se ejecutará, pero es buena costumbre incluirlo
    pthread_exit(0); // Para salir correctamente del hilo
}

// Función principal (primer hilo de ejecución)
int main(void)
{
    pthread_t thread2; // variable para identificar el 2do hilo que se creará
    // Los siguientes son dos strings
    char *message1 = "Hello ";
    char *message2 = "World\n";

    // La siguiente función crea un POSIX thread (pthread), que es un hilo del
    // estándar POSIX. Los argumentos de entrada son: la variable tipo pthread,
    // configuraciones (NULL para usar las default), la función a ejecutar por
    // el hilo, y un puntero para la información que se quiere pasar al nuevo
    // hilo (NULL si no se quiere pasar nada).
    pthread_create(&thread2, NULL, (void *)&My_Thread, (void *)message1);

    while (1)
    {
        printf("%s", message2);
        fflush(stdout);
        usleep(1000000);
    }

    return (0);
}
```

Cuadro 30: Código L2_Hilos_Ej2.c

Al utilizar estos programas luego de que se corre el segundo hilo este imprime el porcentaje del CPU que utiliza el programa y la cantidad de memoria que se utiliza. Para compara de manera sencilla las diferencias de consumo se realizó una tabla comparativa para observar cómo se comportan los dos lenguajes de programación en los sistemas operativos de Linux mencionados anteriormente.

A continuación se muestran las representaciones gráficas de las pruebas realizadas en general, y luego se presentan las específicas comparando los lenguajes de C y C++ en cada sistema operativo utilizado.

Sistema operativo Linux	% de CPU C	% de CPU C++	Memoria (KB) C	Memoria (KB) C++
RockyLinux	0.068	0.151	3000	4072
Ubuntu	0.073	0.125	1996	3200
Debian	0.069	0.122	2352	2352
Oracle	0.061	0.120	2804	2804

Cuadro 31: Comparación de resultados

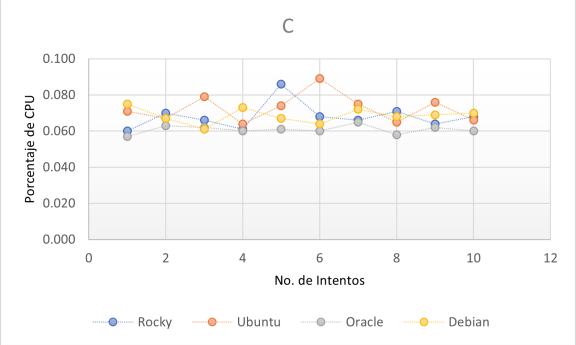


Figura 11: Uso de C en distintos sistemas operativos.

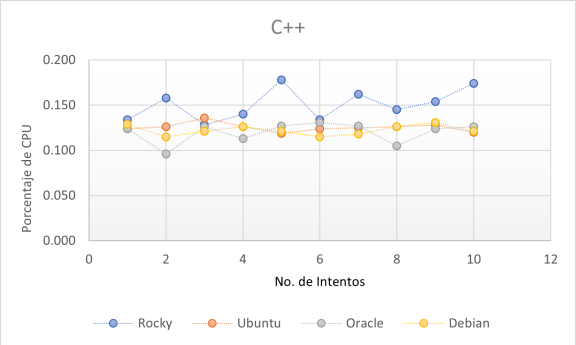


Figura 12: Uso de C++ en distintos sistemas operativos.

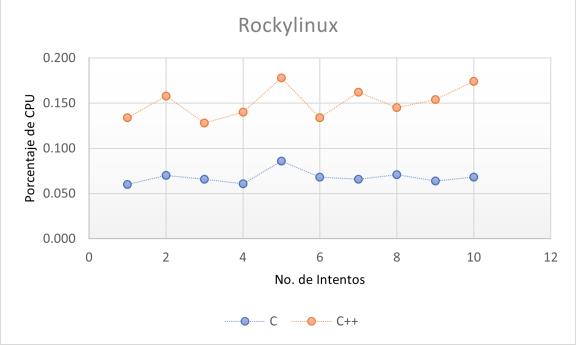


Figura 13: Uso del CPU en RockyLinux entre C y C++

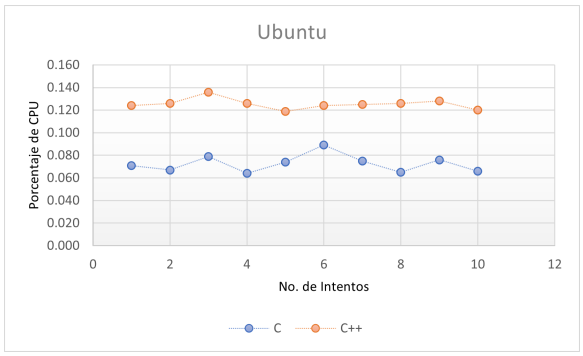


Figura 14: Uso del CPU en Ubuntu entre C y C++

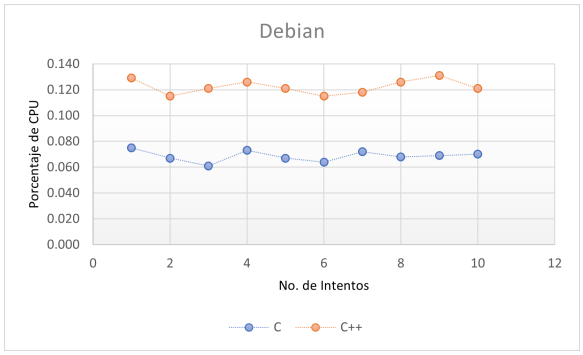


Figura 15: Uso del CPU en Debian entre C y C++

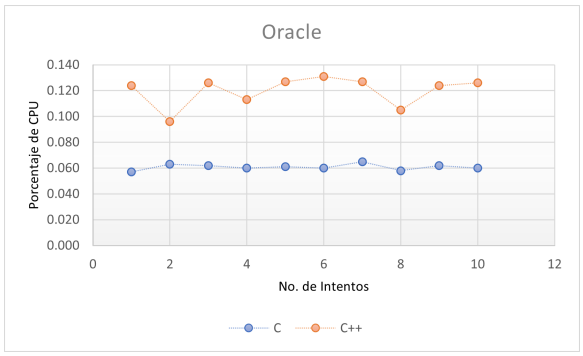


Figura 16: Uso del CPU en Oracle entre C y C++

Después de realizar pruebas en varios sistemas operativos con ambos programas, es evidente que el lenguaje de programación C++ utiliza casi el doble de recursos de la CPU en comparación con los programas escritos en C. Sin embargo, en lo que respecta al uso de memoria en los sistemas operativos Debian y Oracle, ambos programas consumen una cantidad equivalente. En resumen, estos resultados indican claramente que el desarrollo en C++ requiere un poco más de recursos en comparación con C.

Para proporcionar una referencia visual sobre cómo se obtuvieron estos resultados, se adjuntó en los anexos del capítulo 13.2 un archivo PDF con capturas de pantalla de las terminales de cada uno de los sistemas operativos Linux.

CAPÍTULO 10

Conclusiones

- Las librerías estándar disponibles en el lenguaje C++ se pueden utilizar para las funciones como multihilos y multiprocesos, como se observó al realizar el laboratorio núm.3.
- Las librerías estándar disponibles en el lenguaje C++ y las librerías que existen en los sistemas operativos de Linux permiten utilizar las funciones necesarias para la implementación de algoritmos de escalonamiento, sincronización y comunicación de interprocesos. Esto se respaldó con los laboratorios 6 y 7.
- La utilización de la librería WiringPi en C++ para la gestión de puertos de entradas/salidas en la Raspberry Pi es factible y está respaldada por la documentación oficial de la librería. Está se confirmó en los laboratorios 5 y 7, reforzando la eficacia y versatilidad de esta herramienta en proyectos de desarrollo en este entorno.
- La implementación de clases en C++ demuestra ser más eficiente y proporciona un enfoque práctico y organizado para encapsular funciones y datos relacionados. Esto fomenta la modularidad y la reutilización de código, superando el alcance ofrecido por el lenguaje de programación C.
- La configuración de comunicación a través de protocolos de red en C++ resulta notablemente más sencilla de manejar que en el lenguaje C. Esto se evidenció tanto en el laboratorio 8 como en el proyecto, pues se demostró la ventaja de utilizar C++ en la implementación de soluciones de este tipo.
- Según las muestras tomadas experimentalmente se evidencia que el C++ necesita de más recursos del sistema, como CPU y memoria RAM, en comparación con el lenguaje C.

CAPÍTULO 11

Recomendaciones

- Se recomienda considerar la posibilidad de utilizar las librerías de C en ciertos programas de C++. En ocasiones, resulta más sencillo emplearlas, especialmente teniendo en cuenta que C++ tiene la capacidad de utilizarlas.
- Se recomienda familiarizarse con la sintaxis y las funciones de las librerías específicas de Linux al utilizarlas en un código en C++. Esto evitará posibles confusiones con la sintaxis propia de C++, facilitando así la integración de las herramientas proporcionadas por el sistema operativo.
- Se recomienda, para futuros trabajos con un sistema SCADA, mejorar las capacidades de la interfaz gráfica para facilitar el almacenamiento de datos; además de mejorar el sistema de tal manera que se pueda comunicar entre diferentes redes en lugar de solo utilizar redes LAN.
- Se recomienda verificar la versión de C++ que se esté utilizando, ya que algunas librerías solo son compatibles con las versiones más recientes.

CAPÍTULO 12

Referencias

[1] J. E. Archila, "Diseño e implementación de capacidades automáticas de navegación para un Robot Explorador Modular," Tesis de mtría., Universidad del valle de Guatemala, 2022.

[2] J. Matikainen, "Controlling RS-232 equipped devices using Raspberry Pi and C++," Tesis de mtría., Metropolia University of Applied Sciences, 2018.

[3] H. Schildt, *C, the complete reference*, 4th ed. Berkeley: Osborne/McGraw-Hill, 2000, OCLC: 47008743, ISBN: 978-0-07-213295-3.

[4] B. W. Kernighan y D. M. Ritchie, *The C Programming Language*, 2nd ed. Pearson, 1988, ISBN: 0-13-110362-8.

[5] W3schools. "C Syntax." (), dirección: `https://www.w3schools.com/c/c_syntax.php`. (Accedido el 27/5/2023).

[6] W3schools. "C++ Syntax." (), dirección: `https://www.w3schools.com/cpp/cpp_syntax.asp`. (Accedido el 27/5/2023).

[7] J. García de Jalón, J. I. Rodríguez, J. M. Sarriegui y A. Brazález, *Aprenda C++ como si estuviera en primero*, 1998.

[8] G. Henderson. "Wiring Pi GPIO Interface library for the Raspberry Pi." (), dirección: `http://wiringpi.com`. (Accedido el 21/5/2023).

[9] G. Henderson. "Wiring Pi GPIO Interface library for the Raspberry Pi." (), dirección: `http://wiringpi.com/reference/`. (Accedido el 21/5/2023).

[10] R. Garg y G. Verma, *Operating systems: an introduction*. Dulles, Virginia ; Boston, Massachusette ; New Delhi: Mercury Learning e Information, 2017, 290 págs., OCLC: ocn908375738, ISBN: 978-1-942270-38-6.

[11] R. Arpaci-Dusseau y A. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, 1st ed. CreateSpace Independent Publishing Platform, 2018.

[12] J. K. Peckol, *Embedded Systems A Contemporary Design Tool*, 2nd ed. Wiley, 2019, ISBN: 978-0-471-72180-2.

[13] C. L. Liu y J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *J. ACM*, vol. 20, n.º 1, págs. 46-61, ene. de 1973, ISSN: 0004-5411. DOI: 10.1145/321738.321743. dirección: https://doi.org/10.1145/321738.321743.

[14] Baeldung. "Process Scheduling." (), dirección: https://www.baeldung.com/cs/process-scheduling. (Accedido el 28/5/2023).

[15] R. J. Anthony, *Systems Programming Designing and Developing Distributed Applications*. Morgan Kaufmann, 2016, ISBN: 978-0-12-800729-7.

[16] L. L. Peterson y B. S. Davie, *Computer Networks A Systems Approach*, 6th ed. Morgan Kaufmann, 2022, ISBN: 9780128182000.

[17] W.-T. Chan, T.-W. Lam, K.-S. Liu y P. W. Wong, "New resource augmentation analysis of the total stretch of SRPT and SJF in multiprocessor scheduling," *Theoretical Computer Science*, vol. 359, n.º 1, 2006, ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2006.06.003. dirección: https://www.sciencedirect.com/science/article/pii/S0304397506003525.

[18] J. W. Kurose y K. W. Ross, *Computer Networking A Top-Down Approach*, 8th ed. Pearson, 2022, ISBN: 93-5606-131-9.

[19] Imperva. "Transmission control protocol (TCP)." (), dirección: https://www.imperva.com/learn/ddos/tcp-transmission-control-protocol/. (Accedido el 27/5/2023).

[20] Imperva. "User datagram protocol (UDP)." (), dirección: https://www.imperva.com/learn/ddos/udp-user-datagram-protocol/. (Accedido el 27/5/2023).

[21] A. Silberschatz, P. B. Galvin y G. Gagne, *Operating System Concepts*, 10th ed. LSC Kendallville: Laurie Rosatone, 2018.

[22] R. Hat. "What is the Linux kernel?" (), dirección: https://www.redhat.com/en/topics/linux/what-is-the-linux-kernel. (Accedido el 28/5/2023).

[23] R. Pi. "Raspberry Pi OS." (), dirección: https://www.raspberrypi.com/documentation/computers/os.html#introduction. (Accedido el 28/5/2023).

[24] R. Pi. "Unofficial list of Raspberry Pi distributions." (), dirección: https://forums.raspberrypi.com/viewtopic.php?t=328911. (Accedido el 28/5/2023).

[25] phoenixNAP GLOBAL IT SERVICES. "Linux Commands All Users Should Know Ultimate List." (), dirección: https://phoenixnap.com/kb/linux-commands. (Accedido el 28/5/2023).

[26] TheQtCompany. "Qt Group." (), dirección: https://www.qt.io. (Accedido el 10/11/2023).

[27] TheQtCompany. "Qt Documentation." (), dirección: https://doc.qt.io. (Accedido el 10/11/2023).

CAPÍTULO 13

Anexos

13.1. Guías de laboratorio

En esta sección de anexos, se da continuidad a lo realizado en los trabajos de los laboratorios 3, 5, 6, 7 y 8, mediante las guías correspondientes. Se explica detalladamente el proceso que debe seguirse para replicarlos, incluyendo las librerías a utilizar, las funciones que se deben emplear, los programas y los resultados que se pueden esperar al replicar el trabajo realizado. Se mencionan explícitamente los objetivos de cada laboratorio, se ofrecen pequeñas recomendaciones y, además, en las guías se proporcionan los códigos necesarios para llevarlo a cabo.

13.1.1. Laboratorio núm. 3

Laboratorio 3 (y lección 2):
Procesos, hilos posix, condición de carrera

Objetivos

- Crear y ejecutar procesos con uno o más hilos de ejecución.
- Observar similitudes y diferencias entre procesos e hilos.
- Conocer los **thread**, la función **fork()** y el concepto de Condición de Carrera.

Duración: 1 sesión de teoría y 1 sesión de laboratorio

Al trabajar en Linux, pueden crear proyectos de Eclipse para cada programa a continuación, o pueden compilar los programas directamente desde la consola. Si trabajan en Windows, usarán Cygwin.

Primera parte: múltiples procesos, múltiples hilos

1. Compilen los programas **L3_Hello.cpp** y **L3_World.cpp**. Asegúrense de nombrar los ejecutables **L3_Hello** y **L3_World**, respectivamente. En general, el nombre del ejecutable no tiene que ser igual al nombre del archivo fuente. Sin embargo, esta vez nos servirá tener esos nombres (inciso 3 y otros más).
2. Ejecutarán los dos programas en una misma consola. Esencialmente, deberán mandar el primer proceso al "trasfondo". Para ello, usarán el operador & (como al invocar Eclipse desde la terminal). Ejecuten el primer programa así:
   ./L3_Hello &
   Eso permitirá seguir usando la consola, por lo que podrán ejecutar el segundo programa:
   ./L3_World
   Noten que el primer programa imprime los mensajes a la terminal al mismo tiempo que ustedes ingresan el comando para el segundo programa. Tengan cuidado de ingresar correctamente.

3. Abran una segunda consola (o ventana de Cygwin). Mientras los programas anteriores estén ejecutándose, ingresen el comando: **ps ax**
   A continuación, ingresen el comando: **ps ax | grep L3_**
   Tomen una captura de pantalla de la consola para cada caso. Describan las diferencias que observen. ¿Qué hace el operador | (*pipe*)? ¿Qué hace el comando grep?

4. Encuentren el identificador de cada uno de los procesos (PID) correspondientes a sus programas. Para cada proceso, utilicen el comando: **kill PID**, donde PID es un número (el identificador correspondiente). ¿Qué ocurre?

5. A continuación, estudien el programa **L3_Hilos_Ej1.cpp**.
   Atiendan las explicaciones del catedrático sobre este programa y el concepto de hilos.

6. Intenten compilar el programa en la consola directamente, siguiendo la sintaxis descrita en prácticas anteriores. En Linux, deberían observar un problema al compilar (en Cygwin no se verá el problema) ¿Cuál es el problema?
   Este programa utiliza la librería **thread**, la cual permite crear POSIX *threads*. Al momento de compilar el programa, deben indicar explícitamente el uso de la librería al invocar el compilador. Para hacer eso, agreguen la opción **-lpthread** al llamar **g++**.
   Una vez compilado, corran el programa en una de las consolas.

7. Ahora estudien, compilen y ejecuten el programa **L3_Hilos_Ej2.cpp** en una de las consolas.
   ¿Cómo se compara lo observado aquí con lo observado al ejecutar los programas de los incisos 1 y 2?

8. En la otra consola, ingresen el comando **ps ax | grep L3_ .** ¿Qué observan ahora? Tomen una captura de pantalla de la consola.

9. En Eclipse, también se debe indicar el uso de librerías como pthread. En Linux, creen un nuevo proyecto y copien el código de **L3_Hilos_Ej2.cpp.** Intenten compilar el programa. ¿Algún problema?
   Corran el programa creado en Eclipse en una de las consolas que usaron antes (Linux)

Segunda parte: función fork(), contexto entre procesos y entre hilos

Como tarea, ustedes investigaron la función **fork()** y el concepto de Condición de Carrera. En esta parte del laboratorio, ustedes correrán algunos programas que usan/ejemplifican la función y el concepto.

1. Estudien el código **L3_fork_Ej1.cpp**. Brevemente mencionen las similitudes que encuentren con los programas de la Primera Parte.
   El instructor explicará el funcionamiento de la función fork().

2. En una terminal, compilen y ejecuten el programa (nómbrenlo **L3_fork_Ej1**). Sólo deben invocar el programa una vez:
   ./L3_fork_Ej1
   ¿Cómo se compara la salida de este programa con la de los programas de la Primera Parte?

3. En la otra terminal, ingresen el comando **ps ax | grep L3_** . ¿Qué observan ahora? ¿Cuántos procesos L3_fork_Ej1 distintos observan? Tomen una captura de pantalla.

A continuación, compararán los programas **L3_fork_contexto** y **L3_pthread_contexto**.

4. Estudien ambos códigos. Anoten las similitudes y diferencias entre ambos.

5. Compilen los programas. Nuevamente, por conveniencia, nombren los ejecutables **L3_fork_contexto** y **L3_pthread_contexto**, respectivamente.
6. Ejecuten ambos programas, en consolas distintas. ¿Observan alguna diferencia en lo desplegado por ambos programas? Expliquen las diferencias, si las hay.

7. En una tercera consola, ingresen **ps ax | grep L3_** ¿Qué observan ahora? Tomen una captura de pantalla.

8. Según lo investigado en la tarea 1 y lo observado en la práctica, ¿Qué entienden por Condición de Carrera? ¿Qué problemas puede haber cuando múltiples procesos/hilos pueden acceder/modificar un recurso compartido?

Tercera Parte: Más Ejemplos y Ejercicios

1. Estudien, compilen y ejecuten el programa **L3_varios_forks**. Antes que termine la ejecución, corran **ps ax | grep L3_** en otra consola. ¿Hace sentido lo que observan? Expliquen claramente. Tomen una captura de pantalla.

2. Modifiquen el programa **L3_Hilos_Ej2.cpp** para que tenga cuatro hilos en lugar de dos. Para ello, deberán definir más variables tipo **std::thread**, y llamar a la función ***nombre de variable* = std::thread();** más veces en el main. Recuerden que el main es el primer hilo. El segundo hilo debe quedar igual que en **L3_Hilos_Ej2.cpp**.
    a. El tercer hilo debe ejecutar el mismo código **My_Thread**, pero debe desplegar un mensaje distinto. Es decir, al llamar a la función **std::thread()**, el tercer argumento debe ser el mismo que para el segundo hilo. Pero el cuarto argumento debe ser un string distinto.
    b. El cuarto hilo debe ejecutar un código distinto a los hilos 2 y 3. Para ello, deben definir una función adicional. La estructura de esta función será similar a **My_Thread**. Pueden nombrar a la función como ustedes deseen (siempre y cuando no sea una palabra reservada del lenguaje C++). Al llamar a **std::thread()** en el main, el tercer argumento debe tener el nombre de su función nueva.
    Este cuarto hilo no recibirá ningún argumento. Por lo tanto, el último argumento de **std::thread()** en el main debe ser **NULL**.
    Lo único que debe hacer este hilo es "dormir" por 5 segundos, luego desplegar un mensaje cualquiera (por ejemplo, "ELECTRONICA DIGITAL 3, LAB 3, su nombre"), y luego salir (**std::thread::detach()**).

Material de Apoyo

1. Páginas del manual (*man page*) de la función **fork**.

13.1.2. Laboratorio núm. 5

Laboratorio 5: conexión PC – RPi; Puertos de E/S de uso general de la RPi (programas en C++)

Objetivos

- Aprender a conectarse a la RPi desde una PC usando Cygwin/PowerShell/cmd (terminal) y VNC Viewer (interfaz gráfica), y aprender a transferir archivos entre ambos dispositivos.
- Aprender a usar los puertos de la RPi desde programas en el Espacio de Usuario, escritos en lenguaje C++.

Tercera parte: manejo de puertos GPIO con programas escritos en C++

A continuación, deberá crear programas en C++ que manipulen los puertos GPIO. Tendrá la libertad de crearlos/editarlos usando KWrite y compilarlos directamente desde una terminal, o crear proyectos en Eclipse y editar y compilar los programas desde dicho entorno.

1. Encendido y Apagado de LEDs: Escriba un programa en C++ que alternativamente encienda y apague dos LEDs, cada cierto tiempo aleatorio entre 0.5 s y 1.5 s. Cuando un LED esté encendido, el otro deberá estar apagado, y vice-versa. El tiempo debe ser distinto cada vez.

Ayuda: investiguen la funciones rand y srand.

Considere las siguientes funciones de wiringPi:

- int **wiringPiSetup**(void)
- int **wiringPiSetupGpio**(void)
- void **pinMode**(int Pin, int Value)
- void **digitalWrite**(int Pin, int Value)

Nota: Necesitará incluir el encabezado **wiringPi.h** en su código, y necesitará incluir la librería wiringPi al compilar/enlazar su programa (-lwiringPi).

Para el siguiente inciso, el instructor discutirá con ustedes conexiones para usar una bocinita. Necesitará resistencias y un transistor. Recuerde usar el pin de 3.3 V de la RPi para alimentar su circuito.

2. Producir un sonido con la bocina: Escriba un programa en C++ que genere una señal cuadrada para activar una bocina. La señal no debe empezar de inmediato al correr el programa. Deberá esperar hasta que un push button sea presionado. La señal será generada cambiando constantemente el valor del puerto GPIO conectado a la bocina de alto a bajo (en un bucle). Deberá probar distintas frequencias hasta lograr escuchar algo. Trate de obtener un sonido "bonito". Además de estar en un bucle generando la señal cuadrada, el programa debe ser capaz de aceptar las siguientes opciones ingresadas por medio del teclado: 'p', para pausar el sonido, 'r', para reanudar el sonido, y 's' para salir del programa (terminar la ejecución).

Además de las funciones de wiringPi mencionadas en el inciso 1 de la Tercera Parte, considere las funciones siguientes:

- void pullUpDnControl(int Pin, int pud_mode)
- int digitalRead(int Pin)

Recuerde mantener un respaldo de sus programas. Como se mencionó antes, es conveniente mantener un folder con todas sus prácticas en Google Drive, Dropbox, Box, GitHub, etc. Puede abrir su cuenta de Drive (Dropbox, Box, etc.) desde el navegador de Internet de la RPi. Adicionalmente, puede transferir sus archivos de la RPi a su PC, como aprendió en la Primera Parte de esta práctica. Lo importante es tener un back-up de sus archivos, en caso de que le pase algo a la tarjeta SD.

13.1.3. Laboratorio núm. 6

Laboratorio 6: Temporizadores, tareas periódicas y sincronización simple

Objetivos

- Aprender a configurar temporizadores (timers) en espacio de usuario.
- Aprender a configurar hilos como tareas periódicas y a establecer sus prioridades.
- Observar ventajas y desventajas de sincronizar tareas usando únicamente períodos y tiempos de inicio.

Antes de comenzar los experimentos descritos abajo, presten atención a la "IE3059 - Laboratorio 6, presentación". Estudien los archivos Lab6_files_y_strings.cpp y Lab6_Timer_functions.cpp (encuentran los archivos en Canvas). Las funciones mostradas les servirán para completar este laboratorio (y para futuros laboratorios).

Procedimiento

- Cierto texto fue dividido en dos partes. Las líneas impares del texto se guardaron en el archivo llamado **Lab6_primero.txt**, y las líneas pares se guardaron en el archivo **Lab6_segundo.txt**. Deberán crear un programa que reconstruya el texto original, lo despliegue en la terminal, y lo guarde en un tercer archivo, **Lab6_reconstruido.txt**. Para lograr el objetivo, el programa creará tres hilos (*threads*) adicionales al hilo principal (*main*), y utilizará un buffer común a los hilos. El buffer será una cadena de caracteres (un *string*). Puede ser una variable global, aunque se prefiere que sea declarada en el *main* y luego se pase como argumento a los hilos.

- Dos de los *threads* adicionales se encargarán de leer los archivos, una línea a la vez (el primer *thread* abrirá y leerá el archivo **Lab6_primero.txt** y el segundo *thread* abrirá y leerá el archivo **Lab6_segunto.txt**). Luego de leerse una línea (cadena de caracteres), ésta deberá ser guardada en el buffer común. El tercer *thread* adicional se encargará de leer las cadenas que se guarden en el buffer, y las guardará a su vez en un *string array* (arreglo o colección de cadenas). El arreglo debe ser capaz de almacenar hasta 60 cadenas. Por conveniencia, el arreglo sí será una variable global. Finalmente, el hilo principal (*main*) deberá desplegar el texto reconstruido en la terminal, y deberá guardarlo en el archivo **Lab6_reconstruido.txt**.
  Ayuda 1: en sus pruebas, pueden escribir a la terminal lo que vayan leyendo de los archivos en los hilos adicionales 1 y 2, para verificar que la lectura se vaya haciendo correctamente. Sin embargo, en la versión final del programa, sólo el hilo principal debe escribir a la terminal.
  Ayuda 2: el instructor les mostrará la estructura recomendada del *main*.

- Todos los *threads* adicionales al *main* deben inicializarse como periódicos (usando un *timer* cada uno), con la misma prioridad, y agendarse de forma alternante, de modo que los datos puedan ser leídos de los archivos y almacenados en el string array en la secuencia correcta (**sugerencia:** en los hilos que deben leer los archivos, abran primero esos archivos, y después inicialicen los *timers*). Al terminar de leerse todas las líneas de los archivos, los *threads* deben concluir su ejecución. Es decir, las tareas no durarán "para siempre", sino durarán únicamente el tiempo necesario para leer los datos de los archivos y guardarlos en el *string array*.
  Nota: Pueden usar el hecho de que los archivos no tienen más de 30 líneas cada uno.
  Ayuda 3: Consideren el siguiente seudocódigo para los *threads* adicionales:

```
Declaración de variables
Recuperación de información pasada al hilo
Asignar prioridad y política de escalonamiento
Abrir archivo (sólo los hilos 1 y 2) Configurar y arrancar el timer
Esperar a que se cumpla el primer período (que expire el timer)
Ciclo (hasta terminar de leer el archivo / llenar el arreglo)
        Hacer lo que haya que hacer
        Esperar a que expire el timer
Salir del hilo (thread.join())
```

- Deberán sincronizar las tareas ajustando sus períodos y sus tiempos de inicio. Prueben distintas combinaciones. Empiecen con períodos relativamente largos (ej. 100 o 200 ms), para asegurarse de que las tareas tengan tiempo de leer y guardar los datos en el buffer. Luego, prueben reducir los períodos. Encuentren los períodos más pequeños que puedan, tales que la reconstrucción del archivo original sea exitosa. Reporten los distintos tiempos probados, incluyendo los menores que les hayan funcionado. Generen una captura de pantalla mostrando el texto reconstruido en una terminal.

- Recuerden incluir los encabezados (.h) apropiados en sus programas (revisen los archivos.cpp proporcionados). También recuerden incluir lo necesario a la hora de compilar sus programas (*threads*).

13.1.4. Laboratorio núm. 7

Laboratorio 7

Escalonamiento de tareas, prioridades y sincronización usando *semaphores*

Objetivos

- Usar *semaphores* para sincronización de tareas.
- Observar efectos que pueden surgir al usar distintas prioridades en los hilos.
- Aprender sobre problemas y limitaciones de algunos algoritmos de escalonamiento

Procedimiento

En este laboratorio se implementará un semáforo de tránsito usando la Raspberry Pi. Usarán tres luces (LEDs) que representarán las señales para una dirección del tránsito, la otra dirección del tránsito, y una señal para peatones (vean la Figura 1). La luz encendida representa "luz verde" (carros o peatones pueden avanzar), y la luz apagada representa "luz roja" (carros o peatones deben detenerse). La luz peatonal sólo se activará si un botón (push button) se ha presionado previamente (el botón es para indicar el deseo de los peatones de cruzar la calle).

Para las luces se usarán tres puertos GPIO. Asegúrense de configurar dichos puertos como salidas. El botón será conectado a un cuarto puerto GPIO. Este puerto debe configurarse como entrada. Conecten el botón de tal forma que, al estar presionado, en el puerto se lea el valor de 1, y cuando no esté presionado, en el puerto se lea el valor de 0 (consideren un resistor de pull down). Pueden verificar que su conexión es correcta usando los comandos para una terminal de la herramienta gpio.

Nota: si el puerto no está configurado y la conexión del botón no está como se describe arriba, las funciones descritas abajo no funcionarán.

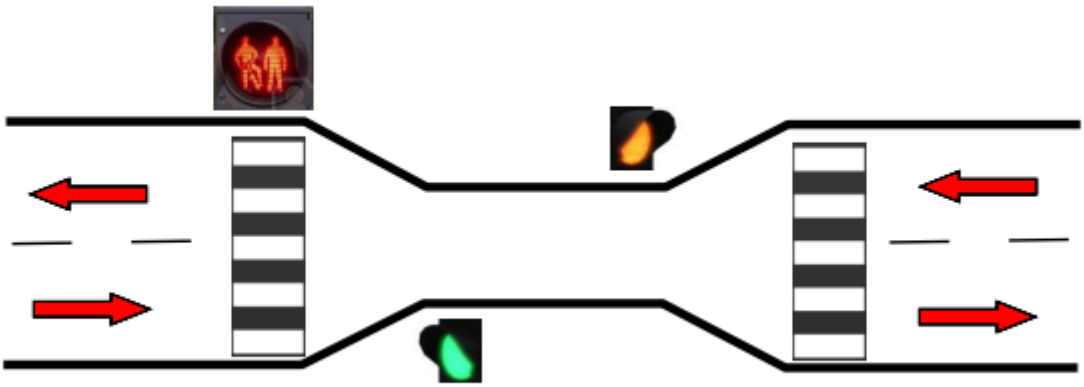


Figura 1. Esquema de las señales de tránsito y peatones.

Clases en C++

En su programa, para verificar si el botón peatonal ha sido presionado o no, deberán utilizar una clase llamada *"ButtonHandler"*. La cual se encuentra a continuación:

```
class ButtonHandler { //Se crea una clase para el control del botón
public:
    ButtonHandler() {
        pinMode(BTN1, INPUT);
        pullUpDnControl(BTN1, PUD_DOWN);
    }

    bool isButtonPressed() {
        return digitalRead(BTN1) == HIGH; // Devuelve true si el botón está
presionado
    }

    void waitForButtonRelease() {
        while (isButtonPressed()) {
            // Se espera a que el botón se suelte
        }
    }
};
```

El lenguaje de programación en C++ permite crear clases las cuales ofrecen soluciones más sencillas en cuanto a la programación orientada a objetos. Esto también ahorra el tener que utilizar librerías estáticas como se realizaba en C.

Parte 1: Escalonamiento por poleo (*polled scheduling*)

En esta parte del laboratorio, crearán un programa con un hilo que actuará como planificador. El hilo deberá tener un ciclo infinito en el que se encienda la luz correspondiente a una dirección, luego se encienda la luz correspondiente a la otra dirección, y luego se revise el estatus del botón, para determinar si se debe encender o no la luz peatonal. En ningún momento debe haber más de una luz encendida. El tiempo en que permanezcan encendidas las luces puede ser de aproximadamente 1 segundo. Usen la función **std::this_thread::sleep_for()** (NO usen *timers*).

¿Qué limitaciones tiene este método?

Parte 2: Escalonamiento por prioridades y sincronización por *semaphores*

En esta parte crearán un programa con tres hilos. Cada hilo será responsable de encender y apagar una de las luces. Para evitar que haya más de una luz encendida a la vez, deberán implementar un mecanismo de sincronización para los hilos. Esto se hará por medio de semáforos (*semaphores*). Deberán ajustar la prioridad y la política de escalonamiento de los hilos. El programa debe permitir ingresar como argumentos las prioridades de los tres hilos. Si se corre el

programa sin argumentos, las prioridades por defecto deben ser todas iguales a 1. Además de la implementación de los hilos deberán crear una clase que se encargue de controlar los LEDs es decir que tenga la capacidad de apagar y encender las luces.

El instructor discutirá la secuencia de instrucciones que los hilos deben ejecutar.

Deberán probar las siguientes configuraciones de prioridades de los hilos. Para cada caso, prueben de primero presionar el botón esporádicamente. Luego, prueben presionar el botón constante y rápidamente. Deberán reportar lo que observen.

Caso No.	Configuración de Prioridades	Observaciones
1	PL1 = PL2 = PLP	
2	PL1 = PL2 > PLP	
3	PL1 = PL2 < PLP	
4	PL1 > PL2 > PLP	
5	PL2 < PL1 < PLP	
6	PL1 < PL2 = PLP	
7	PL1 > PL2 = PLP	

Nota: PL1, PL2 y PLP son las prioridades de la luz 1, luz 2 y luz peatonal, respectivamente.

a) ¿Qué configuración(es) lleva(n) a un esquema de Round Robin?
b) ¿Hay configuración(es) que cause(n) que un hilo no llegue a ejecutar (*starvation*)?
c) ¿Se toparon con problemas al implementar sus programas? ¿Cómo resolvieron esos problemas?

13.1.5. Laboratorio núm. 8

Laboratorio 8

Comunicación por sockets

votación maestro/esclavo

Objetivos

En este laboratorio, los estudiantes aprenderán cómo comunicar y sincronizar procesos en un ambiente distribuido. El objetivo se logrará:

- Creando y usando sockets para la comunicación.
- Creando una configuración de Maestro/Esclavo (Master/Slave)

Procedimiento

En una configuración maestro/esclavo, un dispositivo tiene control sobre uno o más dispositivos adicionales. Un ejemplo es una red de computadoras, donde una de ellas (el maestro) recibe tareas y las asigna a otras computadoras (esclavos) basado en ciertos criterios. Usualmente, si no hay dispositivo maestro presente, los dispositivos existentes hacen una elección para determinar un nuevo maestro.

En este laboratorio, cada estudiante implementará un servidor que correrá en una computadora del laboratorio o Raspberry Pi (dispositivo). Cada dispositivo empezará con estatus de esclavo. Un programa cliente (implementado por el instructor) puede preguntarles a todos los dispositivos quién de ellos es el maestro, enviando el mensaje "QUIEN ES". Si ningún dispositivo responde que es el maestro, el programa cliente puede pedirles a todos que voten para determinar al nuevo maestro, enviando el mensaje "VOTE". Para votar, cada dispositivo enviará un mensaje broadcast a todos los demás dispositivos. Dicho mensaje comenzará con un símbolo # y contendrá la dirección IP del dispositivo, seguido de un espacio en blanco, seguido de un número entero aleatorio que el programa genere (ejemplos: "# 192.168.1.2 4" o "# 10.0.0.23 10", según la red).

Luego de recibir los votos, cada programa servidor deberá decidir si el dispositivo se convertirá en el nuevo maestro o no. Esto se hará comparando su propio voto con los demás votos recibidos. El voto más alto gana. Si se diera un empate, el dispositivo con la dirección IP más baja gana. Si el programa cliente envía otro mensaje "QUIEN ES", el dispositivo que se convirtió en el maestro deberá enviar un mensaje de vuelta únicamente al programa cliente (NO por broadcast) indicando que es el maestro. Este mensaje debe incluir su nombre y la dirección IP del dispositivo (ej. "Pedro en 192.168.1.2 es el master" o "María en 10.0.0.23 es la master"). **Asegúrense que las cadenas que envíen al cliente tengan un máximo de 60 caracteres. Esta es una limitación impuesta por el cliente, no por el protocolo TCP/IP.**
El servidor deberá ignorar cualquier mensaje inválido.

Especificaciones:

1.) Los mensajes son cadenas de 60 caracteres o menos.
2.) Los votos son números enteros aleatorios en el intervalo [1, 15].
3.) El número de puerto para la comunicación debe ser un argumento de su programa, es decir, cuando se corra el programa, debe ser posible proporcionar el número de puerto. Por defecto, el número de puerto será el 2000.
4.) Pueden usar el programa server_udp_broadcast.cpp como punto de partida para sus programas.
5.) La dirección IP del servidor puede ser escrita directamente en el código.

Es posible que el programa determine la dirección IP del dispositivo automáticamente. **Esto es opcional, aunque se darán puntos extra si lo logran. Asegúrense de completar todos los requisitos obligatorios antes de intentar lo opcional.**

Inicialmente, pueden probar sus programas servidor corriendo en Cygwin, y el cliente en la RPi. Si están en la misma red que sus compañeros, se sugiere que se pongan de acuerdo para no usar todos el mismo puerto cuando estén probando/depurando sus programas. También pueden pedirle al instructor que corra el cliente, para probar así sus servidores. Esto último es posible tanto en las Raspberry Pis como en las computadoras del laboratorio. Luego de sus pruebas iniciales, deberán probar sus servidores con los servidores de otros compañeros corriendo al mismo tiempo (usando el mismo puerto). En la evaluación final de este laboratorio, el instructor correrá el cliente y todos deberán correr sus servidores al mismo tiempo.

Funciones útiles:

A continuación, se listan algunas funciones adicionales a las usadas en los ejemplos cliente-servidor que les pueden ser útiles en sus programas. Estudien, compilen y corran el programa de ejemplo funciones_cadenas.cpp, en el que se ilustra el uso de las funciones (algunas ya las conocen de laboratorios y ejemplos anteriores). Se recomienda leer la descripción de las funciones, su uso y sus argumentos. Hay muchas otras funciones para manejo de cadenas que son útiles. **Siempre agregar 'std::' antes de llamar a estas funciones.**

strcmp, strncmp, strcpy, strtok, sprintf, atoi, srand, rand

13.2. Resultados del rendimiento de programas

Rendimiento de programas en distintos sistemas operativos de Linux.

RockyLinux

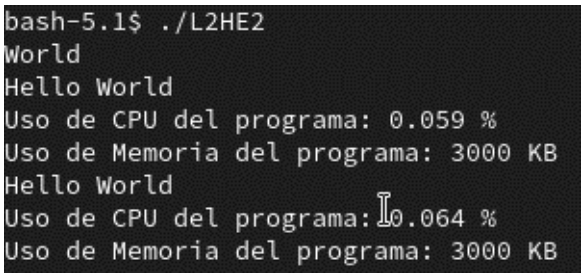


Figura 17: Resultados programa en C

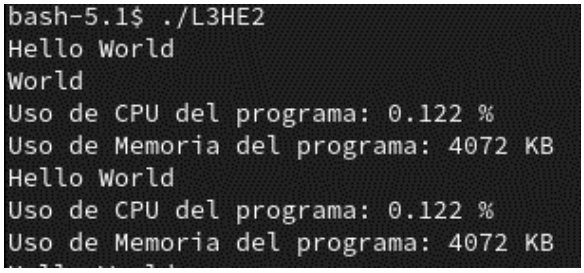


Figura 18: Resultados programa en C

Ubuntu

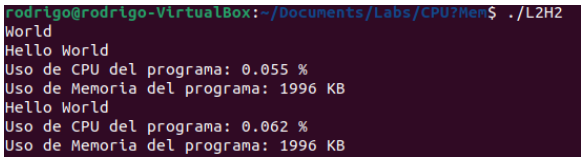


Figura 19: Resultados programa en C

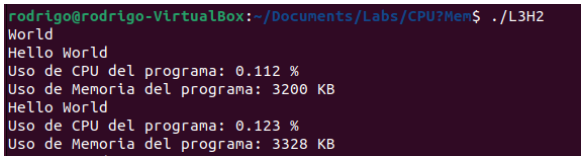


Figura 20: Resultados programa en C

Debian

```
rodrigo@rodrigo:~/Documents/OneDrive_1_9-15-2023$ ./L2H2
World
Hello World
Uso de CPU del programa: 0.064 %
Uso de Memoria del programa: 2352 KB
Hello World
Uso de CPU del programa: 0.072 %
Uso de Memoria del programa: 2352 KB
```

Figura 21: Resultados programa en C

```
rodrigo@rodrigo:~/Documents/OneDrive_1_9-15-2023$ ./L3H2
World
Hello World
Uso de CPU del programa: 0.102 %
Uso de Memoria del programa: 2352 KB
Hello World
Uso de CPU del programa: 0.113 %
Uso de Memoria del programa: 2352 KB
```

Figura 22: Resultados programa en C

Oracle

```
[rodrigo@localhost OneDrive_1_9-15-2023]$ ./L2H2
World
Hello World
Uso de CPU del programa: 0.047 %
Uso de Memoria del programa: 2804 KB
Hello World
Uso de CPU del programa: 0.052 %
Uso de Memoria del programa: 2804 KB
```

Figura 23: Resultados programa en C

```
[rodrigo@localhost OneDrive_1_9-15-2023]$ ./L3H2
World
Hello World
Uso de CPU del programa: 0.110 %
Uso de Memoria del programa: 2804 KB
Hello World
Uso de CPU del programa: 0.117 %
Uso de Memoria del programa: 2804 KB
```

Figura 24: Resultados programa en C

13.3. Repositorio de Github

Se creó un repositorio en Github con el propósito de almacenar todos los códigos utilizados en los laboratorios, el proyecto y la interfaz, así como las guías de laboratorio. Incluye algunos de los códigos de los trabajos realizados en lenguaje de programación C. El repositorio contiene archivos correspondientes a cada uno de los elementos mencionados. A continuación, se proporciona el enlace al repositorio: https://github.com/gar19085/Tesis_-Rodrigo_Garcia