Desarrollo de librerías y herramientas de software para programación multihilos, multiprocesos, escalonamiento, comunicación interprocesos y protocolos de red en el lenguaje Python 3

Carlos Ricardo Cuellar Klingenberger

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 interprocesos y protocolos de red en el lenguaje Python 3

Trabajo de graduación presentado por Carlos Ricardo Cuellar Klingenberger 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 interprocesos y protocolos de red en el lenguaje Python 3

Trabajo de graduación presentado por Carlos Ricardo Cuellar Klingenberger 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 Manuel Morales

(f) 

MBA. Alejandra Carrillo

Fecha de aprobación: Guatemala, 13 de Enero de 2024.

Prefacio

La existencia de este trabajo hubiera sido imposible sin el apoyo de mis padres, Carlos Aroldo Cuellar y Cindy Lorena Klingenberger. El apoyo que han sido en cada uno de estos cinco años es imposible de explicar por medio de palabras. Mi madre, la que desde el primer día de colegio hasta el último día de universidad se ha encargado de estar atrás de mí, preparada para levantarme en los momentos en que caí y al mismo tiempo estando cerca para ser la primera en celebrar todos los éxitos y cada paso que me llevaron a estos. Mi padre, el que es mi ejemplo de vida y se ha encargado de enseñarme el camino de lo que un día aspiro ser, su paciencia y valores han sido los dos apoyos que me han guiado durante todo este proceso.

Agradezco a mis amigos de la carrera, Francisco Montúfar, Rodrigo García y Carlos Gil. Aquellos con las que compartimos incontables noches de desvelo y con los que siempre he contado que me darán una mano para superar cada reto que este proceso universitario nos ha dado.

Agradezco a Renin Cabrera, el maestro de física fundamental que en mis años de bachillerato me inculcó en el camino de la ciencia. Su gran capacidad de docencia y pasión por este campo fueron fundamentales para que descubriera mi capacidad dentro del mundo de la ingeniería. La marca que dejó en mi será de por vida.

Índice

Lista de figuras

Lista de códigos

Resumen

El siguiente trabajo de graduación busca la implementación didáctica de herramientas diversas de programación para el lenguaje Python 3. Por medio de un estudio previo, se identificaron librerías útiles y vigentes para las competencias desarrolladas en el curso de Electrónica Digital 3 de la Universidad del Valle de Guatemala. Con estas librerías se busca replicar los laboratorios del curso en el lenguaje Python en lugar de C, ya que estas cubren funciones básicas dentro del amplio alcance que tiene el lenguaje. En el momento de realizar esa traducción se encontraron oportunidades de mejora donde existió la oportunidad de desarrollar librerías propias, para complementar la selección de herramientas previamente estudiadas. Añadido a eso, se desarrolló una herramienta para la automatización de la inclusión de microcontroladores orquestados desde la Raspberry Pi con una base de datos de SQL. Toda herramienta desarrollada, se hizo acoplando el paradigma de la Programación Orientada a Objetos (POO), implementando el uso de clases y métodos. Todo el código desarrollado fue debidamente documentado y publicado en un repositorio para que esté al alcance de los futuros estudiantes y cualquier otro tipo de comunidad.

Abstract

The following graduation project aims to implement didactic programming tools for the Python3 language. Through a preliminary study, useful and up-to-date libraries were identified for the skills developed in the Digital Electronics 3 course at the Universidad del Valle de Guatemala. These libraries are used to replicate the course laboratories in the Python language instead of C since they cover basic functions within the extensive scope of the language. During this translation, opportunities for improvement were identified, leading to the development of custom libraries to complement the previously studied tools. In addition to that, a tool was developed for the automation of microcontroller inclusion orchestrated from the Raspberry Pi with an SQL database. All developed tools were created using the Object-Oriented Programming (OOP) paradigm, implementing the use of classes and methods. The entire code developed was properly documented and published in a repository for the benefit of future students and other communities.

CAPÍTULO 1

Introducción

La existencia de múltiples lenguajes de programación otorga a los desarrolladores un alto grado de versatilidad que permite la elección a estos de seleccionar distintas maneras de atacar un problema, tanto a nivel de *software* como *hardware*. Pero, a pesar de que la preferencia personal pueda permitir a cada desarrollador elegir el lenguaje que desee para cumplir su objetivo, es irrefutable el hecho que cada lenguaje tiene un área de especialización distinta. Es de esta premisa de donde nace la razón de desarrollar este trabajo de graduación, en donde se busca poner en contraste las virtudes de dos de los lenguajes más populares a día de hoy, C y Python.

El lenguaje C desde sus inicios se ha caracterizado por su velocidad a la hora de compilar, al igual que al momento de ejecución. Esto se debe a su naturaleza de ser un lenguaje compilado, en lugar de interpretado (como lo es Python). Esta condición significa que el código es directamente convertido a código máquina, pudiendo interactuar directamente con el CPU del sistema. Esto también abre otro abanico de alcances, como poder interactuar de manera directa con elementos de *hardware*. Un ejemplo de esto es la interacción directa con los elementos de memoria por medio de los punteros, los cuales pueden hacer referencia hacia las localidades específicas en donde se almacenan datos.

Entre las características mencionadas anteriormente, existen bastantes otras que le dan a C el prestigio y popularidad que tiene. A pesar de las ventajas, estas mismas, llegan a ser los síntomas de sus desventajas. El hecho de que sea un lenguaje compilado limita su portabilidad, debido a esa misma condición de que el código es directamente convertido a código máquina, un *script* desarrollado con un compilador "K"sobre el sistema operativo "X", no tendrá el mismo código máquina si fuera a ser compilado por el compilador "B"sobre el sistema operativo "Y", forzando así a que el *script* tenga que ser compilado múltiples veces para ser portado y no todos los compiladores tendrán compatibilidad con herramientas y librerías empleadas desde la concepción original del código. Es aquí donde entran a escena los lenguajes interpretados, siendo su mayor representante Python.

Los lenguajes interpretados se caracterizan por tener un interpretador. La función de este es que a medida a que el código se ejecuta línea por línea, el interpretador corre el comando

respectivo, para que después este interactué de la manera indicada con el CPU del sistema. Esto hace que todo el código escrito en un lenguaje interpretado sea portable a todo sistema operativo compatible, ya que, debido a que el interpretador es el encargado de interactuar con los recursos del CPU, este es capaz de siempre tener la manera correcta de consumir el recurso apropiado que demanda el código. Sin embargo, este "mediador"también es el culpable de establecer las limitantes y desventajas de estos lenguajes. Debido a la interacción indirecta con el CPU, no se puede actuar directamente sobre elementos de hardware, como con los punteros. También esta constante evaluación que realiza el interpretador ante cada línea de código, hace que el tiempo de ejecución sea considerablemente mayor ante el tiempo de un lenguaje compilado.

Es por eso que este trabajo busca evidenciar las maneras en que las ventajas de Python facilitan la implementación de funciones complejas, que van desde el desarrollo de código multihilos, hasta la implementación de protocolos de comunicación de red y el manejo de *strings*. Para esto se realizó un estudio de múltiples librerías existentes en Python que permiten la integración de estas funciones al desarrollo de código. Para aquellas funciones a las que no se encontró una librería previamente construida, se realizó el respectivo desarrollo de una. Una vez recopiladas todas las librerías para el estudio, se realizaron pruebas de portabilidad en diversos sistemas operativos (Windows y Linux) para luego realizar *benchmarks* en donde se centraron la velocidad de ejecución y consumo de CPU como las variables de estudio. Seguido a esto, se buscó poner de manera práctica el uso de las librerías, donde se les dieran usos más realistas orientados a aplicaciones. Para esto se tomó como base el curso de Electrónica Digital 3 de la Universidad del Valle de Guatemala, buscando "traducir"sus laboratorios, los cuales están escritos en C, para poder ejemplificar los hallazgos encontrados. Finalmente, bajo la misma línea didáctica, se desarrolló una herramienta basada en Python que automatice y optimice la interacción de sistemas embebidos de manera directa con bases de datos de SQL.

CAPÍTULO 2

Antecedentes

2.1. Curso de Electrónica Digital 3

Este proyecto está orientado a la renovación del curso de Electrónica Digital 3. Por lo mismo, el contenido teórico y práctico de este será la base principal del proyecto. En este curso se cubren diversas áreas de desarrollo de código en C para la Raspberry Pi (RPi). Se imparten temas generales como el uso básico de los pines digitales de entrada y salida, hasta áreas más complejas como programas multihilos y programación de *sockets*.

La primer competencia del curso consiste en la familiarización con la RPi y la funcionalidad de sus pines por medio de la librería de *WiringPi*. Dicha librería tiene como propósito dar acceso hacia los pines GPIO de la RPi. Esta desarrollada para implementarse en el lenguaje C, y no es apta para principiantes. Busca brindar una plataforma de acceso a los pines y sus funciones (ADC, PWM, Serial, I2C...) similar a la que manejan las plataformas de desarrollo de Arduino. De igual manera cubre la comprensión e implementación de sistemas multihilos y multiproceso. La segunda competencia se enfoca en la familiarización con el desarrollo de código en lenguaje C. Esto se aborda por medio de la introducción de funciones nativas del lenguaje para el manejo de distintos tipos de variables. Se hace bastante énfasis en el manejo de *strings* y su manipulación como arreglos de caracteres. Finalmente la tercer competencia que se aborda es la interconexión de la RPi hacia otros dispositivos. Las interconexiones estudiadas van desde protocolos de comunicación inalámbrica (TCP Y UDP) por medio de *sockets*, hasta interconexiones físicas por medio de circuitos hacia microcontroladores.

2.2. Uso de la Raspberry Pi en investigación y experimentación

En el 2015 se realizó una experimentación del uso de la RPi como un servidor para la recolección y gestión de datos. El servidor diseñado para la RPi, debía de poder alojar el

acceso de múltiples computadoras para que estas pudieran extraer la información acorde a los permisos que el mismo servidor les concediera. Para la recolección de datos, se diseñaron redes Wi-Fi y Zigbee, para poder tener una comunicación inalámbrica. Este prototipo busca reflejar el ahorro en cableado e infraestructura que los protocolos de interconexión física supone en la industria. Sin embargo, los sensores para los cuales esta diseñado el proyecto deben de monitorear procesos no críticos, por el retardo o caídas que puedan sufrir las redes inalámbricas. La estructura del servidor es bastante versátil, por lo que su diseño permite su aplicación en procesos industriales, hasta el manejo de datos totalmente digitales, como lo podrían ser las notas de un centro de estudios [1].

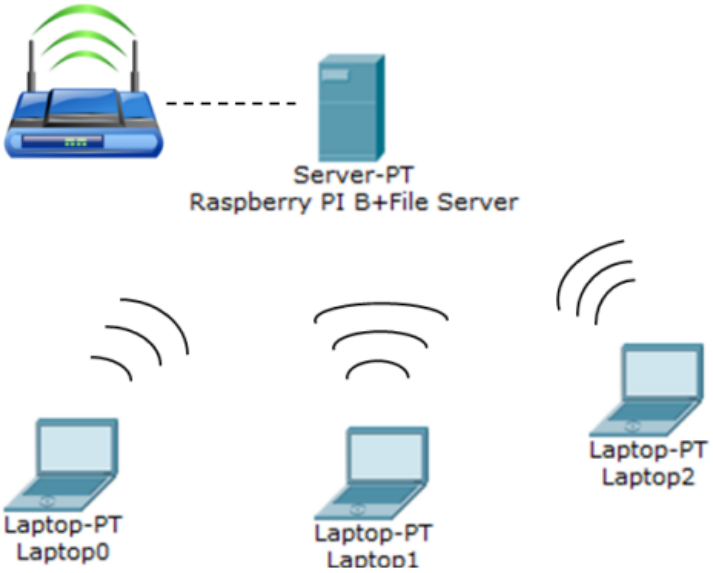


Figura 1: Representación de la red usando a la RPi como servidor de datos

Para la industria biomédica se desarrolló un sistema de recolección de datos de Electrocardiogramas (ECG) y otros procedimientos basado en la RPi. A nivel general la RPi debe subir los datos a una base de datos en tiempo real que pueda advertir al personal médico de inconsistencias que puedan suponer un riesgo al paciente. Esto lo logra por medio de dos procesos distintos. El primero consiste en la comunicación hacia un módulo de MySQLdb para lograr la comunicación hacia la base de datos. El segundo proceso consiste en la implementación de un módulo GSM para poder llevar a cabo los avisos de alerta. Este proyecto tiene mucha área de crecimiento ya que la información recabada puede ser usada para el desarrollo de algoritmos de predicción de patrones más complejos que puedan describir la naturaleza de las enfermedades [2].

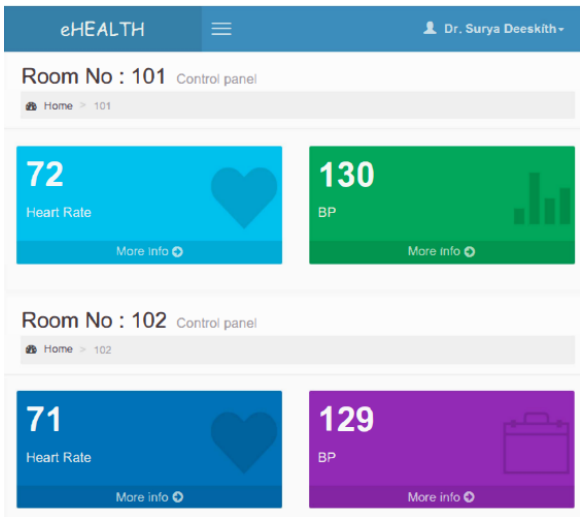


Figura 2: Interfaz de la aplicación de datos biométricos alojada por la RPi

2.3. Desarrollo de código de Python en la Raspberry Pi

En el 2019, se realizó un estudió en India acerca de la calidad de agua potable de distintas fuentes. Los criterios para verificar la potabilidad del agua se basaron en los lineamientos de la *Central Pollution and Control Board* (CPCB). Por lo que los datos recolectados por los sensores se sometían a una comparación con un *benchmark* basado en la información de la CPCB para la autenticación de los resultados. Los sensores se organizaron en un arreglo multisensor (MSA, por sus siglas en inglés). Dicho arreglo estaba compuesto por sensores de pH, conductividad eléctrica, oxígeno disuelto y reducción potencial de oxidación. Estos sensores estaban controlados directamente por la RPi. Finalmente se desarrolló en Python una interfaz gráfica en la que el usuario pudiera acceder de manera inmediata al sistema de medición. Dicha interfaz se despliega en una pantalla táctil LCD, que de igual manera esta conectada directamente a la RPi [3].

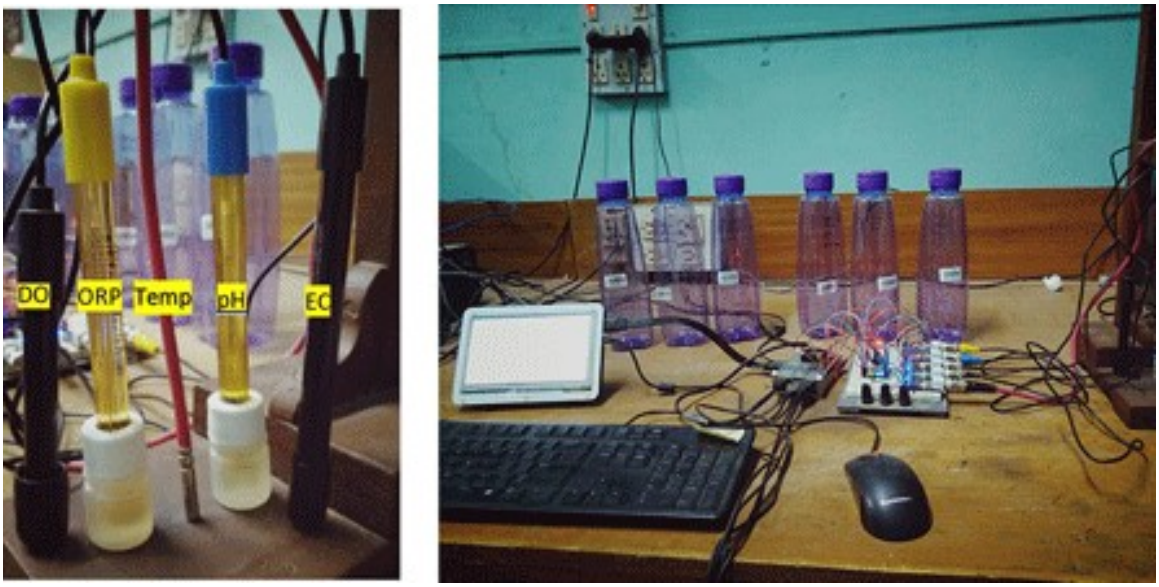


Figura 3: Estación de medición de calidad de agua

CAPÍTULO 3

Justificación

Este proyecto nace de la necesidad de mantener vigente el formato y contenido del curso de Electrónica Digital 3, de la Universidad del Valle de Guatemala. Dicho curso desde su origen se ha impartido orientado hacia el desarrollo de código para sistemas embebidos en el lenguaje C. Sin embargo, dicho lenguaje con el paso del tiempo ha perdido relevancia por ser de bajo nivel. Ante la aparición de lenguajes de más alto nivel, como Python, el desarrollo de herramientas de software nuevas se ha concentrado en ellos, y su uso en la industria se ha popularizado con fuerza. Por lo tanto, al plantear una transición de las herramientas actuales del curso hacia Python, implica una serie de oportunidades para los estudiantes, de poder usar los conocimientos impartidos (programación multihilos, comunicación interprocesos y protocolos de red), junto a herramientas nuevas exclusivas a este lenguaje, haciendo que las competencias adquiridas no solo sirvan como una puesta en práctica de los conceptos, sino que sean herramientas utilizables en la vida profesional actual.

Python tiene otras ventajas sobre C, no solo el tema de su vigencia y modernidad. También su naturaleza de alto nivel permite la simplificación de tareas. Una labor complicada y tediosa en C, como lo podría ser abrir un canal de comunicación de *sockets* o el procesamiento de *strings*, en Python son tareas de suma facilidad. Por lo tanto, al trasladar los laboratorios a Python, permite que el estudiante pueda enfocar su tiempo de estudio hacia las dinámicas importantes de las competencias y que no tenga que repartir su tiempo de trabajo elaborando tareas de menor relevancia como suele ser en el desarrollo en C.

Desde un punto de vista didáctico, Python brinda al curso de electrónica digital 3 la oportunidad de pertenecer a la vanguardia en cuanto a las fronteras del desarrollo de software concierne. Muchas de las últimas innovaciones centran sus herramientas de desarrollo, como API's, orientadas a Python. Por lo que al renovar el curso hacia este lenguaje, permitirá brindar a los estudiantes las herramientas para poder innovar con los últimos y más innovadores avances del sector.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Desarrollar librerías de Python usando programación orientada a objetos para el desarrollo de aplicaciones que tengan las funcionalidades multihilos, multiprocesos, escalonamiento, comunicación interprocesos y protocolos de red.

4.2. Objetivos específicos

- Evaluar y analizar las librerías disponibles en Python que permiten la conexión entre procesos por medio de conexiones de red.
- Evaluar y analizar las librerías disponibles en Python para implementar algoritmos de escalonamiento, sincronización y comunicación interprocesos
- Verificar el funcionamiento de las librerías de programación multihilos y multiprocesos en Windows, Rocky Linux y Raspbian.
- Comparar el rendimiento por medio de *benchmarks* de las librerías de programación multihilos y multiprocesos entre los lenguajes de Python y C.
- Replicar los laboratorios del curso de Electrónica Digital 3 en Python usando las librerías desarrolladas.
- Adaptar las guías de laboratorio actuales para la elaboración de las prácticas usando Python.
- Desarrollar herramientas para Python para facilitar la interacción con el entorno de bases de datos de SQL.
- Crear un repositorio debidamente documentado, para la implementación versátil de las librerías.

CAPÍTULO 5

Alcance

Con el amplio estudio de librerías realizado, se tiene una amplia base para cubrir el desarrollo de múltiples aplicaciones pertenecientes a distintos ámbitos. Esto debido a que las áreas cubiertas con este trabajo son los pilares de distintas ramas de especialización. Se hace especial énfasis con el material desarrollado para la unificación de sistemas embebidos y bases de datos, ya que aquí llega a converger la gran mayoría de temas cubiertos en el estudio de selección de librerías realizado.

Sin embargo, a pesar de haber buscado otorgar al trabajo realizado el mayor grado de versatilidad posible, siempre se trabajó dentro de unos límites bastante claros. El primero de ellos son los lenguajes trabajados. Se hace énfasis en las diferencias entre los lenguajes de C y Python, pero vale la pena resaltar que estos no son los únicos lenguajes compilados e interpretados. La dinámica de los experimentos realizados podría ser aplicada para contrastar una mayor cantidad de lenguajes y esto podría ayudar a extraer conclusiones más completas acerca de la diferencia entre los tipos (compilado/interpretado) de lenguajes de programación, especialmente en la sección de *benchmarking* del proyecto. La razón por la que se optó por realizar únicamente la comparación entre C y Python, fue por la facilidad que brindaba el material preexistente del curso de Electrónica Digital 3. Esto brindaba una plataforma eficaz para poder realizar las comparaciones, debido a que ya se tenía la dinámica de cada laboratorio a traducir establecida, y para no descuidar el enfoque didáctico era conveniente enfocarse en un solo lenguaje para poder realizar guías de calidad que pudieran explorar a fondo las funciones de Python expuestas.

La otra frontera claramente delimitada es el uso de bases de datos de SQL Server. Se optó por usar estas debido a su fácil integración y despliegue de bases de datos con las herramientas de *SQL Server Management Studio* (SSMS) y *SQL Server Integration Services (SSIS)*. Estas herramientas permiten una integración y automatización especializada para las bases de SQLServer, y no brindan un soporte tan profundo para otro tipo de bases como MySql.

CAPÍTULO 6

Marco teórico

6.1. Conceptos

6.1.1. *Semaphores*

Un *sempahore* es una estructura de datos que consiste en un identificador, un contador y una cola. Su función principal consiste en el bloqueo de procesos. Una vez bloqueado el proceso este entra a la cola y el contador monitorea la cantidad de procesos bloqueados. El *semaphore* mantendrá a los procesos controlados en pausa, hasta que el proceso encargado de señalizar, indique la liberación de cada proceso correspondiente. Este funcionamiento permite que sean una herramienta eficaz para la sincronización de tareas y escalonamiento de procesos [4].

6.1.2. Procesos

Un proceso, dentro de la terminología Unix hace referencia a un programa que está siendo ejecutado. Para que un CPU pueda ejecutar varios procesos en simultáneo debe de realizar cambios de contexto. Esto significa que el sistema operativo correspondiente, interrumpe el proceso activo y guarda su estado actual, para ejecutar otro de los procesos dentro de la lista de tareas generada por las demandas del usuario. Esto se realiza de manera cíclica según temporizadores dentro del CPU. De esta manera se genera la ilusión de una operación multiprocesos del CPU. En los CPU que poseen más de un núcleo, la dinámica de ejecucción multiprocesos es la misma para cada núcleo, pero núcleos distintos son capaces de trabajar en paralelo. Por lo tanto, se puede concluir que tareas asignadas en distintos núcleos si pueden correr genuinamente en paralelo, pero toda tarea asignada a la cola de un mismo núcleo siempre tendrá que ser escalonada [5].

6.1.3. Escalonamiento

Por escalonamiento se entiende a la labor del sistema operativo de ordenar los procesos a ejecutar y cómo se organizará la alternancia de estos. El escalonamiento busca optimizar el funcionamiento de la administración de recursos y tiempos de ejecución del CPU. El sistema operativo es el encargado de realizar el escalonamiento, por medio de este evitar situaciones como la inanición o condiciones de carrera. Para esto, el sistema operativo, debe realizar un escalonamiento ordenado que garantice una sucesión de tareas rápida y segura. Para esto, existen distintos algoritmos como *First-Come-First-Serve*, *Shortest-Job-Scheduling*, *Round Robin scheduling*, entre otros. Este último es de los algoritmos más populares y se usa con frecuencia en la industria por su capacidad de maximizar tiempos y recursos [6].

6.1.4. Hilos

Un hilo es similar a un proceso, e incluso se podrían considerar de la misma naturaleza dependiendo del sistema empleado para generar los hilos. Una ventaja universal de los hilos sobre cualquier proceso es el hecho de que en general ocupan menos memoria y necesitan menos tiempo para inicializarse. En el contexto de software generado por Python, los hilos pueden ser interrumpidos en cualquier momento. Otra característica de los hilos es su capacidad de generar hilos hijos dentro de sí mismos. Esto permite que se pueda desarrollar software multihilos, permitiendo así la elaboración de múltiples tareas en simultáneo [7].

6.1.5. Hilos POSIX

En un principio, cada desarrollador de hardware diseñaba su propia versión de sistemas de hilos. Dado a que entre las versiones de los distintos fabricantes habían diferencias significantes, era bastante difícil para los desarrolladores de software crear aplicaciones utilizables en hardware de diversos fabricantes. Para aprovechar el potencial que presentan los hilos, se estandarizó la interfaz de programación, creando así el estándar *Portable Operating System Interface* (POSIX). El estándar POSIX se ha continuado desarrollando desde su aparición, y dentro de él se encuentra la especificación de los *Pthreads*. Estos últimos se definen como un set especializado para la implementación en el desarrollo de código en C. En cuanto a Python, su interpretador permite conducir las solicitudes de hilos hacia el estándar POSIX, ya que permite que los hilos sean administrados por el sistema operativo huésped [8].

6.1.6. Implementación de hilos en Python

El uso de hilos en Python es ineficiente por diversas razones. La primer característica a destacar del uso de hilos en este lenguaje es el hecho que el interpretador de Python no tiene ningún algoritmo para el escalonamiento de hilos. Por lo tanto, el escalonamiento y cambio de contexto de los hilos generados en Python queda en manos del sistema de escalonamiento del sistema operativo huésped. El software multihilos esta diseñado para aprovechar redundancias de hardware, maximizar el rendimiento y reducir tiempos de ejecución, no obstante, los hilos dentro de Python no pueden satisfacer estos principios. Esto se debe a que en el

interpretador de Python solo puede haber un hilo activo a la vez. Cada hilo generdado en Python comparte una estructura de datos global llamada el bloqueo general de interpretador (GIL por sus siglas en Inglés). El GIL se implementa con un *mutex* (objeto que permite la exclusión mutua) y una variable condicional. Esto garantiza que el hilo que este corriendo reciba acceso exclusivo a los recursos del interpretador. Así que solo el hilo con el GIL habilitado es capaz de ejecutarse, mientras todos los demás esperan a que el sistema de escalonamiento del huésped habilite el GIL para ellos [9].

6.1.7. *Pipes*

Los acercamientos a la comunicación interprocesos, suelen ser provistos por el sistema operativo huésped. Dentro de los mecanismos más utilizados se encuentran los *pipes*. Estos se proveen en prácticamente todo tipo de distribución basada en Linux y por naturaleza son unidireccionales, donde un proceso A manda información para que un proceso específico B sea capaz de leerla. Para poder usar los *pipes* de manera satisfactoria, los programas a comunicarse deben de tener un ancestro en común para entablar el canal de comunicación. El flujo de datos enviados se comunica en un *buffer*, que ordena la información según el orden en que llegan al otro extremo del *pipe*. Para poder crear una comunicación bidireccional usando este mecanismo lo ideal sería establecer dos canales distintos. Otra opción, dado a que debe de existir dentro de un mismo contexto de ancestros, ambos programas involucrados tendrían información acerca del canal original establecido y se podría revertir el flujo de la comunicación. Sin embargo, este es un proceso complicado y por lo tanto lo recomendable siempre va a ser el establecimiento de canales en pares según sea necesario [10].

6.1.8. *Sockets*

Se conoce como *socket* como el punto final de un flujo de comunicación interprocesos que se da a través de una red. Debe de tener una dirección IP al igual que un número de puerto asignado por el sistema operativo de la computadora. Es la barrera entre la capa de aplicación y el protoclo de comunicación empleado (como lo podría ser TCP o UDP). Un *socket* debe de cumplir cuatro operaciones fundamentales: establecer un canal de comunicación hacia otra computadora, enviar datos, recibir datos y cerrar la conexión. El puerto de un *socket* es un identificador cuya función es apuntar la información recibida por el *socket* hacia un proceso específico dentro de la máquina en que se esta ejecutando. Existen números de puerto reservados para servicios específicos como lo puede ser *telnet* o *smtp*, por lo que al construir un canal de comunicación por medio de *sockets* hay que tener bastante cuidado. La comunicación por medio de *sockets* es empleada con bastante frecuencia en los modelos de servicios Cliente-Servidor. Donde el cliente es el encargado de crear la solicitud para abrir el canal, para solicitar al servidor los distintos recursos o datos que este pueda proveer. Los *sockets* permiten establecer comunicaciones versátiles debido a su compatibilidad de poder ser usados por diversos protocolos de red. Son una herramienta fundamental para la existencia de comunicación interprocesos a través de una red [11].

6.2. Dispositivos

6.2.1. Raspberry Pi 3B

Esta iteración de la Raspberry Pi brinda una plataforma bastante capaz y robusta, tanto para aprendizaje como desarrollo de proyectos orientado hacia los sistemas embebidos. Esta computadora cuenta con un CPU Cortex-A53 de cuatro núcleos y 64 bits, con una frecuencia de reloj de 1.2Ghz. Cuenta con una serie de 40 pines GPIO capaces de manejar señales digitales y analógicas. Esta versión de la placa, cuenta con módulos para establecer conexiones por medio de Wi-Fi y Bluetooth (al igual que un puerto de Ethernet). El fabricante da soporte a un sistema operativo oficial, llamado Raspbian. Esta es una distribución derivada de Debian, perteneciente a la familia de sistemas Linux. A pesar de lo anterior, la plataforma es capaz de correr otros sistemas operativos tanto de Windows, como diversas distribuciones de Linux. En cuanto a memoria, tiene una ranura para tarjetas MicroSD. La mayor mejora en cuanto a versiones anteriores es el rendimiento, al tener un controlador LAN capaz de manejar velocidades mayores a 200 Mbps [12].

6.3. Protocolos

6.3.1. SPI

La interfaz serial de periférica (SPI, por sus siglas en inglés) es un estándar de comunicación de hardware usado en el desarrollo de sistemas embebidos. El SPI no usa un protocolo estándar y solo transmite paquetes de datos en comunicaciones dúplex, haciéndolo ideal para comunicaciones de datos extensos. Usa un máximo de cuatro líneas de comunicación: El control de relok (SCK), la selección de chip (CS), la salida de maestro y entrada de esclavo (MOSI) y la entrada de maestro y salida de esclavo (MISO), la estrucutra permite la existencia de que un sistema tenga un maestro pirmario y uno secundario. El SPI esta abierto a dos tipos de conexión. Una conexión directa compuesta de 3 buses (SCK, MOSI y MISO) conectados directamente al maestro y todos los esclavos y cada esclavo tiene una bus individual directo al maestro para la línea de CS. También se pueden realizar conexiones en cadena donde el único bus compartido es el SCK y el CS. Donde la línea de MOSI se segmenta en pares entre cada nodo del sistema y solo existe una línea de retorno del MISO del último miembro de los eslcavos hacia el maestro. Debido a no estar ligada a las normas habituales de un protocolo, el SPI siempre será el sistema de comunicación predilecto para aplicaciones con sensores robustos de alta precisión por la longitud de los datos a transmitir [13].

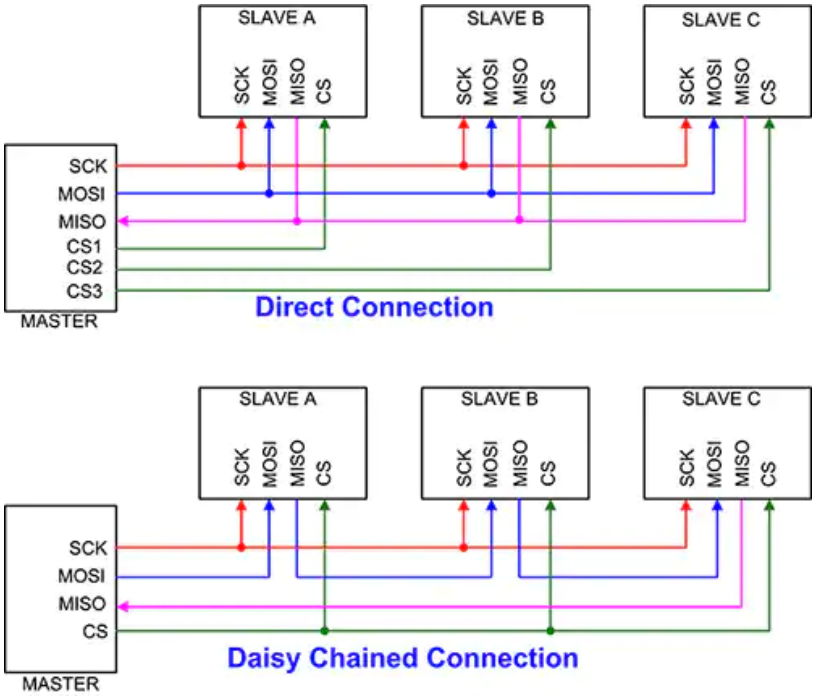


Figura 4: Diagramas de conexión de circuitos SPI

6.3.2. I2C

El protocolo *Inter-Integrated Circuit* (I2C), es usado en comunicaciones seriales físicas entre dispositivos. Es capaz de sostener la comunicación de dos o más circuitos distintos. El I2C es capaz de establecer comunicación a 2 vías de una manera simple y entre varios nodos de un sistema. Su simpleza radica en el uso de dos cables bidireccionales para la transmisión y envío de datos. La estructura del protocolo permite la conexión de múltiples maestros y múltiples esclavos. Fue desarrollado por Phillips con el fin de comunicar diversos componentes residentes de la misma placa. La interfaz consiste en dos cables principales, la línea de datos serial (SDA) y la línea del reloj serial (SCL). Ambas líneas requieren de una resistencia de *pull-up*. Cuando las líneas se encuentran en un estado de inactividad, las resistencias mencionadas fuerzan a que estas estén en un estado alto. Por lo tanto, los bits se reconocen en flancos de bajada. El paquete de transmisión de I2C siempre tiene en su encabezado la dirección a la cuál va dirigido el mensaje, pudiendo ser una solicitud de lectura o escritura. Ya que las líneas de comunicación están compartidas por todos los nodos del sistema, la sección de dirección del encabezado es fundamental, ya que cada nodo siempre revisa esta parte y solo responderá a la solicitud del maestro en caso la dirección del paquete coincida con su propia dirección [14].

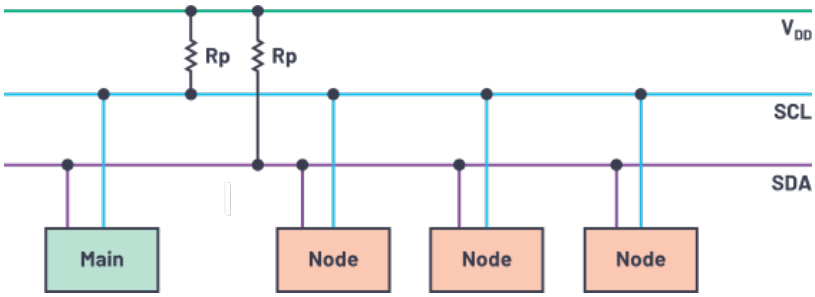


Figura 5: Diagrama de conexión de un circuito i2c

6.3.3. TCP

El protocolo de control de transmisión (TCP, por sus siglas en inglés) es un estándar de comunicación interprocesos a través de una red. Este protocolo está diseñado para asegurar la distribución exitosa de paquetes a través de las redes. Es uno de los protocolos de comunicación de red más utilizados hoy en día. La característica principal es su garantía de proteger la integridad de los datos que se estén transmitiendo. Antes de empezar a transmitir datos, el TCP se encarga de establecer una conexión entre el punto de origen y el destino. Dada a la seguridad del protocolo, muchos protocolos de alto nivel usan TCP como base, algunos ejemplos son HTTP, SMPT y SSH. Este protocolo rompe los datos a enviar en paquetes más pequeños para garantizar la integridad de estos. Para establecer la conexión, el TCP usa un *handshake* de tres vías entre el cliente y el servidor.

Este es un protocolo orientado a conexión. A parte de asegurarse de establecer un canal entre las partes involucradas, es capaz de detectar paquetes perdidos y solicitar su reenvío para tener la información enviada completa en el receptor. Su arquitectura se categoriza bajo el modelo cliente servidor. Donde el servidor siempre está escuchando para entablar posibles conexiones y el cliente es el encargado de enviar la solicitud para entablar esta misma. El TCP incursiona en múltiples capas del modelo OSI, principalmente en la capa de aplicación y transporte. El *handshake* de tres vías consiste en el siguiente orden:

- Paquete SYN: el cliente envía este paquete al servidor a manera de solicitud de inicialización de un canal de comunicación.
- Paquete SYN-ACK: el servidor envía este segundo paquete, reconociendo que recibió la solicitud y que esta dispuesto a entablar la comunicación con el cliente.
- Paquete ACK: el último paso del *handsahke*, lo envía el cliente después de haber recibido el SYN-ACK, para acceder a la comunicación y se establece la conexión bidireccional entre ambas partes [15].

6.3.4. UDP

El protocolo de datagrama de usuario (UDP, por sus siglas en inglés), es usado en situaciones donde se prioriza la baja latencia, haciendo este protocolo ideal para solicitudes DNS, *streaming* de contenido multimedia, *gaming*, entre otros. En este protocolo la transmisión de

datos inicia sin que el participante a recibir datos acceda a conectarse. Esto último es una de las mayores diferencias respecto al TCP, y por lo mismo la integridad y seguridad de los datos no se garantiza, haciendo que las aplicaciones a usar este protocolo sean capaces de funcionar a pesar de la perdida esporádica de datos. El UDP no toma en cuenta condiciones como el tiempo de llegada o la duración de transmisión para realizar el envío de paquetes. El UDP no necesita de confirmación de llegada de los paquetes y el orden de llegada a estos es irrelevante para el protocolo. Dentro del encabezado de los paquetes UDP se encuentra información relevante de cómo se transmiten los paquetes: el puerto de destino, el puerto de origen, el largo en bytes del paquete a enviar y el *checksum* para que el receptor revise la integridad del contenido del paquete. A continuación, se describe la operación del protocolo: [16]

- Encapsulación de datos: la información de la aplicación del emisor de datos, se separa en unidades más pequeñas llamadas datagramas. Estas unidades se insertan dentro del encabezado del UDP.
- Transmisión: se envían los paquetes a través de la red hacia la dirección IP de destino. Los paquetes de una misma comunicación son susceptibles a tomar distintos caminos y por lo mismo llegar en distinto orden hacia su destino. Esta misma naturaleza, hace que los paquetes se pierdan. Por la arquitectura del protocolo si esto llega a pasar no existe un método de recuperación de paquetes. Por medio del *checksum*, el receptor podría identificar la falta de información, pero no podría volver a solicitarla.
- Carencia de control de congestión: el UDP no tiene un método de control de congestión. Esto implica que canales de comunicación UDP con tráfico pesado de paquetes puedan saturar la red [15].

6.4. Librerías de Python preexistentes

6.4.1. GPIO Zero

Esta es una librería instalada por defecto en la imagen oficial de Raspbian. Esta librería esta diseñada para el uso de los pines GPIO de la Raspberry en conjunto con Python. Permite acceder a los registros de los pines y configurarlos para desempeñar diversas funciones, desde entradas/salidas simples, hasta el manejo de servomotores. A parte de esas funcionalidades básicas también permite el uso de protocolos de comunicación como por ejemplo SPI. Esta es una herramienta bastante útil que simplifica el desarrollo de aplicaciones aprovechando todas las funcionalidades de la RPi. La funcionalidad más importante es la interacción inmediata hacia los pines GPIO y sus distintas funcionalidades. Para hacer esto, se puede acceder a cada uno por medio de la numeración estándar, y la estructura de la librería se encarga de trazar dicho número hacia el registro puntual al que se debería de accesar dentro de la RPi. Accesar hacia las distintas funcionalidades que pueden tener los pines se facilita por medio del paradigma de la programación orientada a objetos. A continuación, se pueden apreciar ejemplos de estas funcionalidades [17].

En el siguiente ejemplo se pueden apreciar estas dos funcionalidades, por medio de *prompts* de Python. En este caso se trabaja con el pin con nominación estándar de 16.

En una primera instancia se desea utilizar como una salida PWM para interactuar con un *buzzer*. Para hacer esto se asigna a una variable la clase de *Buzzer*, para acceder a los métodos de conectividad de esta. En el ejemplo se usan dos sencillos métodos para encender y apagar dicho dispositivo. Luego de esto se desea usar el mismo pin para otro tipo de funcionalidad, en este caso una salida digital para un LED. Primero se debe de cerrar la relación del pin con la clase de *Buzzer*. Seguido a esto, el pin 16 ahora se relaciona con la clase de LED, y un método de esta misma permite establecer una alternancia de estados del pin para generar un parpadeo.

```
>>> from gpiozero import *
>>> bz = Buzzer(16)
>>> bz.on()
>>> bz.off()
>>> bz.close()
>>> led = LED(16)
>>> led.blink()
```

Esta librería tiene una gama amplia de funciones prediseñadas, a las cuales se pueden acceder por medio métodos de clases. Así como el ejemplo anterior demostró la funcionalidad para salidas de la RPi, también tiene funcionalidades para señales de entrada. En el código que se puede apreciar a continuación, se tiene un simple *scrpit* de como se pueden utilizar los métodos para pines de entrada para realizar un antirebote sencillo. Al asignar el pin 4 a la clase de *Button*, se puede accesar a una función que bloquea el código hasta que se presione dicho botón, detectando ya sea un flanco positivo o negativo. A pesar de esto al establecer el parámetro de *timeout*, si se supera esa cantidad de segundos el código se desbloquea y continúa hacia la siguiente línea.

```
from gpiozero import Button
button = Button(4)
button.wait_for_press(timeout=10)
print("Fin del script")
```

6.5. Sistema operativo

6.5.1. Raspbian OS

Este sistema operativo está basado en Debian, el cuál es una de las distribuciones principales de GNU/Linux. Raspbian es código abierto y se encuentra en constante desarrollo. Está distribución esta desarrollada específicamente para funcionar en los microprocesadores RPi. Esto permite que el *kernel* del sistema operativo sea capaz de accesar a sus funcionalidades de placa de desarrollo, como los pines GPIO. En cuanto a la interacción directa con el kernel, esta se puede entablar por medio del uso de la terminal de comandos como interfaz de comunicación. Los comandos que se pueden utilizar en esta son los mismos del entrono Linux. Adicional a esto, vale la pena mencionar que la herramienta de instalación, actualización y eliminación de software es *APT (Advanced Packaging Tool)*, herramienta

desarrollada por Debian. Esto es importante a tener en cuenta ya que dentro del entorno de Linux existen otros administradores de paquetes como *YUM (Yellow Dog Updater)*, utilizado en sistemas Red-Hat, que no son nativos dentro de Raspbian. A continuación, se puede encontrar una tabla con los comandos más utilizados en sistemas Linux [18] [19].

Comando	Operación	Opciones
pwd	Imprime el directorio actual.	n/a
ls	Imprime los elementos del directorio actual.	-l: Genera la impresión en forma de lista. -a: Muestra los archivos ocultos.
cd	Cambiar de directorio.	-P: No sigue links simbólicos.
cat	Despliega el contenido de un archivo en la terminal.	-n: Despliega los números de línea.
touch	Generación de archivos y actualización de timestamp.	-t: Crear un archivo con un tiempo específico.
cp	Copiar archivos.	-f: Forzar la copia.
mv	Mover archivos de localización.	-i: Garantizar permiso de sobrescribir archivos.
mkdir	Crear un directorio.	-p: Crea directorios padres en caso sea necesario.
rmdir	Elimina directorios vacíos.	-r: Permite borrar directorios con archivos.
locate	Ubica archivos dentro del sistema.	-i: No discrimina entre mayúsculas y minúsculas. -a: Solo muestra coincidencias específicas.
grep	Permite buscar adentro del texto de un archivo.	-c: Cantidad de líneas con coincidencias. -v: Líneas que no contengan coincidencias.
sudo	Eleva los permisos de la terminal a administrador.	-v: Refresca el límite de tiempo.
df	Despliega la cantidad de memoria libre en el sistema de archivos.	-h: Despliega la información en un formato legible
du	Despliega la cantidad de memoria ocupada.	-h: Despliega la información en un formato legible.
diff	Compara dos archivos e imprime las diferencias entre estos.	-q: Genera una salida cuando los archivos difieren. -s: Reporta cuando los archivos son idénticos.
tar	Permite archivar/comprimir/extraer archivos	-c: Se genera un documento archivado. -x: Se extrae el documento archivado. -t Se enlistan todos los archivos archivados.
chmod	Permite cambiar los permisos de un archivo/directorio	Por medio de una notación numérica se establecen los tipos de permisos deseados.
chown	Cambiar la propiedad de los archivos.	-R: Permite realizar la acción de manera recursiva.
ps	Enlista los procesos que están corriendo.	-a: Selecciona todos los procesos.
kill	Suspende un proceso.	-15: Guarda el proceso antes de cerrar el proceso. -9: Fuerza el cierre de manera inmediata.
ping	Enviar paquetes icmp hacia una IP/Dominio específico.	-c: Cantidad específica de paquetes a enviar. -i: Intervalo entre el envío de paquetes.
wget	Descargar archivos de internet.	-o: Indicar un archivo para logs.
uname	Imprimir información del sistema.	-a: Imprime una vista completa de todo el sistema.
apt install	Instalar paquetes por medio de la herramienta APT	-y: Conceder permisos de descarga.

Cuadro 1: Comandos de Linux

CAPÍTULO 7

Benchmarking de librerías y funciones de Python en distintos sistemas operativos

7.1. Mediciones realizadas

Para la realización de mediciones de rendimiento se estará calculando el consumo de memoria RAM de las funciones multiprocesamiento y multihilos tanto de Python como de lenguaje C. De esta manera se tendrá una referencia de como la naturaleza de cada lenguaje afecta su rendimiento. Como se ha referenciado anteriormente en este trabajo, existen múltiples diferencias entre un leguaje compilado (C) y uno interpretado (Python), por lo que al realizar estar pruebas, se busca poner dicha teoría en magnitud, para poder interpretar de manera tangible el efecto de la naturaleza del lenguaje en su rendimiento.

La estructura de las pruebas realizadas para ambos lenguajes consistía en dos *scripts* simples. El primero de ellos consistía en la creación de un hilo vacío, y el segundo en un proceso vacío. De esta manera se puede concentrar la medición del consumo en las funciones específicas a estudiar en este trabajo. A continuación, se pueden apreciar los códigos encargados a realizar las funciones para cada lenguaje. Seguido a estos, se encontrarán dos *scripts* de *bash* que sirven para ejecutar los códigos de manera iterativa y almacenar los datos de su rendimiento para el análisis.

```
1  '''
2   ================================================================================
3   Nombre: benchmarkmeasure.py
4   Autor:  Carlos Ricardo Cuellar Klingenberger
5   ================================================================================
6   '''
7  import timeit
8  import psutil
9  import threading
10 def imprimir():
```

```
    pass

def your_code1():
    global flag
    hilo = threading.Thread(target=imprimir)
    hilo.start()
    hilo.join()
    print(f'Memory {psutil.Process().memory_info().rss}')

execution_time = timeit.timeit(your_code1, number=1)
print(f"Time {execution_time}")
```

Listing 1: Benchmark Multihilo Python

```
'''
 ================================================================================
 Nombre: BenchForks.py
 Autor:  Carlos Ricardo Cuellar Klingenberger
 ================================================================================
 '''
import timeit
import psutil
import portable_forks
def imprimir():
    pass

def your_code1():
    PID=portable_forks.crear_proceso(imprimir)
    print(f'Memory {psutil.Process().memory_info().rss} seconds')

execution_time = timeit.timeit(your_code1, number=1)
print(f"Time {execution_time} KBs")
```

Listing 2: Benchmark Multiproceso Python

```
#include <stdio.h>
#include <pthread.h>
#include <sys/resource.h>
#include <time.h>

void* empty_thread(void* arg) {
    printf("Empty thread is running!\n");
    return NULL;
}

unsigned long get_memory_usage() {
    struct rusage r_usage;
    getrusage(RUSAGE_SELF, &r_usage);
    return r_usage.ru_maxrss;  // ru_maxrss is in kilobytes
}

int main() {
    pthread_t tid;
    clock_t start_time = clock();

    if (pthread_create(&tid, NULL, empty_thread, NULL) != 0) {
        fprintf(stderr, "Error creating thread.\n");
        return 1;
    }
```

```
25
26      pthread_join(tid, NULL);
27
28      clock_t end_time = clock();
29      double execution_time = ((double) (end_time - start_time)) /
    CLOCKS_PER_SEC;
30
31      printf("Thread has finished.\n");
32      printf("Execution Time: %f\n", execution_time);
33
34      unsigned long memory_usage = get_memory_usage();
35      printf("Memory Usage: %lu\n", memory_usage);
36
37      return 0;
38  }
```

Listing 3: Benchmark Multihilo C

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/wait.h>
5  #include <time.h>
6
7  int main() {
8
9      clock_t start_time = clock();
10     pid_t pid = fork();
11     if (pid == -1) {
12         fprintf(stderr, "Fork failed.\n");
13         return 1;
14     } else if (pid == 0) {
15         exit(0);
16     } else {
17         wait(NULL);
18         clock_t end_time = clock();
19         double execution_time = ((double) (end_time - start_time)) /
    CLOCKS_PER_SEC;
20         printf("Execution Time: %f seconds\n", execution_time);
21         unsigned long memory_usage = (unsigned long) getpid();
22         printf("Memory Usage: %lu KB\n", memory_usage);
23         return 0;
24     }
25 }
```

Listing 4: Benchmark Multiproceso C

```
1  ###
2   #
       ##============================================================
3   ###Nombre: run_testsP.sh
4   ###Autor:  Carlos Ricardo Cuellar Klingenberger
5   #
       ##============================================================
6   ###
7  #!/bin/bash
8
```

```
# Nombre del archivo csv
csv_file="python_results.csv"

# Encabezado CSV
echo "Memory Usage (KB),Time (seconds)" > "$csv_file"
sudo apt install python3-pip -y
pip3 install psutil
# Se corre el script deseado 30 veces
for ((i=1; i<=30; i++)); do
    # Se captura el output final del script en cada iteracion
    output=$(python3 BenchThread.py | tail -n 2)

    # Se accede a la memoria del standard output para ubicar donde se
    encuentran los datos
    time=$(echo "$output" | grep "Time" | awk '{print $2}')
    memory=$(echo "$output" | grep "Memory" |awk '{print $2}')


    # Se agregan como linea al csv
    echo "$memory,$time" >> "$csv_file"
done

echo "CSV file generated: $csv_file"
```

Listing 5: Bash para C

```
###
 #
    ##============================================================================

 ###Nombre: run_testsC.sh
 ###Autor:  Carlos Ricardo Cuellar Klingenberger
 #
    ##============================================================================

 ###
#!usr/bin/bash

# Nombre del archivo csv
csv_file="results.csv"

# Encabezado CSV
echo "Time (seconds),Memory (KB)" > $csv_file

# Se compila el codigo para generar el ejecutable de C
gcc -o BenchFork BenchFork.c -pthread

# Se corre 30 veces el programa de C
for ((i=1; i<=100; i++)); do
    # Se captura el retorno del programa
    output=$(./BenchFork)

    # Se extraen los valores impresos en el stdout
    time=$(echo "$output" | grep "Execution Time" | awk -F ": " '{print $2}')
    memory=$(echo "$output" | grep "Memory Usage" | awk -F ": " '{print $2}')

    # Se agregan al archivo csv
    echo "$time,$memory" >> $csv_file
done
```

```
30
31 echo "CSV file generated: $csv_file"
```

Listing 6: Bash para Python

Una vez medido el consumo de memoria y tiempo de ejecución de los *scripts* previamente mostrados, en los sistemas operativos Rocky Linux 9.0, Debian 12 y Ubuntu 18.04. Vale la pena remarcar, que los tres sistemas operativos se encontraban en igualdad de condiciones, siendo ejecutados en un ambiente virtual con 25GB de disco duro y 8GB de memoria RAM. Los hallazgos encontrados se reflejan en las siguientes gráficas y tablas:

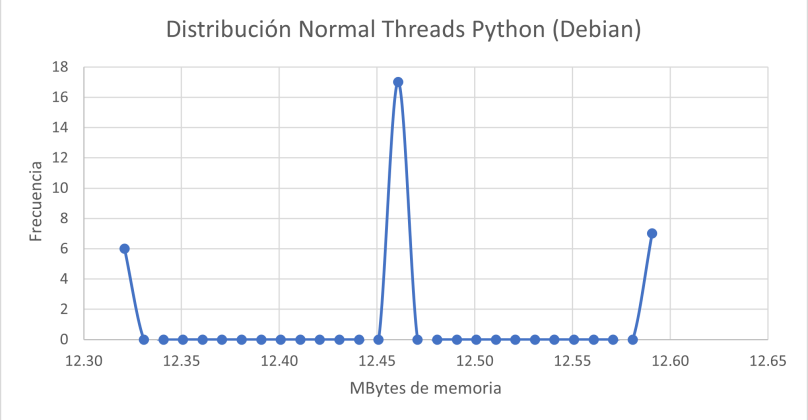


Figura 6: Distribución de consumo de memoria en hilos, Python ejecutado en Debian

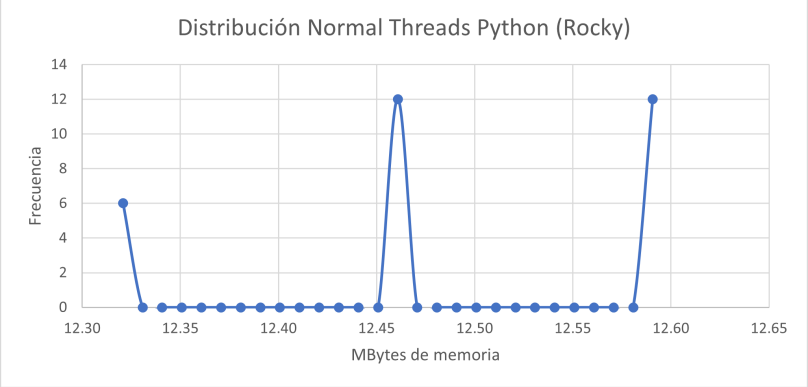


Figura 7: Distribución de consumo de memoria en hilos, Python ejecutado en Rocky

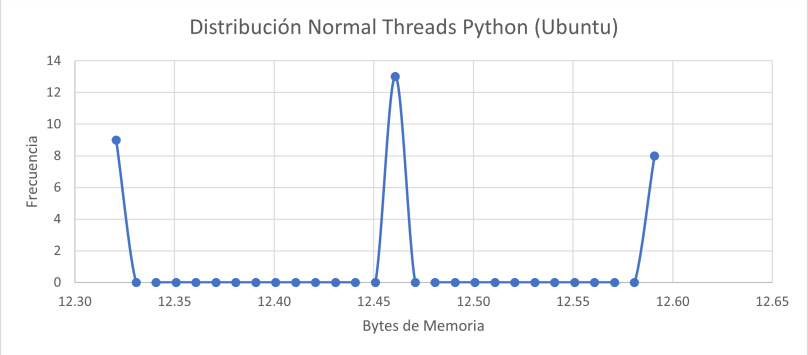


Figura 8: Distribución de consumo de memoria en hilos, Python ejecutado en Ubuntu

		Lenguaje		
Sistema Operativo	Variable	C	Python	Tasa de rendimiento C vs. Python
UBUNTU	Tiempo (s)	0.00013856	0.00052741	3.80635272
UBUNTU	Memoria (Byte)	1933025.28	12447470.9	6.43937307
ROCKY	Tiempo (s)	0.00014455	0.00045197	3.1267061
ROCKY	Memoria(Byte)	1933025.28	12478054.4	6.45519463
DEBIAN	Tiempo (s)	0.00014095	0.00046638	3.30881116
DEBIAN	Memoria(Byte)	2064097.28	12456209.1	6.03470059

Cuadro 2: Tasas de rendimiento de lenguajes

7.2. Análisis de mediciones realizadas

Con los resultados obtenidos acerca de las mediciones se pudo encontrar que el espectro de consumo de memoria para las funciones no variaba considerablemente a través de las series de iteraciones realizadas. El espectro se centraba dentro de tres valores, siendo los valores de los extremos los que menos frecuencia tenían regularmente. El comportamiento se observa para los tres sistemas operativos. Este es un hallazgo satisfactorio, ya que indica la fiabilidad de las funciones evaluadas, confirmando así el gran atributo de Python la portabilidad. A este hallazgo también lo dota de confiabilidad, el hecho de que la distribución de los resultados de rendimiento, reflejen una distribución normal. Finalmente en el cuadro comparativo de porcentajes de rendimiento, se puede apreciar como entre los tres sistemas operativos, los valores se mantenían discretamente cercanos unos a otros, siendo esto un síntoma del hallazgo descrito anteriormente. Sin embargo, lo importante del cuadro es hacer énfasis en como los porcentajes de rendimiento casi mantienen una tasa bastante similar entre el mejoramiento de C a Python a través de los tres sistemas. Lo destacable de esto, no es el hecho de que C supere a Python, sino que el interpretador de Python trabaja de una manera tan eficiente, que es capaz de localizar los recursos del *kernel* de una manera constante a través de sistemas operativos con distinta arquitectura. Vale la pena destacar como Debian es el sistema operativo evaluado donde Python tiene menor diferencia de rendimiento comparado con C.

CAPÍTULO 8

Resolución de laboratorios de Electrónica Digital 3, por medio de Python 3

8.1. Laboratorios a desarrollar

Se usarán los laboratorios de Electrónica Digital 3 como plataforma para evidenciar el uso exitoso del conjunto de librerías estudiadas. Con eso en mente, únicamente se elaborará sobre las secciones de los laboratorios que conlleven desarrollo de código donde se empleen herramientas referentes a la RPi y los temas de estudio de este trabajo. En esta lista de laboratorios del curso se encuentran:

- Laboratorio 3
- Laboratorio 5
- Laboratorio 6
- Laboratorio 7
- Laboratorio 8

Vale la pena mencionar que no se realizarán los laboratorios 1, 2 y 4 ya que estos no abarcan competencias relacionadas a la programación o a los temas de este trabajo.

8.2. Laboratorio 3

El laboratorio 3 consiste en la introducción a la programación multihilos y multiprocesos. En la guía original esto se consigue por medio de programas de ejemplo que utilizan funciones

bastante establecidas en C como lo es *fork()*. Sin embargo, para poder portarlo a Python se tomaron en cuenta bastantes condicionantes. La primera de ellas es la incompatibilidad del sistema operativo Windows con la creación de procesos por medio de *fork()*. Esto condujo al desarrollo de una librería que permitiera al estudiante experimentar con programación multiprocesos sin importar si este utiliza un sistema operativo Linux o Windows. Otra restricción que va de la mano con la anterior, es que la capacidad de generar multiprocesos en Windows requiere que se haga refrencia a una función específica, no como en C que un *fork* solo sigue ejecutando el código consecuente dentro de un contexto distinto. En esta práctica también se ponen a prueba conceptos como el contexto compartido e independiente y los *Program Identifiers* (PID).

8.2.1. Librería multiprocesos portable

Como se mencionó anteriormente, la utilización de *fork* como única función dentro de la estructura del curso destinada para el desarrollo de procesos hijos dentro del código, limita la portabilidad de los códigos a desarrollar. Esto debido a la falta de soporte de la herramienta mencionada en sistemas operativos de Windows. En consecuencia, se aprovechó el hecho de que la portabilidad es una de las más grandes ventajas de Python, haciendo muy fácil su compilación en una diversa gama de OS's, al igual que su acceso a una variada selección de librerías. Por lo tanto, se planteó el diseño de una librería que eliminara las restricciones de desarrollo de código multiproceso a la hora de programar en distintos OS's. Esto se logró por medio de la detección del sistema operativo local donde se esté ejecutando el programa. Una vez detectado, si corresponde a la familia Linux se crean los multiprocesos por medio de la función *fork()* de la librería os. En caso el sistema operativo local sea Windows, se utilizará el método *Process(targes, *args, **kwargs)* del módulo multiprocessing. Este permite la ejecución de procesos distintos sobrepasando las barreras que establece Windows en términos de multiprocesamiento. A continuación, se encuentra la documentación de la función *crear_proceso()* de la librería desarrollada portable_forks.

```
'''
 ================================================================================
 Nombre: Librería Portable Forks
 Autor:  Carlos Ricardo Cuellar Klingenberger
 ================================================================================
 '''
import multiprocessing
import os
import types
# Se define la función para crear procesos desde otro proceso
# Se contemplan 3 argumentos: uno que puede ser una función o nulo
# otro correspondiente a todos los argumentos posicionales que pueda tener la función a llamar
# otro para todos los argumentos de palabra clave de la función original
# los argumentos posicionales y de palabra clave NO son obligatorios, dependerá de la función a llamar
# el argumento de la función principal es obligatorio SOLO para OS's windows.
def crear_proceso(funcion=None, *args, **kwargs):
    # Se genera un error si el argumento función no es nulo o no es una función
    if funcion is not None and not isinstance(funcion, types.FunctionType):
        raise TypeError("Primer argumento debería ser una función llamable o nulo")
```

```
20      # Se crea un condicional para determinar si el OS's es windows o UNIX
21      if os.name == 'nt':
22          # Si el os es windows se levanta un error si el argumento función es
        nulo
23          if funcion is None:
24              raise TypeError("Para poder crear hilos dentro de Windows se debe
        hacer referencia a una función")
25          # De no serlo se genera un proceso usando la librería multiprocessing
26          else:
27              process = multiprocessing.Process(target=funcion, args=args,
        kwargs=kwargs)
28
29          process.start()
30          # Se retorna el PID del proceso generado
31          return process.pid
32      # En caso el OS sea unix se puede utilizar la función fork.
33      else:
34          # Se genera el subproceso
35          a = os.fork()
36          # Únicamente para el proceso hijo se manda a llamar la función a
        correr
37          if a == 0:
38              funcion(args, kwargs)
39              # Se retorna el pid del proceso hijo en el contexto del proceso
        principal
40              return a
41          # Rutina para no hacer nada por parte del proceso padre inmediatamente
         después de generar un hijo
42          else:
43              pass
```

Listing 7: Código del librería portable_forks

8.2.2. Desarrollo del laboratorio 3

El Laboratorio 3 constaba de tres partes. En la primera parte, se incursionaba dentro de la programación multihilos. Se desarrollaron dos programas distintos que debían de ser ejecutados desde la misma terminal. Seguido a esto, se desarrolló un tercer programa en donde se realizará el resultado anterior pero desde un mismo código utilizando hilos. Finalmente, se desarrolla un cuarto programa que utiliza los hilos de manera distinta al código anterior para generar un *output* similar. La segunda parte se adentraba en la programación multiprocesos y la diferencia de contexto entre hilos y procesos. En primer lugar, se desarrolló un programa que ejemplificara una ejecución multiproceso simple. Seguido a esto, se programaron dos programas adicionales donde se contrasta la diferencia del contexto a la hora de usar hilos y procesos. En la tercera y última parte se realizaron códigos para profundizar en lo abordado previamente. Se hizo un código que manejará cuatro hilos en simúltaneo y otro que realiza una generación de subprocesos en cadena.

Parte 1

La traducción de los primeros cuatro archivos no supuso ninguna complicación, ya que se dominaban conceptos y funciones equivalentes en Python a las que se usaban en la guía actual de C. Vale la pena resaltar la brevedad de los códigos y destaca la fácil manipulación de variables de todo tipo dentro del lenguaje de Python. Los *Strings* se representan de manera concisa sin necesidad de armar arreglos de caracteres, como en muchas situaciones lo amerita el lenguaje de C. A continuación, se tienen los códigos debidamente documentados para la realización de la primera parte del laboratorio 3.

```
'''
 ================================================================================
 Nombre: L3_Hello.py
 Autor:  Carlos Ricardo Cuellar Klingenberger
 ================================================================================
 '''
#Se importa la librería para los delays
import time
#Ciclo infinito
while True:
  #Se imprime Hola en la terminal
  print('Hola')
  #Delay de 1.1 segundos
  time.sleep(1.1)
  pass
```

Listing 8: Código del archivo L3_Hello.py

```
'''
 ================================================================================
 Nombre: L3_World.py
 Autor:  Carlos Ricardo Cuellar Klingenberger
 ======
 '''
#Se importa la librería para los delays
import time
#Ciclo infinito
while True:
  #Se imprime Mundo en la terminal
  print('Mundo')
  #Delay de 1 segundo
  time.sleep(1)
  pass
```

Listing 9: Código del archivo L3_World.py

```
'''
 ================================================================================
 Nombre: L3_Hilos_Ej1.py
 Autor:  Carlos Ricardo Cuellar Klingenberger
 ================================================================================
 '''
import threading
import time
#Función a ejecutar por el segundo hilo
def imprimir(mnsj):
    print(mnsj)
```

```
    time.sleep(1)
    pass
#Argumetno que se pondrá de entrada para el hilo
mensaje='Soy el segundo hilo'
#Creación del hilo. En targer se indica la función que este correra
#y en args los argumetnos de entrada de la función. Se pone una coma adicional
#ya que se espera una tupla.
hilo=threading.Thread(target=imprimir,args=(mensaje,))
#Se inicia el hilo secundario
hilo.start()
#Se trabaja desde el hilo principal
print('Soy el hilo principal')
#Se espera a que termine el hilo secundario
hilo.join()
print('Ya acabaron los dos hilos')
```

Listing 10: Código del archivo L3_Hilos_Ej1.py

```
'''
 ================================================================================
 Nombre: L3_Hilos_Ej2.py
 Autor:  Carlos Ricardo Cuellar Klingenberger
 ================================================================================
 '''
import threading
import time
#Se declara una funcion que sera usada por el hilo posteriormente. Esta
    funcion requiere un argumento posicional.
def My_Thread(mensaje):
  #Se crea un ciclo infinito
  while True:
    #Se imprime el argumento posicional
    print(mensaje)
    #Se espera 1.1 seg antes de repetir el ciclo
    time.sleep(1.1)
    pass
#Se crea el objeto del hilo identificando al funcion y argumentos posicionales
    de esta
thread2=threading.Thread(target=My_Thread, args=('Hello ',))
#Se inicializa el hilo secundario
thread2.start()
#Codigo a correr sobre el segundo hilo
#Se crea un ciclo infinito
while True:
  #Se imprime un string constante
  print('World')
  #Se espera un segundo antes de repetir el ciclo
  time.sleep(1)
  pass
```

Listing 11: Código del archivo L3_Hilos_Ej2.py

Parte 2

La segunda parte de este laboratorio incursiona por primera vez con la programación multiproceso, y a partir de esta se desarrolla la primera función de la librería orientada

a portabilidad para programación multiproceso desarrollada. Dicha función fue expuesta al principio del capítulo, y nace a partir de las necesidades detectadas en esta parte del laboratorio 3. El desarrollo de los scripts en sí, no presentó ningún hallazgo destacable, a diferencia de que sí lo hizo el desarrollo de la librería. Durante ese proceso, se destaca el hecho de la incapacidad de Windows de generar subprocesos en *ramas*, ya que solo se permite hacerlo por medio de un proceso de propósito dedicado a una función previamente definida. En los siguientes extractos se pueden encontrar ejemplos de aplicación de la función desarrollada, dentro de los códigos del laboratorio.

```
'''
 ================================================================================
 Nombre: L3_Hilos_Ej1.py
 Autor:  Carlos Ricardo Cuellar Klingenberger
 ================================================================================
 '''
import portable_forks
import time
'''
Se definen las funciones para los procesos hijos que se crearan. Si se está utilizando
windows como OS host, es obligatorion redirigir los procesos hijos hacia una funcion objetivo.
En caso se tenga un OS unix, se puede crear subprocesos sin necesidad de hacer referencia a otra
funcion.
'''
def child(numero, nombre,palabra='Hello'):
    while True:
        print(f'{palabra} {nombre} y {numero}')
        time.sleep(1.1)
        pass
def child2():
    while True:
        print(f'!!!!!!!!!!')
        time.sleep(1.2)
        pass


#Este condicional solo se ejecuta cuando el código se corre desde un proceso padre.

if __name__=="__main__":
    #Se crean 2 subprocesos con funciones distintas. Una función sin argumentos
    #y otra función con argumentos posicionales y de palabras clave.
    #La funcion crear_proceso del módulo portable forks retorna el PID del proceso generado
    PID=portable_forks.crear_proceso(child,49,'Carlos',palabra='PSHHHH')
    PID2=portable_forks.crear_proceso(child2)
    print(f'El PID de los hijos es: {PID} y {PID2}')

#Este es el código del proceso principal
    while True:
        print("World")
        time.sleep(1)
```

Listing 12: Código del archivo L3_fork_Ej1.py

```
'''
 ================================================================================
 Nombre: L3_thread_contexto.py
 Autor:  Carlos Ricardo Cuellar Klingenberger
 ================================================================================
 '''
import threading
import time
#Se define la variable contador dentro del contexto del primer hilo
contador=0
#Se define la funcion del hilo
def my_thread():
  #Se indica el uso de contador como variable global, para no crear una
    localidad de memoria aparte para el neuvo valor
  #Sino que hacer referencia al original
  global contador
  #Se imprime el cuenta iniclal
  print(f'Cuenta inicial: {contador}')
  #Se establece el valor base como 200
  contador=200
  #Se incrementa y se imprime indfefinidamente el contador
  while True:
    contador+=1
    print(f'My htread: {contador}')
    time.sleep(1.1)
  pass
#Se crea el objeto del hilo con la funcion dearrollada como objetivo
thread2=threading.Thread(target=my_thread)
#Se inicializa el hilo
thread2.start()
#Se establece un valor base de contador
contador=100
print(f'Cuenta Hilo Padre: {contador}')
#Se incrementa y se imprime indfefinidamente el contador
while True:
  contador+=1
  print(f'Hilo Padre: {contador}')
  time.sleep(1)
  pass
```

Listing 13: Código del archivo L3_thread_contexto.py

```
'''
 ================================================================================
 Nombre: L3_fork_contexto.py
 Autor:  Carlos Ricardo Cuellar Klingenberger
 ================================================================================
 '''
import portable_forks
import time
#Se define la variable contador dentro del contxto general
contador=0
#Se define la funcion del proceso hijo
def my_proc():
  #Se indica el uso de contador como variable global, pero a pesar de esto por
     la naturaleza
  #de los procesos no se compartirá el contexto con el proceso padre,
    difirnedo así en el valor.
  global contador
```

```
  #Se imprime el cuenta iniclal
  print(f'Cuenta inicial: {contador}')
  #Se establece el valor base como 200
  contador=200
  #Se incrementa y se imprime indfefinidamente el contador
  while True:
    contador+=1
    print(f'My Proc: {contador}')
    time.sleep(1.1)
  pass
#Codigo a ejecutar por el programa principal
if __name__=="__main__":
  #Se crea el objeto del proceso con la funcion dearrollada como objetivo
  proc2=portable_forks.crear_proceso(my_proc)
  #Se obtiene el pidf del hijo
  print(f'El valor del PID del hijo es {proc2}')
  #Se establece un valor base de contador
  contador=100
  print(f'Cuenta Proc Padre: {contador}')
  #Se incrementa y se imprime indfefinidamente el contador
  while True:
    contador+=1
    print(f'Proc Padre: {contador}')
    time.sleep(1)
    pass
```

Listing 14: Código del archivo L3_fork_contexto.py

Parte 3

En cuanto a nuevos hallazgos el desarrollo de esta última parte del laboratorio fue intrascendente. Lo más destacado es cómo la función para crear procesos desarrollada (cuando es usada en Windows), no puede establecer un árbol genealógico de procesos hijos como lo hace *fork()* de manera exponencial. Esta lo hace de manera lineal y por la necesidad de declarar una función previa para la ejecución del subproceso, resulta complicado ascender en una gran cantidad de generaciones a la hora de crear subprocesos hijos.

```
'''
 ================================================================
 Nombre: L3_Hilos_Ej2_MOD.py
 Autor:  Carlos Ricardo Cuellar Klingenberger
 ================================================================
 '''
import threading
import time
#Se declara una funcion que sera usada por el hilo posteriormente. Esta
    funcion requiere un argumento posicional.
def My_Thread(mensaje):
  #Se crea un ciclo infinito
  while True:
    #Se imprime el argumento posicional
    print(mensaje)
    #Se espera 1.1 seg antes de repetir el ciclo
    time.sleep(1.1)
    pass
```

```
#Se crea una funcion para el cuarto hilo con argumentos posicionales y de palabtas clave
def cuarto_hilo(producto, cantidad=1, precio=10):
  #Se declara ciclo infinito que va a imprimir una combinacion de los argmentos
  while True:
    print(f'Se compraron {cantidad} {producto}(s) PRECIO: {cantidad*precio}')
    time.sleep(1.5)
  pass
#Se crea el objeto del hilo identificando al funcion y argumentos de esta
thread2=threading.Thread(target=My_Thread, args=('Hello ',))
thread3=threading.Thread(target=My_Thread, args=('TESIS CUELLAR',))
thread4=threading.Thread(target=cuarto_hilo, args=('Manzana',),kwargs={'cantidad':2, 'precio':5})

#Se inicializa el hilo
thread2.start()
thread3.start()
thread4.start()
#Codigo a correr sobre el main hilo
#Se crea un ciclo infinito
while True:
  #Se imprime un string constante
  print('World')
  #Se espera un segundo antes de repetir el ciclo
  time.sleep(1)
  pass
```

Listing 15: Código del archivo L3_Hilos_Ej2_MOD.py

```
'''
 ================================================================================
 Nombre: L3_varios_forks.py
 Autor:  Carlos Ricardo Cuellar Klingenberger
 ================================================================================
 '''
import portable_forks
import time
#Se crea una funcion para re dirigir el proceso del proceso
def funex2():
    time.sleep(0.1)
    pass
#Funcion para los procesos hijos de primera generacion
def funex():
    ident=portable_forks.crear_proceso(funex2)
    print(f'Valor PID: {ident}')
    pass
#Codigo para el proceso principal
if __name__=="__main__":
    #Creacion de procesos de primera generacion
    PID=portable_forks.crear_proceso(funex)
    print(f'Valor PID: {PID}')
    PID=portable_forks.crear_proceso(funex)
    print(f'Valor PID: {PID}')
    PID=portable_forks.crear_proceso(funex)
    print(f'Valor PID: {PID}')
    time.sleep(10)
```

Listing 16: Código del archivo L3_varios_forks.py

8.3. Laboratorio 5

El laboratorio 5 es el primero en donde se interactúa desde el lenguaje de programación con los pines GPIO de la RPi. Como se describe en la sección de marco teórico, la librería seleccionada para lograr esta interacción con el mundo físico es la de GPIO Zero. En el desarrollo de esta práctica se pone a prueba su estructura de programación orientada a objetos, empleando sus objetos y métodos específicos para trabajar los pines de manera especializada. Tal es el caso de la clase *LED*, la cuál permite realizar el *toggle* del pin, por medio de solo un método, o la clase *Button*, proveyendo funciones *blocking* para facilitar la lectura de los pulsos. Añadido a esto, se implementó la programación multihilos en uno de los dos programas que se desarrollaron, implementando así una de las competencias fundamentales cubiertas en las traducciones hacia Python.

8.3.1. Desarrollo del laboratorio 5

En este laboratorio, únicamente la tercera parte solicita el desarrollo de programas. En esta se pide la escritura de dos códigos. El primer de estos indica que se debe de encender y apagar dos LEDs, cada cierto tiempo aleatorio entre 0.5s y 1.5s, de manera intermitente. El segundo programa solicita que se produzca un sonido con una bocina. Dicho sonido se debe de inicializar con el pulso de un botón y el sistema debe de ser capaz de pausarse, reanudarse y apagarse por medio de entradas ingresadas por el teclado.

Parte 3

El desarrollo del primer código es bastante discreto. A la hora de hacer la traducción, esta puede servir como una manera para familiarizarse con la dinámica de funcionamiento de la librería de GPIO Zero y sus respectivos objetos y métodos. Aparte de eso la estrucutra del código es bastante sencilla y se puede entender fácilmente con el apoyo de la documentación provista a continuación. En cuanto al segundo programa, se buscó hacer un diseño eficiente de lo solicitado por medio de la integración de programación multihilos. En esta oportunidad, el hilo principal es el encargado de monitorear las entradas del sistema, tanto el botón como el *input* del teclado, mientras que el hilo secundario es el encargado de controlar el buzzer por medio de variables globales, compartidas en el contexto de los hilos. Se adjunta el código a continuación.

```
'''
 ================================================================================
 Nombre: L5_LED.py
 Autor:  Carlos Ricardo Cuellar Klingenberger
 ================================================================================
 '''
from gpiozero import LED
import time
import random
#Se importa la clase LED del módulo GPIO Zero y se asigna a dos objetos
led1=LED(17)
led2=LED(18)
#Ciclo infinito para repetir la intermitencia de los LEDs
```

```
while True:
  #Se genera un número random en el intervalo solicitado
  dly=random.uniform(0.5,1.5)
  #Se enciende solo un Led con el delay prestablecido
  led1.on()
  led2.off()
  time.sleep(dly)
  #Se invierten los estados anteriores por la duración del delay
  led1.off()
  led2.on()
  time.sleep(dly)
  pass
```

Listing 17: Código del archivo L5_LED.py

```
'''
 ================================================================================
 Nombre: L5_BUZZER.py
 Autor:  Carlos Ricardo Cuellar Klingenberger
 ================================================================================
 '''
from gpiozero import Button
from gpiozero import LED
import time
import threading

#Se crean los objetos para el boton y buzzer
bot=Button(17)
buzzer=LED(18)
#Declaracion e inicializacion de variables para controlar el estado del buzzer
global play,pausa
play=False
pausa=False
#Se define la función para general del PWM
def PWM(freq);
  #Variables globales para manejar el estado del buzzer
  global play, pausa
  while play:
    #Rutina de reproduccion de nota
    if not pausa:
      #Se invierte el estado de la salida digital
      buzzer.toggle()
      #Se pasa de frecuencia a período, Duty cicle de 50%
      time.sleep(0.5/freq)
    else:
      buzzer.off()
while True:
  #Rutina para encender el sistema por medio del boton
  if not play:
    bot.wait_for_press()
    #Actualización de estado del sistema
    play=True
    pausa=False
    #Frecuencia de A5
    freq=440
    #Se reproduce el sonido en un hilo
    hilo=threading.Thread(target=PWM, args=(freq,))
    hilo.start()
  #Procesamiento de comandos para pausa, reanudar y salir
```

```
  else:
    entrada=input()
    if entrada=='p':
      pausa=True
    elif entrada=='r':
      pausa=False
    elif entrada=='s':
      #Apagado del sistema
      play=False
      buzzer.off()
      hilo.join()
      break
  pass
```

Listing 18: Código del archivo L5_BUZZER.py

8.4. Laboratorio 6

Este laboratorio originalmente se concentra en la manipulación de cadenas de caracteres. Sin embargo, debido a que en Python existe el tipo de variable *string* esta labor se simplifica bastante. La actividad consiste en unir de manera alternante el texto contenido en dos archivos en un *buffer* común. La lectura de ambos archivos se realizará de manera alternante por medio de sincronización de hilos. Finalmente, cuando ambos hilos hayan terminado de leer sus respectivos archivos, el buffer común se enviará a un único archivo donde se tenga el texto combinado de manera consolidada.

8.4.1. Desarrollo del laboratorio

Para poder realizar de manera satisfactoria esta práctica se necesitará emplear la librería *threading* y *queue* para poder realizar la sincronización de hilos por medio de *timers*. De primero se debe de declarar un *buffer* común que será compartido en ambos hilos. Luego de esto se definen las funciones para la reconstrucción de los archivos y las lecturas individuales. Finalmente se estructuran los temporizadores para posteriormente definir y ejecutar los respectivos hilos.

```
import threading
import time
from queue import Queue

# Se crea un buffer compartido para almacenar información entre hilos de
    manera eficiente y segura
buffer = Queue()

# Se define una función para leer las líneas de un archivo
def leer(archivo):
    with open(archivo, 'r') as arch_abierto:
        for linea in arch_abierto:
            buffer.put(linea.strip())
            print('Hola')

# Función para reconstruir el archivo
```

```
def reconstruir():
    texto_rec = ""
    while True:
        if not buffer.empty():
            linea = buffer.get()
            if linea == "EOF":
                break
            texto_rec += linea + "\n"
    with open("Lab6_reconstruido.txt", "w") as archivo:
        archivo.write(texto_rec)
    print(texto_rec)

archivo1 = "Lab6_primero.txt"
archivo2 = "Lab6_segundo.txt"
hilo1 = threading.Thread(target=leer, args=(archivo1,))
hilo2 = threading.Thread(target=leer, args=(archivo2,))
hilo3 = threading.Thread(target=reconstruir)

timer1 = threading.Timer(0.5, hilo1.start)
timer2 = threading.Timer(0.7, hilo2.start)
timer3 = threading.Timer(1.1, hilo3.start)

print('start timers')
timer1.start()
timer2.start()
timer3.start()
print('join timers')
timer1.join()
timer2.join()
timer3.join()
print('join hilos')


buffer.put("EOF")
```

Listing 19: Código del archivo Lab6.py

8.5. Laboratorio 7

Esta práctica busca poner en prueba distintos tipos de escalonamiento. El primero de ellos por poleo calendarizado y el segundo sería por medio de *semaphores*. La puesta a prueba de estos métodos se haría por medio de la construcción de un semáforo usando la RPi y sus pines GPIO. Se deben de encender de manera alternante, dos LED's, pero cuando un botón se presiona, se enciende una tercera LED, simulando el paso peatonal. Vale la pena mencionar que las luces sos mutuamente excluyentes, es decir nunca puede estar encendida más de una al mismo tiempo. Las tareas del funcionamiento de este sistema vial, se deberán de separar por bloques para lograr el escalonamiento deseado. El código entre cada una de las partes se puede re usar, ya que lo que cambia es únicamente las secciones de escalonamiento.

8.5.1. Desarrollo del laboratorio

Para el desarrollo de este laboratorio no es necesario usar alguna librería específica para la aplicación de algoritmos de escalonamiento. Con los hilos del módulo *threading* es más que suficiente para poder ordenar tareas por poleo escalonado, y esta misma librería contiene un clases de *semaphores*, para aplicar de manera eficaz. Con los conocimientos previos del uso de GPIOZero, este práctica no supondría un gran contenido nuevo en términos de funciones de programación. Esta práctica esta poniendo a prueba la lógica de estructuración de código.

```
'''
 ================================================================
 Nombre: Lab7_1.py
 Autor:  Carlos Ricardo Cuellar Klingenberger
 ================================================================
 '''
from gpiozero import LED, Button
from time import sleep
import threading

# Definir los pines GPIO para los semáforos y el botón de peatones
semáforo_vehicular_1 = LED(23)
semáforo_vehicular_2 = LED(27)
semáforo_peatonal = LED(22)
botón_peatonal = Button(16, pull_up=True)

# Variable para seguir el estado de la solicitud peatonal
peatón_solicitado = False

# Función para controlar los semáforos
def controlar_semáforos():
    global peatón_solicitado  # Declarar peatón_solicitado como una variable global
    while True:
        # Permitir que el tráfico vaya en una dirección (Semáforo 1)
        if botón_peatonal.is_pressed:
            # Solicitud peatonal
            peatón_solicitado = True
        semáforo_vehicular_1.on()
        semáforo_vehicular_2.off()
        semáforo_peatonal.off()
        sleep(1)
        if botón_peatonal.is_pressed:
            # Solicitud peatonal
            peatón_solicitado = True

        # Permitir que el tráfico vaya en la otra dirección (Semáforo 2)
        semáforo_vehicular_1.off()
        semáforo_vehicular_2.on()
        semáforo_peatonal.off()
        sleep(1)

        # Verificar si se presionó el botón de peatones
        if botón_peatonal.is_pressed:
            # Solicitud peatonal
            peatón_solicitado = True

        if peatón_solicitado:
```

```
            # Permitir que los peatones crucen
            semáforo_vehicular_1.off()
            semáforo_vehicular_2.off()
            semáforo_peatonal.on()
            sleep(1)  # El semáforo peatonal permanece encendido durante 5 segundos
            semáforo_peatonal.off()
            peatón_solicitado = False  # Reiniciar la solicitud peatonal
            sleep(.5)  # Asegurarse de que el semáforo peatonal esté apagado antes de que se reanude el tráfico

# Crear un hilo para controlar los semáforos
hilo_semáforos = threading.Thread(target=controlar_semáforos)

# Iniciar el hilo de los semáforos
hilo_semáforos.start()

try:
    # Mantener el hilo principal en ejecución
    while True:
        pass
except KeyboardInterrupt:
    # Limpiar GPIO en Ctrl+C
    hilo_semáforos.join()
    semáforo_vehicular_1.off()
    semáforo_vehicular_2.off()
    semáforo_peatonal.off()
```

Listing 20: Código de poleo por escalonamiento Lab7

```
'''
 ================================================================================
 Nombre: Lab7_2.py
 Autor:  Carlos Ricardo Cuellar Klingenberger
 ================================================================================
 '''
from gpiozero import LED, Button
from time import sleep
import threading

# Definir los pines GPIO para los semáforos y el botón de peatones
semáforo_vehicular_1 = LED(23)
semáforo_vehicular_2 = LED(27)
semáforo_peatonal = LED(22)
botón_peatonal = Button(16, pull_up=True)

# Variable para seguir el estado de la solicitud peatonal
peatón_solicitado = False
semaphore_hilo1 = threading.Semaphore(1)
semaphore_hilo2 = threading.Semaphore(0)
semaphore_hilo3 = threading.Semaphore(0)

# Función para controlar los semáforos
def semaforo_1():
    global peatón_solicitado
    while True:
        semaphore_hilo1.acquire()
        if botón_peatonal.is_pressed: # Solicitud peatonal
            peatón_solicitado = True
```

```
        semáforo_vehicular_1.on()
        semáforo_vehicular_2.off()
        semáforo_peatonal.off()
        sleep(1)
        semaphore_hilo2.release()#La ultima linea de cada fun se encarga del escalonamiento por semaphores
def semaforo_2():
    global peatón_solicitado
    while True:
        semaphore_hilo2.acquire()
        if botón_peatonal.is_pressed: # Solicitud peatonal
            peatón_solicitado = True
        semáforo_vehicular_1.off()
        semáforo_vehicular_2.on()
        semáforo_peatonal.off()
        sleep(1)
        semaphore_hilo3.release()
def semaforo_3():
    global peatón_solicitado
    while True:
        semaphore_hilo3.acquire()
        if botón_peatonal.is_pressed: # Solicitud peatonal
            peatón_solicitado = True
        if peatón_solicitado:
            semáforo_vehicular_1.off()
            semáforo_vehicular_2.off()
            semáforo_peatonal.on()
            peatón_solicitado=False
            sleep(1)
        semaphore_hilo1.release()
# Crear un hilo para controlar los semáforos
hilo_semáforo_1 = threading.Thread(target=semaforo_1)
hilo_semáforo_2 = threading.Thread(target=semaforo_2)
hilo_semáforo_3 = threading.Thread(target=semaforo_3)



try:
    # Mantener el hilo principal en ejecución
    while True:
        hilo_semáforo_1.start()
        hilo_semáforo_2.start()
        hilo_semáforo_3.start()
        hilo_semáforo_1.join()
        hilo_semáforo_2.join()
        hilo_semáforo_3.join()
        pass
except KeyboardInterrupt:
    # Limpiar GPIO en Ctrl+C
    semáforo_vehicular_1.off()
    semáforo_vehicular_2.off()
    semáforo_peatonal.off()
```

Listing 21: Código de poleo por *semaphores* Lab7

8.6. Laboratorio 8

Para este laboratorio se debieron de crear dos códigos. El primero es un cliente (que sirve como material didáctico de la guía) que por medio de *sockets* envía mensajes de *broadcast* a todos los dispositivos de la red. Dichos mensajes son "QUIÉN ES", "VOTE". El estudiante es el encargado de programar un servidor capaz de responder a esos mensajes y disputar la contienda de votación de manera efectiva, obteniendo el mismo resultado que los demás servidores que se vayan a conectar. Al comando "QUIÉN ES"únicamente responderá el servidor, considerado como maestro, al cliente que envía el mensaje. Al comando "VOTE"todos los servidores conectados deben de emitir un broadcast emitiendo su voto, y cada servidor será encargado de decidir si el mismo es el maestro en función de la comparación de su propios voto en contraste a los votos recibidos.

8.6.1. Desarrollo del laboratorio

Al estar realizando la migración completa de los laboratorios se desarrolló tanto el servidor como el cliente. Como premisa básica ambos tienen que usar *sockets*, pero lo realmente retador es alternar los *sockets* por medio de hilos, para tener de manera escalonada, procesos de envío de mensajes así como de constante recepción. El código del cliente es relativamente sencillo ya que solo tiene dos labores: enviar mensajes de *broadcast*, y recibir mensajes apuntados directamente al mismo notificando de la existencia del maestro. Por parte del servidor, se debe de crear estrucutras de decodificación de los mensajes, para convertir el paquete de tráfico en un string, y posteriormente aplicarle a dicho string el debido procesamiento para evaluar su contenido.

```
'''
================================================================================
Nombre: server.py
Autor: Carlos Ricardo Cuellar Klingenberger
================================================================================
'''
import socket
import threading
import random
# Definir constantes
'''TCP_IP = "0.0.0.0"  # Escuchar en todas las interfaces disponibles
TCP_PORT = 12345'''
BROADCAST_PORT = 12346
voto = 15
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.connect(("8.8.8.8", 80)) #Se usa una direción arbitraria para identificar la
    IP de salida a internet
IP = s.getsockname()[0]  #Se obtiene la IP
IP_vote = IP.split(".")[3] #Se le da formato a la IP
s.close()
last_big_vote = 0
MASTER = False
def get_ip():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.settimeout(0)
    try:
        # ni siquiera tiene que ser alcanzable
```

```
        s.connect(('10.254.254.254', 1))
        IP = s.getsockname()[0]
    except Exception:
        IP = '127.0.0.1'
    finally:
        s.close()
    return IP
def broadcast_message(message, target_ip):
    # Crear un socket UDP para la transmisión
    broadcast_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    broadcast_socket.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)

    try:
        # Transmitir el mensaje a la IP de destino
        broadcast_socket.sendto(bytes(message, 'utf-8'), (target_ip,
    BROADCAST_PORT))
    finally:
        broadcast_socket.close()
def listen_for_messages():
    global last_big_vote, voto, IP, IP_vote, MASTER

    # Crear un socket UDP para recibir transmisiones
    receive_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    receive_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    receive_socket.bind(("0.0.0.0", BROADCAST_PORT))

    print(f"El servidor está escuchando transmisiones en el puerto {
    BROADCAST_PORT}")

    try:
        while True:
            # Recibir el mensaje transmitido
            data, addr = receive_socket.recvfrom(1024)
            received_message = data.decode('utf-8')

            # Imprimir el mensaje recibido
            print(f"Mensaje de transmisión recibido de {addr}: {
    received_message}")
            if received_message == "QUIEN ES":
                if MASTER:
                    broadcast_message("Alonso es el maestro", addr[0])

            # Verificar si el mensaje recibido es "VOTE"
            if received_message == "VOTE":
                last_big_vote = 0
                voto = random.randint(1, 15)
                print(f'VOTO ALONSO: {voto}')
                voto_message = f"#{IP} {voto}"
                broadcast_message(voto_message, "255.255.255.255")

            if received_message.startswith("#"):
                voto_ajeno = int(received_message.split(" ")[1])
                ip_ajena = received_message.split(" ")[0].split(".")[3]

                if voto_ajeno > last_big_vote:
                    last_big_vote = voto_ajeno
                if voto > last_big_vote:
                    MASTER = True
                if IP_vote <= ip_ajena:
```

```
                    if last_big_vote == voto:
                        MASTER = True
                if voto < voto_ajeno:
                    MASTER = False
    finally:
        receive_socket.close()
if __name__ == "__main__":
    # Iniciar un hilo para escuchar transmisiones
    listen_thread = threading.Thread(target=listen_for_messages)
    listen_thread.start()
    # Esperar a que el hilo finalice
    listen_thread.join()
```

Listing 22: Código de servidor lab 8

```
'''
 ================================================================================
 Nombre: client.py
 Autor:  Carlos Ricardo Cuellar Klingenberger
 ================================================================================
 '''
 import socket
import threading
import random

# Definir constantes
TCP_IP = "0.0.0.0"  # Escuchar en todas las interfaces disponibles
TCP_PORT = 12345
BROADCAST_PORT = 12346
voto = 15
IP = socket.gethostbyname(socket.gethostname())
IP_vote = IP.split(".")[3]
last_big_vote = 0
MASTER = False

def get_ip():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.settimeout(0)
    try:
        # ni siquiera tiene que ser alcanzable
        s.connect(('10.254.254.254', 1))
        IP = s.getsockname()[0]
    except Exception:
        IP = '127.0.0.1'
    finally:
        s.close()
    return IP

def broadcast_message(message, target_ip):
    # Crear un socket UDP para la transmisión
    broadcast_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    broadcast_socket.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)

    try:
        # Transmitir el mensaje a la IP de destino
        broadcast_socket.sendto(bytes(message, 'utf-8'), (target_ip,
    BROADCAST_PORT))
    finally:
        broadcast_socket.close()
```

```
44
45 def listen_for_messages():
46     global last_big_vote, voto, IP, IP_vote, MASTER
47
48     # Crear un socket UDP para recibir transmisiones
49     receive_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
50     receive_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
51     receive_socket.bind(("0.0.0.0", BROADCAST_PORT))
52
53     print(f"El servidor está escuchando transmisiones en el puerto {
       BROADCAST_PORT}")
54
55     try:
56         while True:
57             # Recibir el mensaje transmitido
58             data, addr = receive_socket.recvfrom(1024)
59             received_message = data.decode('utf-8')
60
61             # Imprimir el mensaje recibido
62             print(f"Mensaje de transmisión recibido de {addr}: {
       received_message}")
63             if received_message == "QUIEN ES":
64                 if MASTER:
65                     broadcast_message("Alonso es el maestro", addr[0])
66
67             # Verificar si el mensaje recibido es "VOTE"
68             if received_message == "VOTE":
69                 last_big_vote = 0
70                 voto = random.randint(1, 15)
71                 print(f'VOTO ALONSO: {voto}')
72                 voto_message = f"#{IP} {voto}"
73                 broadcast_message(voto_message, "255.255.255.255")
74
75             if received_message.startswith("#"):
76                 voto_ajeno = int(received_message.split(" ")[1])
77                 ip_ajena = received_message.split(" ")[0].split(".")[3]
78
79                 if voto_ajeno > last_big_vote:
80                     last_big_vote = voto_ajeno
81                 if voto > last_big_vote:
82                     MASTER = True
83                 if IP_vote <= ip_ajena:
84                     if last_big_vote == voto:
85                         MASTER = True
86                 if voto < voto_ajeno:
87                     MASTER = False
88
89     finally:
90         receive_socket.close()
91
92 if __name__ == "__main__":
93     # Iniciar un hilo para escuchar transmisiones
94     listen_thread = threading.Thread(target=listen_for_messages)
95     listen_thread.start()
96
97     # Esperar a que el hilo finalice
98     listen_thread.join()
```

Listing 23: Código de cliente lab 8

CAPÍTULO 9

Desarrollo de herramienta programadora e integradora de bases de datos y sistemas embebidos

La herramienta busca automatizar la configuración de funciones de sistemas embebidos y enlazar estos de manera inmediata a una base de datos. En dicha base, se podrá llevar un registro histórico de los valores de las entradas y salidas digitales, al igual que entradas digitales. Para centralizar la aplicación sobre sistemas embebidos, se enfocará la herramienta a la automatización con los microcontroladores de Arduino, específicamente el Arduino Uno y el Arduino Nano. Todo esto se hará por medio de una librería en Python, en donde se utilizará la programación orientada a objetos para estructurar la librería. La ejecución de la herramienta se pondrá a prueba sobre la RPi y el servidor de SQL correrá sobre un equipo Windows. Esta librería busca aprovechar todas las herramientas que provee Microsoft para el entorno de SQL Server, pudiéndola unificar con elementos externos como lo son los sistemas embebidos, gracias a la versatilidad y portabilidad que ofrece Python.

9.1. Instalación de herramientas para ecosistema SQL

Las herramientas a utilizar son las siguientes:

- SQL Server: es un servidor de bases de datos SQL. Se utilizará la versión para desarrolladores que tiene límites de memoria para los datos a ingresar. Sin embargo, es suficiente para los fines experimentales del trabajo a realizar.
- SQL Server Management Studio (SSMS): es el entorno para gestionar las bases de datos del servidor, sus tablas, automatizaciones y otras diversas características. También provee un editor especializado para realizar *queries*.
- SQL Server Integration Services (SSIS): su función principal es facilitar la integración y

el movimiento de datos entre diferentes fuentes y destinos. SSIS permite la creación de paquetes que definen flujos de trabajo para tareas como la extracción, transformación y carga de datos (ETL).

- SQLcmd: es una utilidad especializada para la terminal de Windows. Esta permite la interacción hacia servidores de SQL por medio de comandos. Esta utilidad viene instalada por defecto dentro de los sistemas Windows.

9.1.1. Instalación de SQL Server

Lo primero que se debe de descargar es el servidor de base de datos. Este se puede obtener directamente desde la página de Microsoft, en el siguiente link: `https://www.microsoft.com/es-es/sql-server/sql-server-downloads`. Para este trabajo se utilizará la versión para desarrolladores. Si se desea usar otra de las versiones del servidor se debe de pagar una licencia para su uso. Una vez descargado el ejecutable se debe de instalar. Al correr dicho programa el proceso de instalación es bastante sencillo, solo se deben de seleccionar las opciones resaltadas en las siguientes imágenes.

Figura 9: Selección del tipo de instalación

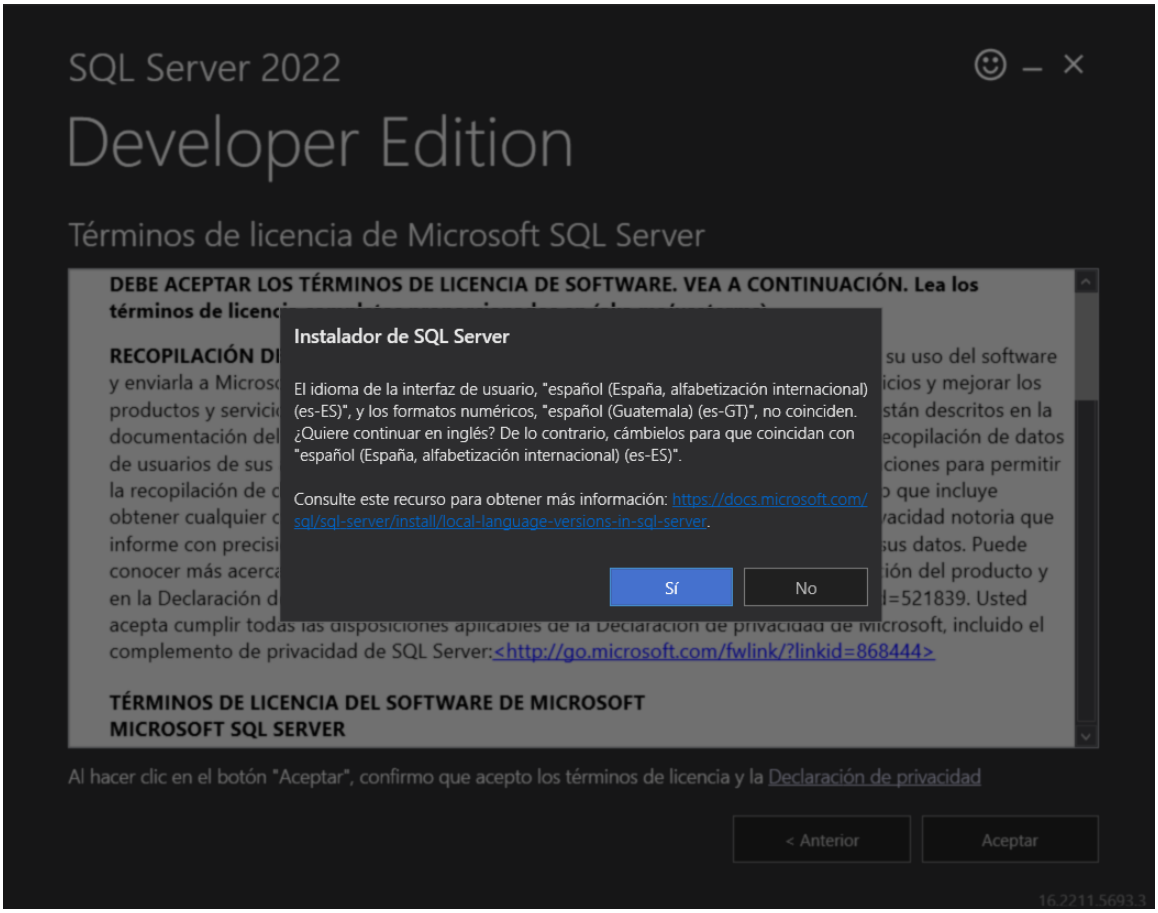


Figura 10: Términos de licenciamiento para el servidor

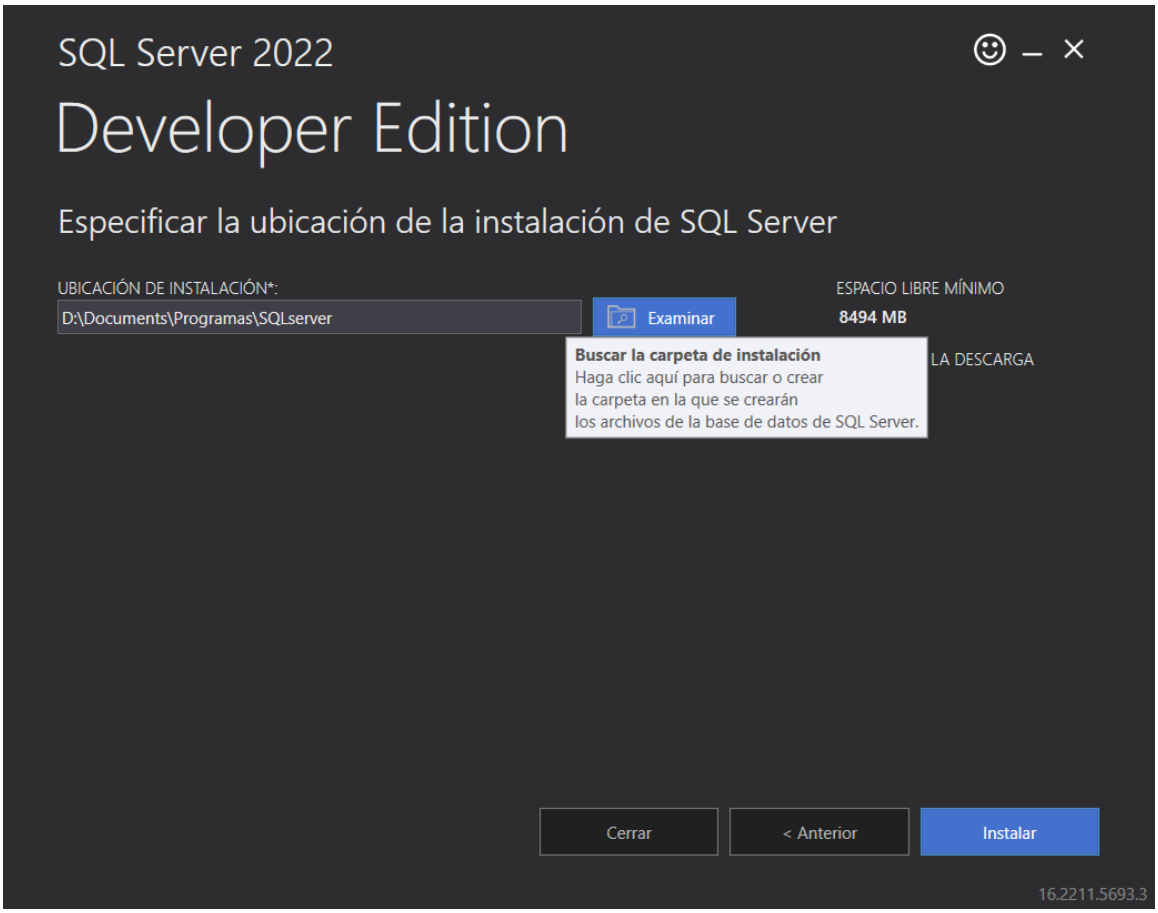


Figura 11: Selección de ruta de instalación

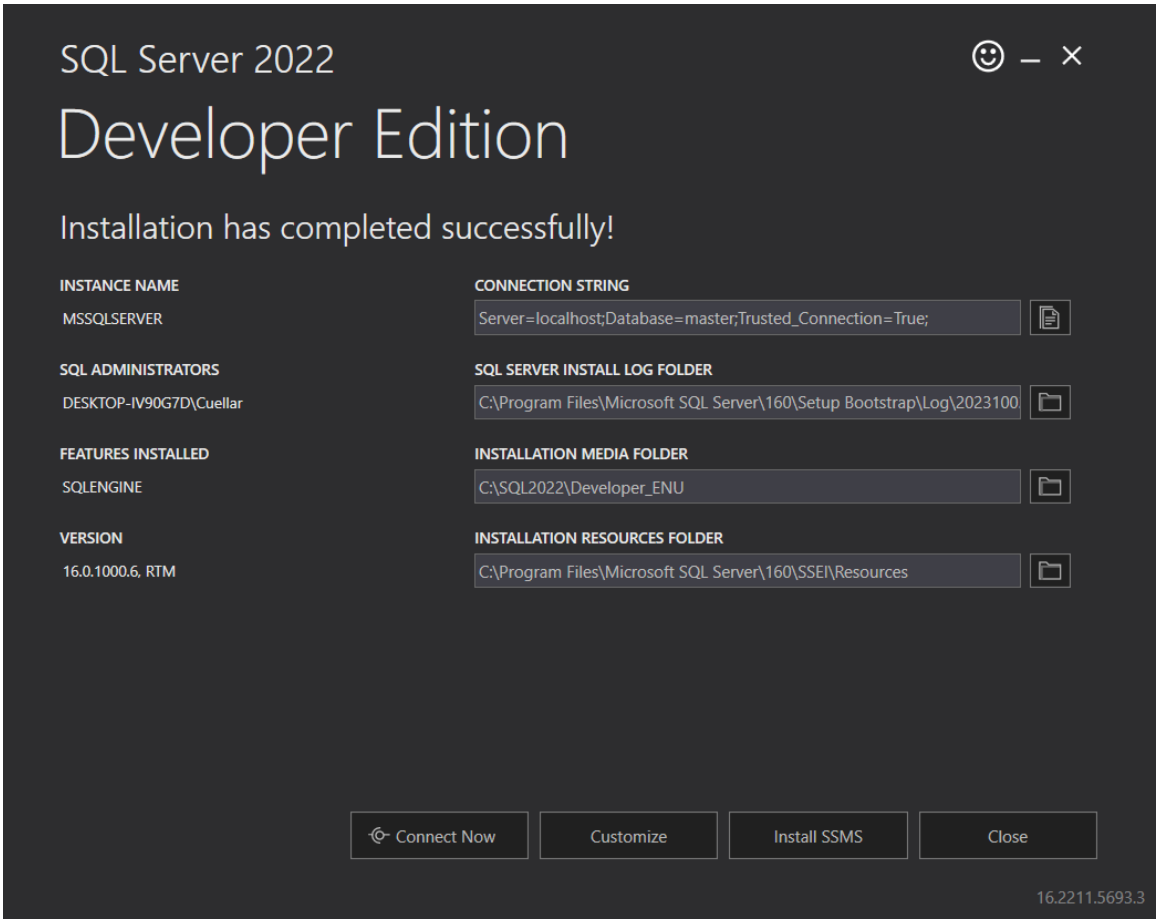


Figura 12: Pantalla indicadora de finalización del proceso

9.1.2. Instalación de SSMS

SSMS se puede descargar directamente desde la pantalla de finalización del proceso anterior, como se puede apreciar en la figura 12. Si no se desea utilizar dicho enlace, de igual manera se puede descargar la versión más reciente directamente desde la página oficial de Microsoft en el siguiente link: `https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16#download-ssms`. Una vez más este proceso es bastante sencillo, solo se deben de seguir las opciones resaltadas en las siguientes imágenes.

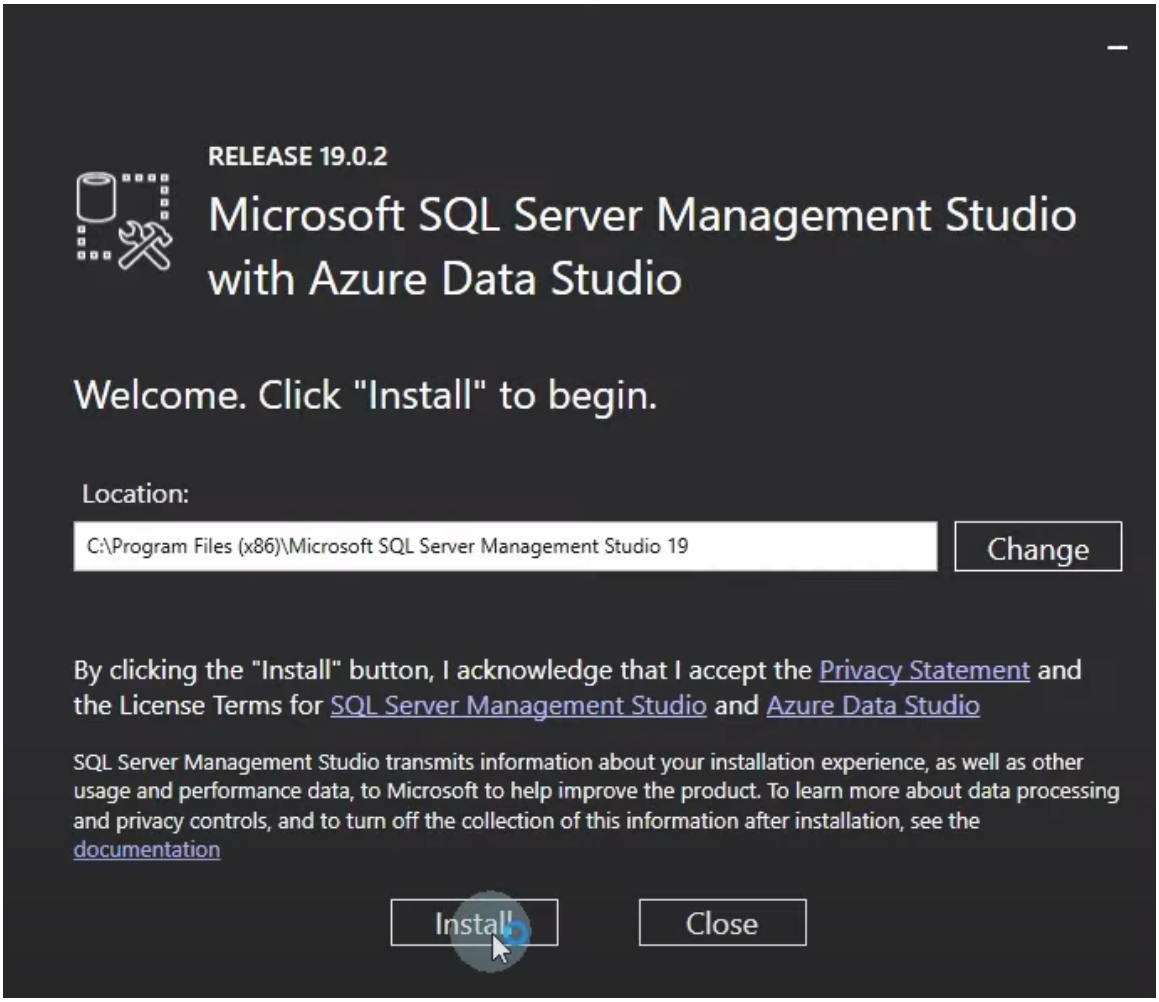


Figura 13: Selección de ruta de instalación para SSMS

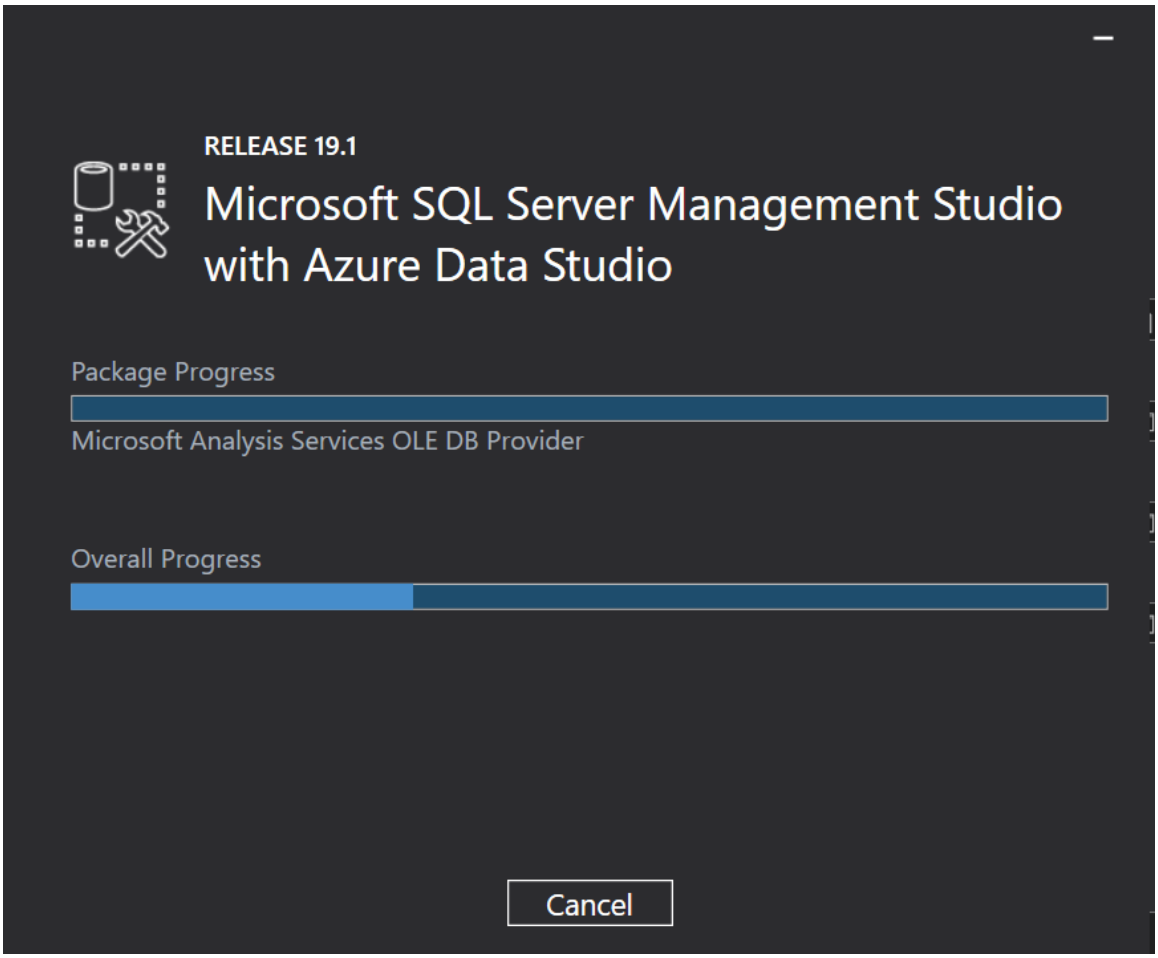


Figura 14: Carga de instalación de SSMS

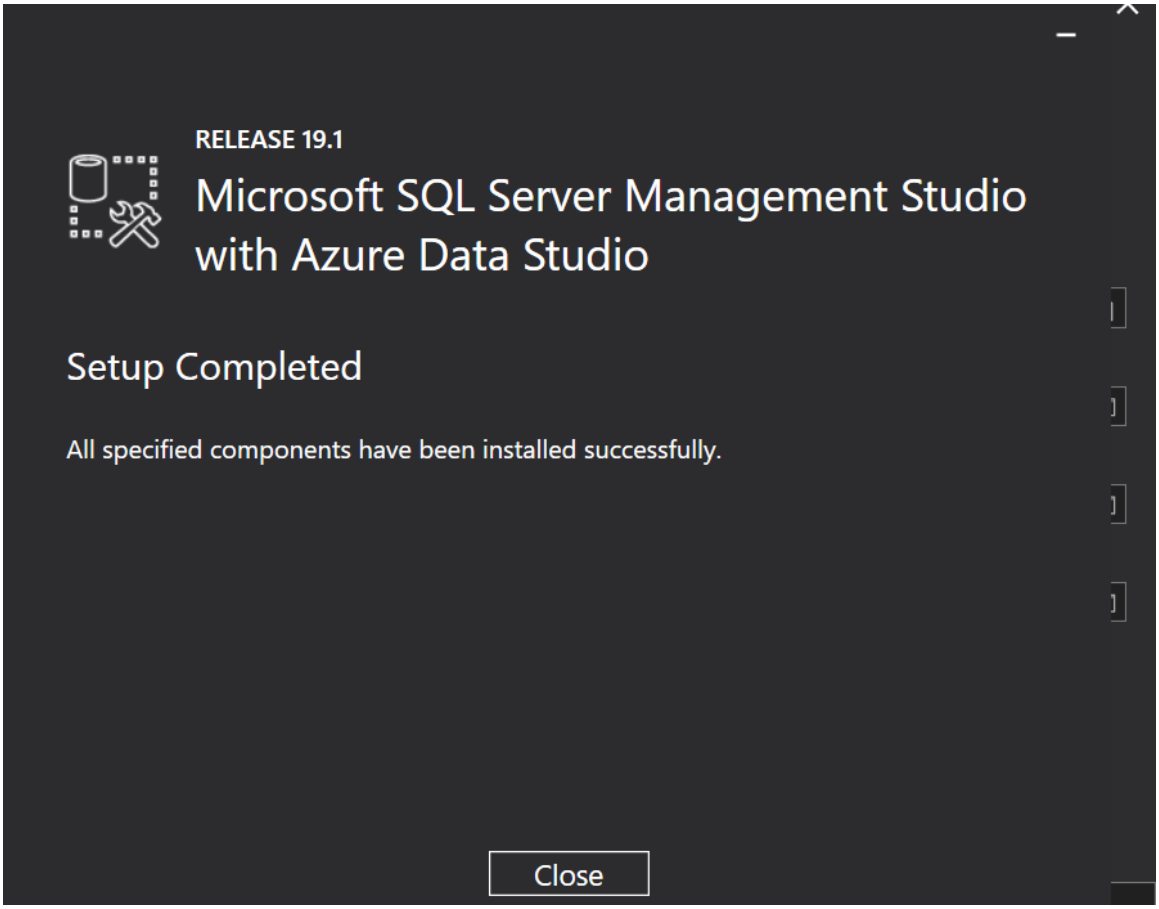


Figura 15: Fin de instalación de SSMS

9.1.3. Instalación de SSIS

Para poder utilizar SSIS se debe de instalar también Visual Studio, ya que por medio de este entorno se interactuará con los paquetes generados por la herramienta. Visual Studio se puede descargar desde este link: `https://visualstudio.microsoft.com/es/downloads/`. Una vez descargado el ejecutable del instalador se deben de realizar lo siguiente:

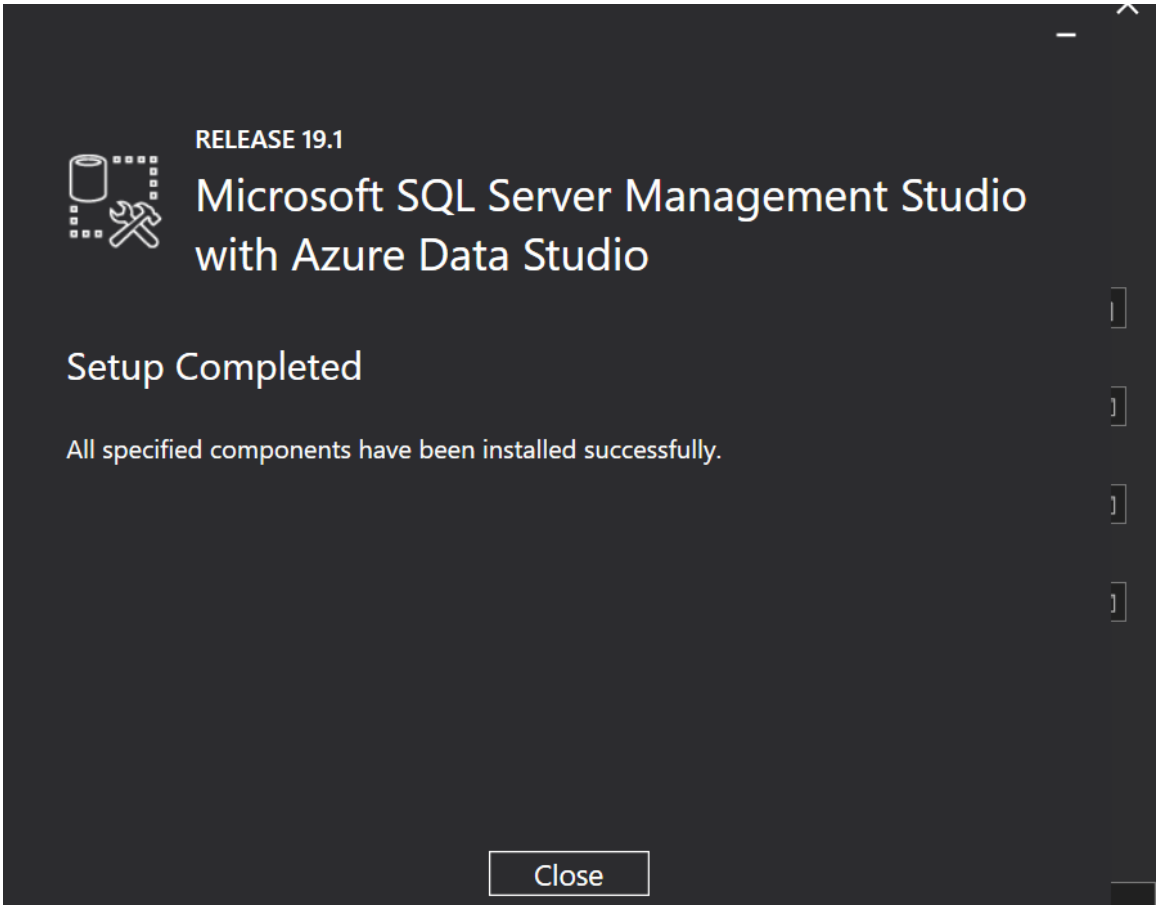


Figura 16: Selección de paquete de SSIS en instalación de Visual Studio

Una vez realizado, se selecciona la opción de instalar. Una vez concluida esta instalación y todas las demás descritas anteriormente se debe de reiniciar el equipo, para poder concluir los procesos de manera correcta y utilizar el software.

9.2. Configuración y preparación de SQL Server

Una vez instalado el servidor SQL se debe de garantizar de que el servicio esté instalado y verificar las propiedades de este. Aquí entra en juego la ventaja de correr dicho servicio sobre un equipo Windows, ya que el servidor por defecto se configura para tener credenciales por *Windows Authentication*, esto significa que se ingresa al servidor por medio del usuario de Windows al que pertenece el servidor. Esta verificación de propiedades se puede hacer por medio del programa **SQL Server Configuration Manager**, el cuál se instala de manera automática al realizar las instalaciones previas de esta sección.

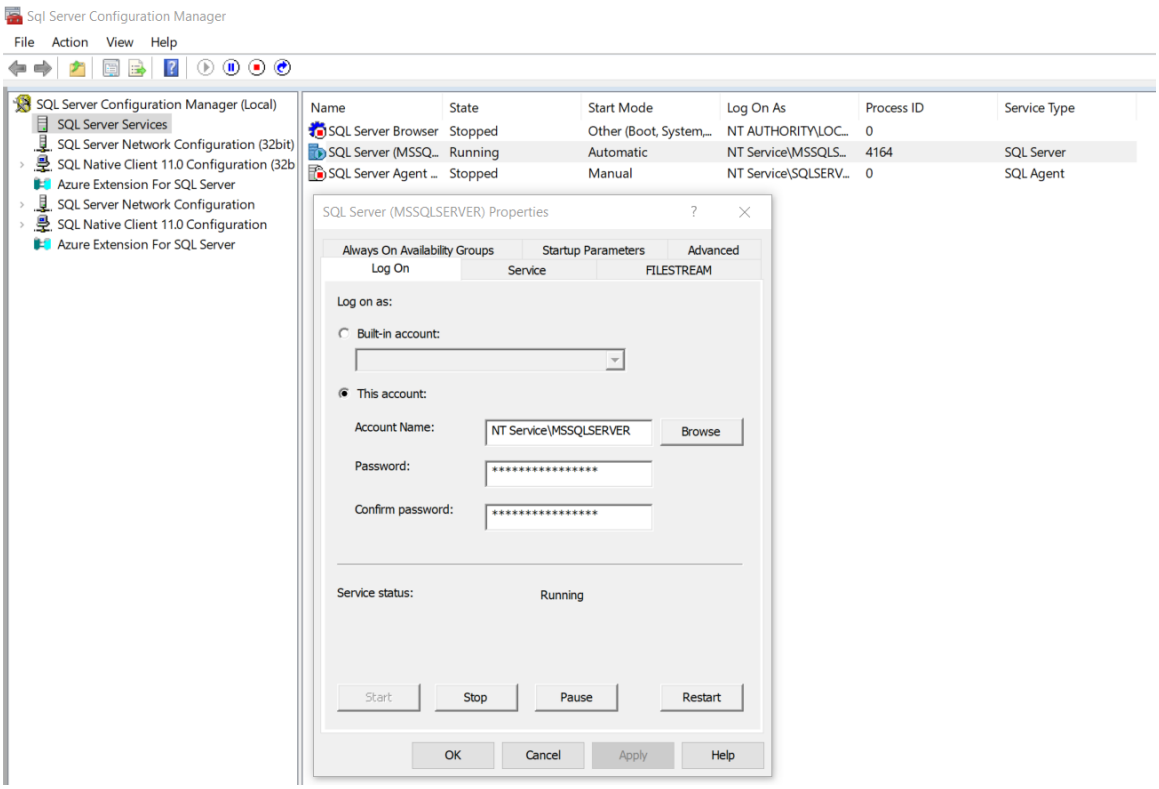


Figura 17: Verificación de propiedades del servidor SQL

Seguido a esto, se debe de inicializar SSMS para verificar la conectividad hacia el servidor. Al correr el programa se deben de ingresar las credenciales del servidor. Como se mencionó anteriormente, por utilizar *Windows Authentication*, únicamente se debe de indicar el nombre del servidor si se está trabajando desde el usuario de Windows *host* del servidor. El nombre del servidor corresponde al *host name* del equipo en donde se esté corriendo. En Windows esto se puede verificar por medio del comando *whoami* en el CMD.

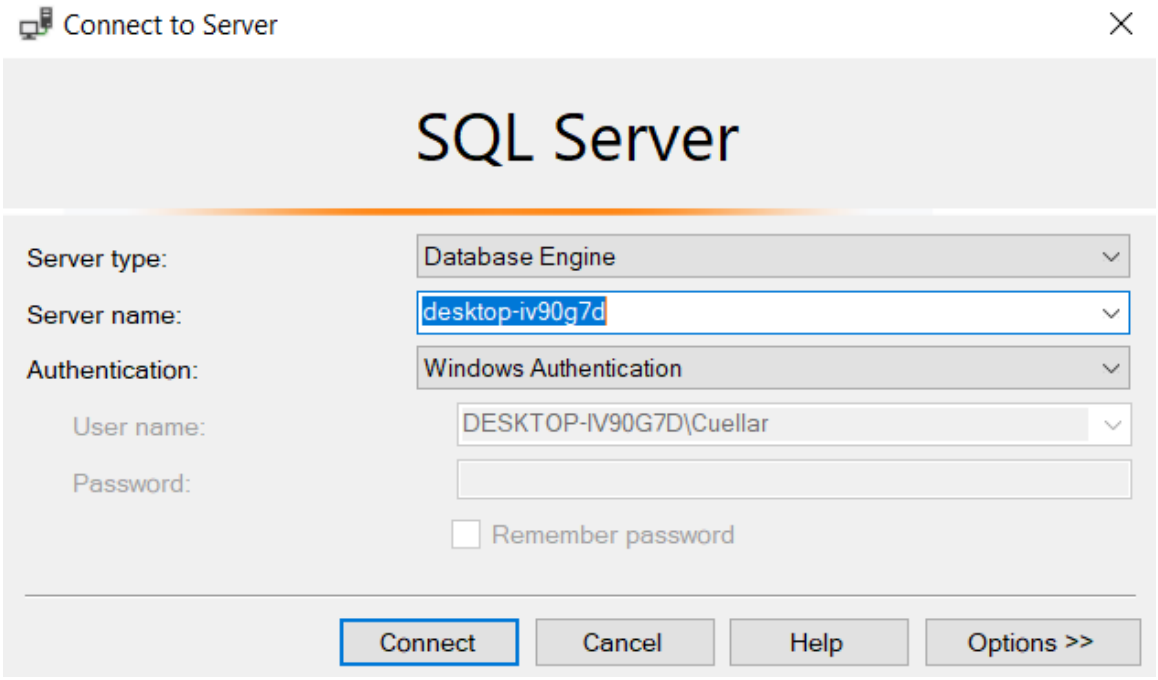


Figura 18: Ingreso de credenciales SSMS

Finalmente, el último paso para poder terminar de configurar el servidor para el desarrollo de la herramienta integradora, es la creación de un usuario SQL para el servidor. Esto se puede hacer desde el directorio de *Logins* del servidor en SSMS al presionar clic derecho sobre este.

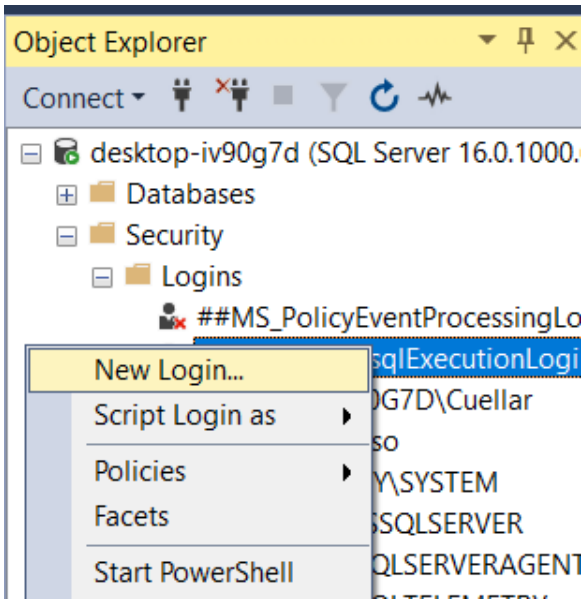


Figura 19: Creación de nuevo usuario SQL

Al crear el usuario se le debe de asignar un nombre y contraseña, distintos a los usuarios

preexistentes en el servidor. Este usuario será el medio por el cuál se ingresará al servidor de base de datos desde la RPi. Esto debido a que para poder acceder desde equipos externos, no se puede utilizar *Windows Authentication*, es necesario tener otro usuario SQL específico para el servidor. Para que este usuario tenga permisos de realizar actividades desde *queries* como crear tablas, manipular registros, crear automatizaciones... se deben de configurar los roles como se ejemplifica en la imagen a continuación.

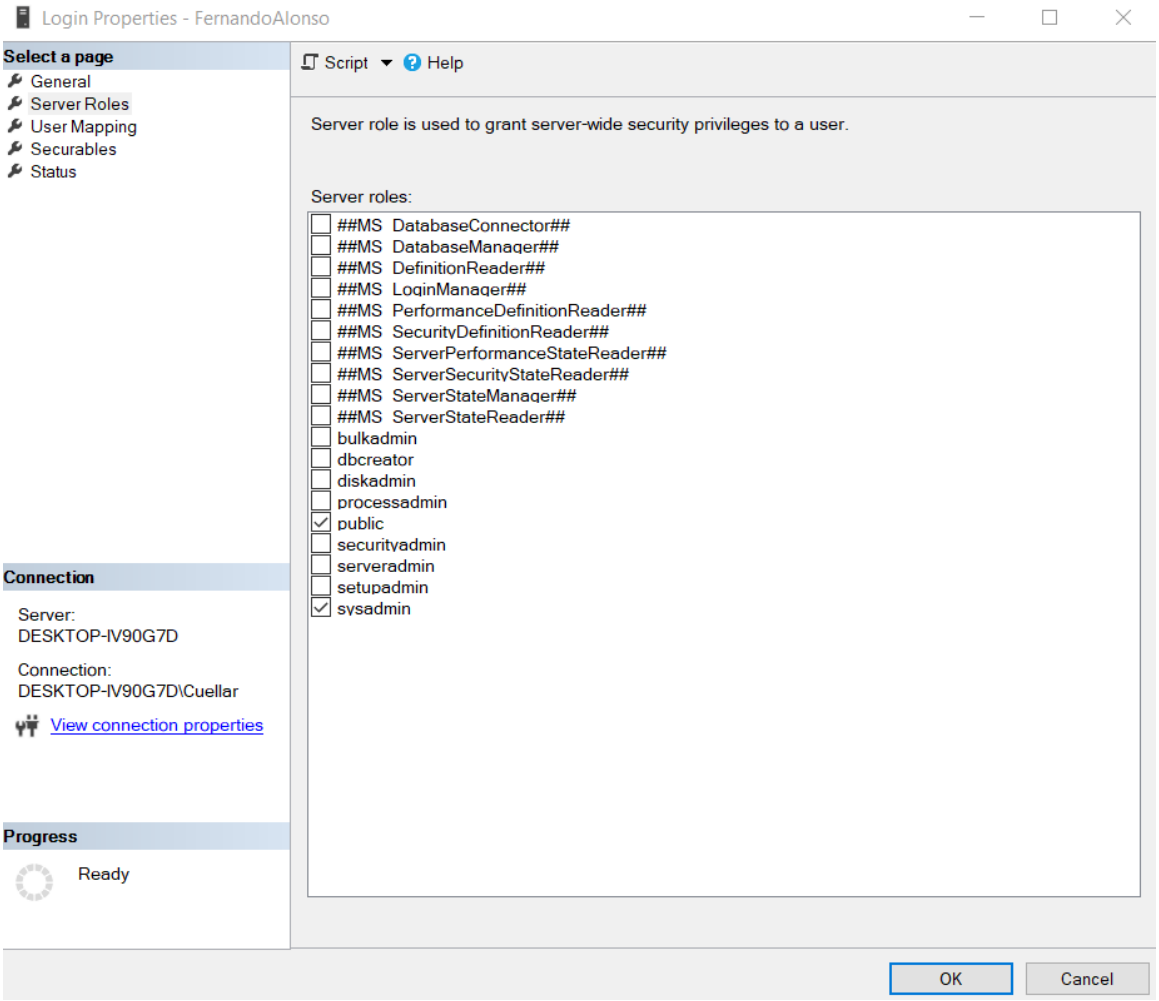


Figura 20: Roles de usuario SQL

9.3. Desarrollo de librería pySQLino

La librería titulada pySQLino es la encargada de realizar la integración automática entre los *boards* de Arduino y la base de datos SQL Server. Para realizar esto se necesitan diversas librerías de Python, estas pertenecen al conjunto de herramientas que vienen instaladas con el sistema para facilitar la implementación. Estas librerías son usadas para poder acceder a la terminal desde el *script*, interacción con los puertos seriales del equipo y la implementación de hilos. A continuación, se muestra el código fuente de la librería y posteriormente se discute la estructura de esta.

```
'''
 ================================================================================
 Nombre: Librería pySQLino
 Autor:  Carlos Ricardo Cuellar Klingenberger
 ================================================================================
 '''
import os
import serial
import serial.tools.list_ports
import subprocess
import serial
import time
import threading
#funciones

#función para buscar el puerto donde esta conectado el arduino
def find_arduino_port():
    # Lista de los puertos seriales disponibles
    ports = list(serial.tools.list_ports.comports())

    for port in ports:
        # Se revisa que este conectado un arduino uno
        if "Arduino Uno" in port.description:
            return port.device

    return None

#funcion para aceptar conexión con arduino
def connect_to_arduino(port):
    try:
        # Se abre una conexión serial hacia el arduino
        ser = serial.Serial(port, 9600)  # Adjust baudrate as needed
        print(f"Connected to Arduino Uno on {port}")

        # Add your Arduino communication logic here

        #while True:
        #    a=0

        # Close the serial connection when done
        #ser.close()
    except Exception as e:
        print(f"Error conectando al Arduino: {e}")
        raise
    return ser

#clases
#Clase de intereaccion de ArduinoUno con entrono de SQL
class UNO_SQL:
    def __init__(self, server_name, user, psswd,board_name,serial_port=None,
    DB=None):
        # Se inician los atributos de la clase
        #Estos son las credenciales para el servidor y el puerto serial del
    objeto
        self.server_name = server_name
        self.user = user
        self.psswd = psswd
        self.serial_port=serial_port
        self.board_name=board_name
```

```
        self.input_pins=None
        self.output_pins=None
        self.analogin_pins=None
        self.Dig_Inp=[None,None,None,None,None,None]
        self.Dig_Out=[None,None,None,None,None,None]
        self.An_Inp=[None,None,None,None,None,None]

#########Se crean métodos##################
    #Método para verificar conexión al server
    def check_server_conn(self):
        # Se establece la conexión
        connection_string = f"""sqlcmd -S {self.server_name} -U {self.user} -P {self.psswd} -Q "GO" """
        #connection_string = f"""sqlcmd -S {self.server_name} -U {self.user} -P {self.psswd} -Q "CREATE DATABASE Prueba" """

        try:
            # Se ejecuta la conxión por medio del cmd
            subprocess.run(connection_string, shell=True, check=True)
            print(f"Conexión exitosa al servidor {self.server_name}.")
        except subprocess.CalledProcessError as e:
            print(f"Error al conectar al servidor: {e}")
    #Método para establecer conexión al board de arduino
    def board_conn(self):
        arduino_port = find_arduino_port()
        if arduino_port:
            # Se conecta al arduino por medio del puerto serial de la clase
            self.serial_port=connect_to_arduino(arduino_port)
        else:
            raise RuntimeError("Fallo la conexion hacia el Arduino Uno.")
        response = b''
        #Ciclo que espera respuesta por parte del arduino
        while True:
            response=self.serial_port.readline().decode().strip()
            #self.serial_port.write("connrequest\n".encode())
            #Decodificación de la respuesta
            print(response)
            if response=='connrequest':
                self.serial_port.write("connconfirmed\n".encode())
                break
            #Lógica para proceder únicamente bajo la respuesta esperada por la recepciíon
                #de establecimiento de repsuesta
            if response.endswith('\n'):
                decoded_response = response.decode().strip()
                print(f"Received: {decoded_response}")
                if decoded_response == 'connrequest':
                    print("Arduino confirmed connection.")
                    break
                response = b''  # Reset response for the next line
    #Método para desconexión con el board
    def close_board_conn(self):
        self.serial_port.close()
    #Función para crear una base de datos para el arduino conectado
    #Método para crear la base de datos del board
    def create_board_DB(self,DB_name):
        #Ingreso de query para crear DB
        self.DB=DB_name
```

```
113         comando_cmd=f"""sqlcmd -S {self.server_name} -U {self.user} -P {self.
    psswd} -Q "CREATE DATABASE {DB_name}" """
114         try:
115             subprocess.run(comando_cmd, shell=True, check=True)
116             print(f"Database {DB_name} creada o existente.")
117         except subprocess.CalledProcessError as e:
118             print(f"Error creando database: {e}")
119     #Metodo para crear la tabla respectiva. Se hace en función te los valores
    ingresados para la clase
120     def create_table(self):
121         #comando_cmd=f"""sqlcmd -S {self.server_name} -U {self.user} -P {self.
    psswd} -Q "CREATE TABLE {self.DB}.dbo.{self.board_name}_Digtial_IO (date
    DATETIME DEFAULT GETDATE(),Input_1 NVARCHAR(255) NULL,Input_2 NVARCHAR
    (255) NULL,Input_3 NVARCHAR(255) NULL,Input_4 NVARCHAR(255) NULL,Input_5
    NVARCHAR(255) NULL,Input_6 NVARCHAR(255) NULL,Output_1 NVARCHAR(255) NULL,
    Output_2 NVARCHAR(255) NULL,Output_3 NVARCHAR(255) NULL,Output_4 NVARCHAR
    (255) NULL,Output_5 NVARCHAR(255) NULL,Output_6 NVARCHAR(255) NULL);" """
122         comando_cmd=f"""sqlcmd -S {self.server_name} -U {self.user} -P {self.
    psswd} -Q "CREATE TABLE {self.DB}.dbo.{self.board_name}_All_IO (date
    DATETIME DEFAULT GETDATE(),Input_1 NVARCHAR(255) NULL,Input_2 NVARCHAR
    (255) NULL,Input_3 NVARCHAR(255) NULL,Input_4 NVARCHAR(255) NULL,Input_5
    NVARCHAR(255) NULL,Input_6 NVARCHAR(255) NULL,Output_1 NVARCHAR(255) NULL,
    Output_2 NVARCHAR(255) NULL,Output_3 NVARCHAR(255) NULL,Output_4 NVARCHAR
    (255) NULL,Output_5 NVARCHAR(255) NULL,Output_6 NVARCHAR(255) NULL,
    Analog_1 NVARCHAR(255) NULL,Analog_2 NVARCHAR(255) NULL,Analog_3 NVARCHAR
    (255) NULL,Analog_4 NVARCHAR(255) NULL,Analog_5 NVARCHAR(255) NULL,
    Analog_6 NVARCHAR(255) NULL);" """
123         try:
124             subprocess.run(comando_cmd, shell=True, check=True)
125             print(f"Tabla {self.board_name} creada o existente.")
126 
127         except subprocess.CalledProcessError as e:
128             print(f"Error creando Tabla: {e}")
129     #Se configurar los inputs digitales. Se pueden tener unicamente 6 inputs
    digitales
130     #Se ingresa la CANTIDAD de pines que se desan configurar
131     #Se pueden tener unicamente un maximo de 6 pines de entrada digital
132     #Los pines posibles son del pin 8 al 13 de los GPIO de arduino
133     def config_dig_input(self,pins):
134         self.serial_port.reset_input_buffer()
135         while True:
136             if pins<=0:
137                 print('Se ingreso un número incorrecto de pines')
138                 break
139             self.serial_port.write(f"input_{pins}\n".encode())
140             response=self.serial_port.readline().decode().strip()
141             if response=='inputdone':
142                 self.input_pins=pins
143                 break
144      #Se configurar los outputs digitales. Se pueden tener unicamente 6
    outputs digitales
145     #Se ingresa la CANTIDAD de pines que se desan configurar
146     #Se pueden tener unicamente un maximo de 6 pines de entrada digital
147     #Los pines posibles son del pin 2 al 7 de los GPIO de arduino
148     def config_dig_output(self,pins):
149         self.serial_port.reset_input_buffer()
150         while True:
151             if pins<=0:
152                 print('Se ingreso un número incorrecto de pines')
```

```
                break
            self.serial_port.write(f"output_{pins}\n".encode())
            response=self.serial_port.readline().decode().strip()
            if response=='outputdone':
                self.output_pins=pins
                break
    #Se solicita leer los pines digitales
    #Se lee automaticamente según la cantidad de pines congifurados
    previamente
    def read_dig_inputs(self):
        self.serial_port.reset_input_buffer()
        flag_break=0
        while True:

            if self.input_pins==None:
                print('No se han configurado pines de entrada')
                break
    #Se lee automáticamente según los pines asignados a la clase
    #A continuación se tiene la logica para leer la infomración esperada
            self.serial_port.write(f"readinput:\n".encode())
            response=self.serial_port.readline().decode().strip()
            if (response.startswith("readinput:")) & (len(response)==21):
                valores=response.split(":")[1].split("_")

                for i in range(len(self.Dig_Inp)):
                    if valores[i]=="N":
                        self.Dig_Inp[i]=None
                    else:
                        self.Dig_Inp[i]=valores[i]
                    if i==5:
                        flag_break=1
            if flag_break==1:
                self.serial_port.write(f"readingdone\n".encode())
                print(f'Los valores de los pines de entrada son: {self.Dig_Inp
    }')
                comando_cmd = f"""sqlcmd -S {self.server_name} -U {self.user}
    -P {self.psswd} -Q "INSERT INTO {self.DB}.dbo.{self.board_name}_All_IO (
    Input_1, Input_2, Input_3, Input_4, Input_5, Input_6, Output_1, Output_2,
    Output_3, Output_4, Output_5, Output_6,Analog_1,Analog_2,Analog_3,Analog_4
    ,Analog_5,Analog_6) VALUES ({', '.join(['NULL' if val is None else str(val
    ) for val in self.Dig_Inp + self.Dig_Out + self.An_Inp])});" """
                subprocess.run(comando_cmd, shell=True, check=True)
                break
    #Se escribe de manera individual al pin y el estado que se quiere escribir
    #Se protege el codigo para unicamente recibir pines dentro del tango de
    configuracion

    def write_dig_output(self, pin, state):
        self.serial_port.reset_input_buffer()
        while True:
            if (pin>self.output_pins) or (pin==0):
                print("Se ingreso un numero incorrecto de pin de salida")
                break
            '''if state!=0 or state!=1:
                print("Se ingreso un estado incorrecto")
                break'''
            if self.output_pins==None:
                print('No se han configurado pines de salida')
                break
```

```
204             self.serial_port.write(f"writeoutput:{pin}_{state}\n".encode())
205             #print(f"writeoutput:{pin}_{state}")
206             response=self.serial_port.readline().decode().strip()
207             print(response)
208             if response=='outputwritedone':
209                 print(f"Se escribio el pin {pin} en {state}")
210                 self.Dig_Out[pin-1]=state
211                 # Assuming Dig_Inp and Dig_Out are lists of values or None
212                 #comando_cmd = f"""sqlcmd -S {self.server_name} -U {self.user} -P {self.psswd} -Q "INSERT INTO {self.DB}.dbo.{self.board_name}_Digtial_IO (Input_1, Input_2, Input_3, Input_4, Input_5, Input_6, Output_1, Output_2, Output_3, Output_4, Output_5, Output_6) VALUES ({', '.join(['NULL' if val is None else str(val) for val in self.Dig_Inp + self.Dig_Out])});" """
213                 comando_cmd = f"""sqlcmd -S {self.server_name} -U {self.user} -P {self.psswd} -Q "INSERT INTO {self.DB}.dbo.{self.board_name}_All_IO (Input_1, Input_2, Input_3, Input_4, Input_5, Input_6, Output_1, Output_2, Output_3, Output_4, Output_5, Output_6,Analog_1,Analog_2,Analog_3,Analog_4,Analog_5,Analog_6) VALUES ({', '.join(['NULL' if val is None else str(val) for val in self.Dig_Inp + self.Dig_Out + self.An_Inp])});" """
214
215
216
217                 subprocess.run(comando_cmd, shell=True, check=True)
218
219                 break
220     #Se lee la entrada de inputs analogicos
221     #Por la arcquitecura de arduino estos no requieren de un setup previo para congigurar
222     #Se puede recibir unicamente como analogico info de los pones AO a A5 del arduino UNO
223     #Se utiliza la misma logica para separar datos que en la entrada digital, son emnbargo con distintos delimitadores
224     def read_an_inputs(self, pins):
225         self.serial_port.reset_input_buffer()
226         self.analogin_pins=pins
227         flag_break=0
228         while True:
229             if self.analogin_pins==None:
230                 print('No se han configurado pines de entrada analo')
231                 break
232             self.serial_port.write(f"analoginput_{self.analogin_pins}\n".encode())
233             response=self.serial_port.readline().decode().strip()
234             if (response.startswith("readanaloginput:")) & (response.endswith("END")):
235                 valores=response.split(":")[1].split("_")
236                 for i in range(len(self.An_Inp)):
237                     print(f'LOS VALORES SON: {valores}')
238                     if valores[i]=="N":
239                         self.An_Inp[i]=None
240                     else:
241                         self.An_Inp[i]=valores[i]
242                     if i==self.analogin_pins:
243                         flag_break=1
244
245             if flag_break==1:
246                 self.serial_port.write(f"readingdone\n".encode())
```

```
247                 print(f'Los valores de los pines analogicos de entrada son: {
    self.An_Inp}')
248                 comando_cmd = f"""sqlcmd -S {self.server_name} -U {self.user}
    -P {self.psswd} -Q "INSERT INTO {self.DB}.dbo.{self.board_name}_All_IO (
    Input_1, Input_2, Input_3, Input_4, Input_5, Input_6, Output_1, Output_2,
    Output_3, Output_4, Output_5, Output_6,Analog_1,Analog_2,Analog_3,Analog_4
    ,Analog_5,Analog_6) VALUES ({', '.join(['NULL' if val is None else str(val
    ) for val in self.Dig_Inp + self.Dig_Out+self.An_Inp])});" """
249                 subprocess.run(comando_cmd, shell=True, check=True)
250                 break
```

Listing 24: Código de librería pySQLino

La librería está desarrollada bajo el paradigma de POO. Esto permite que se puedan usar múltiples instancias de las clases para diversos equipos que se encuentren conectados por medio de puertos seriales al sistema principal. Los atributos de la clase *UNO_SQL* consisten en propiedades necesarias para realizar la conectividad hacia el servidor SQL y al servidor SQL. Esta clase esta especializada para la identificación del *board* Arduino Uno. Los métodos están orientados para la realización de las actividades necesarias para establecer las conexiones pertinentes. En caso, el ingreso erroneo de credenciales derive en una conexión fallida se diseñó la librería para que esta genere un error en el código. Esto se hizo para que se asegure la ejecución correcta de los métodos posteriormente.

Al observar los atributos de la clase, se puede ver que la mayoría se tendrán que saber desde el principio en que esta se convoque, bajo el código de implementación. Esto es cierto, pero el diseño de la librería esta pensado en buscar limitar lo menos posible, por lo que si el usuario desea realizar las cosas de un orden distinto al sugerido, tenga la libertad de poder seguir usando una herramienta dotada de adaptabilidad. Para el correcto uso de la librería se necesita que un Arduino esté preconfigurado para recibir el tipo de instrucciones que se le darán por medio de la interfaz serial. Para esto se realiza una preconfiguración del Arduino, en donde se le indica cómo almacenar y enviar los estados de sus pines de entrada y salida, al mismo tiempo, que cómo definir su configuración. Por motivos de buscar la simpleza en la interfaz de programación de Arduino que se otorgaría a través de Python, se eligió el diseño de que se decidiera cuántos pines de entrada se utilizarían en lugar de cuáles se deseaban usar. Esto así como tiene sus limitantes a la hora de la realización de circuitos, tiene su ventaja en la simpleza que se le dota al uso y familiarización con la librería.

```
1  int device_enable=1;
2  int input_quant = 0;
3  int analoginput_quant = 0;
4  int output_quant = 0;
5  int flag_pruebas=0;
6  int sel_state=0;
7  int sel_output=0;
8  String input_array[6];
9  String analoginput_array[]={"N","N","N","N","N","N"};
10 bool readcycle=true;
11 String concatenatedString="";
12 String concatenatedString2="";
13 String request="";
14 void setup() {
15   Serial.begin(9600);
16   pinMode(2,OUTPUT);
17   //Serial.setTimeout(100);
```

```
}

void process_request(String request){
  if (request.startsWith("input_")){
  input_quant = request.substring(6).toInt();
  for(int i=8; i<=(input_quant+7);i++){
    pinMode(i,INPUT);
    flag_pruebas=1;
    //Serial.println(i);
  }
  Serial.println("inputdone");
  }
    if (request.startsWith("analoginput_")){
  analoginput_quant = request.substring(12).toInt();
  for(int i=1; i<=(analoginput_quant)&& analoginput_quant!=0;i++){
    analoginput_array[i-1]=analogRead((i+13));
    //flag_pruebas=1;
    //Serial.println(i);
  }
  concatenatedString2 = "readanaloginput:"+analoginput_array[0] + "_" +
    analoginput_array[1] + "_" + analoginput_array[2] + "_" +
                              analoginput_array[3] + "_" + analoginput_array
    [4] + "_" + analoginput_array[5]+":END";
    Serial.println(concatenatedString2);
  }

  if (request.startsWith("output_")){
  output_quant = request.substring(7).toInt();
  for(int j=2; j<=(output_quant+1);j++){
    pinMode(j,OUTPUT);
  }
  Serial.println("outputdone");
  }
  if(request.startsWith("readinput:")){
    for (int i=1;i<=(input_quant) && input_quant!=0;i++){
      input_array[i-1]=digitalRead(i+7);
    }
    for (int i=input_quant+8;i<=13&&i>=8;i++){
      input_array[i-8]="N";
    }
    readcycle=false;
    concatenatedString = "readinput:"+input_array[0] + "_" + input_array[1] +
    "_" + input_array[2] + "_" +
                              input_array[3] + "_" + input_array[4] + "_" +
    input_array[5];
    Serial.println(concatenatedString);
}
if(request.startsWith("writeoutput:")){
  sel_output=request.substring(12).toInt();
  sel_output=sel_output+1;
  sel_state=request.substring(14).toInt();
  digitalWrite(sel_output,sel_state);
  Serial.println("outputwritedone");
}
}
/*void read_request(String request){

}*/
void loop() {
```

```
  // Check if data is available to read
  if (Serial.available() > 0) {
    // Read the incoming data
    request = Serial.readStringUntil('\n');
    process_request(request);
    // Check if the received data is 'connrequest'
    if (request.equals("connconfirmed")) {
      // Send 'connconfirmed' back to Python
      delay(500);
      Serial.println("Se logra autorizacion");
      device_enable=0;
    }
  }
  if (device_enable==1){
    Serial.println("connrequest");
  }

  if(flag_pruebas==1){
  }
  // Add your other Arduino logic here
}
```

Listing 25: Código de Arduino Uno para pySQLino

9.4. Implementación de la librería

A continuación, se muestra un *script* donde se crea la conexión hacia el servidor SQL y al Arduino, seguido a esto, se crea una base de datos y, finalmente, se cierran las conexiones. El código simplifica un proceso sumamente laborioso, que en principio consumiría una cantidad considerable de tiempo y la realización de las tareas necesitaría de la navegación entre diversos programas para lograr su cometido. Sin embargo, la eficiencia de la librería desarrollada, logra reducir este proceso a solo unas cuantas líneas de código. Este código es capaz de poblar bastante bien la base de datos, para la experimentación de uso de herramientas de visualización. Este mismo código puede usarse como el código a correrse en el proyecto que se propone para electrónica digital 3, como se sugiere en la guía anexa de este trabajo.

```
'''
 ================================================================
 Nombre: Implementacion pySQLino
 Autor:  Carlos Ricardo Cuellar Klingenberger
 ================================================================
 '''
from pySQLino import *
import time
# Crea una instancia de la clase UNO_SQL
sql_server_instance = UNO_SQL(server_name='desktop-iv90g7d', user='
    FernandoAlonso', psswd='Realmadridcf10', board_name='ArdJonathan')
# Se establece conexión al server SQL
sql_server_instance.check_server_conn()
#Se establece la conexión con el arduino
sql_server_instance.board_conn()
#Se crea una base de datos con el nombre ArdUno
sql_server_instance.create_board_DB('DemoTesis')
```

```
#Se cierra la tabla dentro de la DB creada
sql_server_instance.create_table()
#Configuracion de pines
sql_server_instance.config_dig_input(6)
sql_server_instance.config_dig_output(6)
#Lectura de pines
sql_server_instance.read_dig_inputs()
#Escritura de pines
sql_server_instance.write_dig_output(1,0)
sql_server_instance.write_dig_output(2,0)
sql_server_instance.write_dig_output(3,0)
sql_server_instance.write_dig_output(4,0)
sql_server_instance.write_dig_output(5,0)
sql_server_instance.write_dig_output(6,0)
estado=1
for x in range(200):
    estado = x % 2
    sql_server_instance.read_dig_inputs()
    sql_server_instance.write_dig_output(1,estado)
    sql_server_instance.write_dig_output(2,estado)
    sql_server_instance.write_dig_output(3,estado)
    sql_server_instance.read_an_inputs(3)
    print(f'Corrida {x}')
    #time.sleep(.5)
    pass
#Cierre de conexión serial
sql_server_instance.close_board_conn()

print('FIN')
```

Listing 26: Código de implementación de librería pySQLino

9.4.1. Visualización de datos

Con la implementación realizada de la librería, se puede exprimir el potencial al máximo de la presencia de bases de datos en nuestro sistema. Para esto nos enfocaremos en la visualización de la data del código anterior. Para realizar un *dashboard*, que contenga visuales relacionados a nuestra tabla se utilizara la herramienta *PowerBi*. Dentro de esta se exportará data, desde una fuente de servidor SQL, como se puede apreciar en las siguientes imágenes.

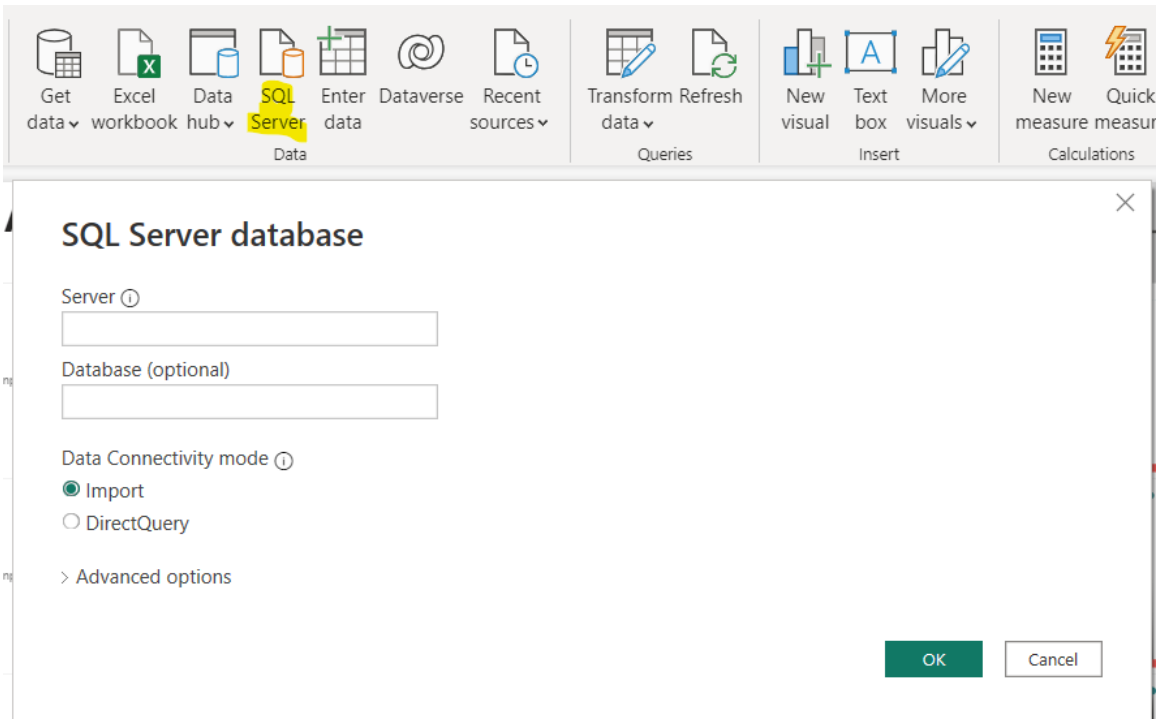


Figura 21: Extracción de datos en PowerBi

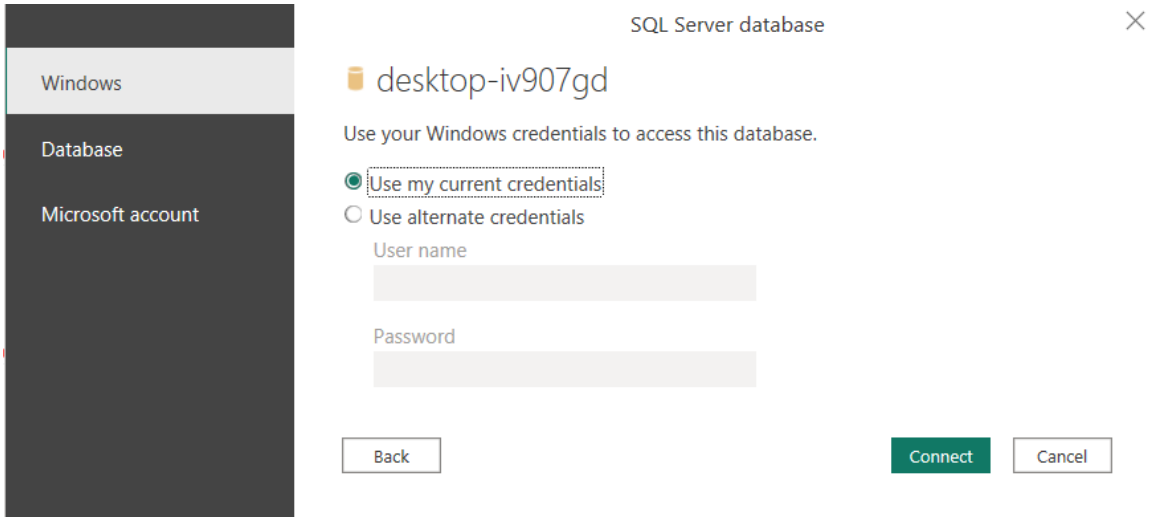


Figura 22: Credenciales para extracción

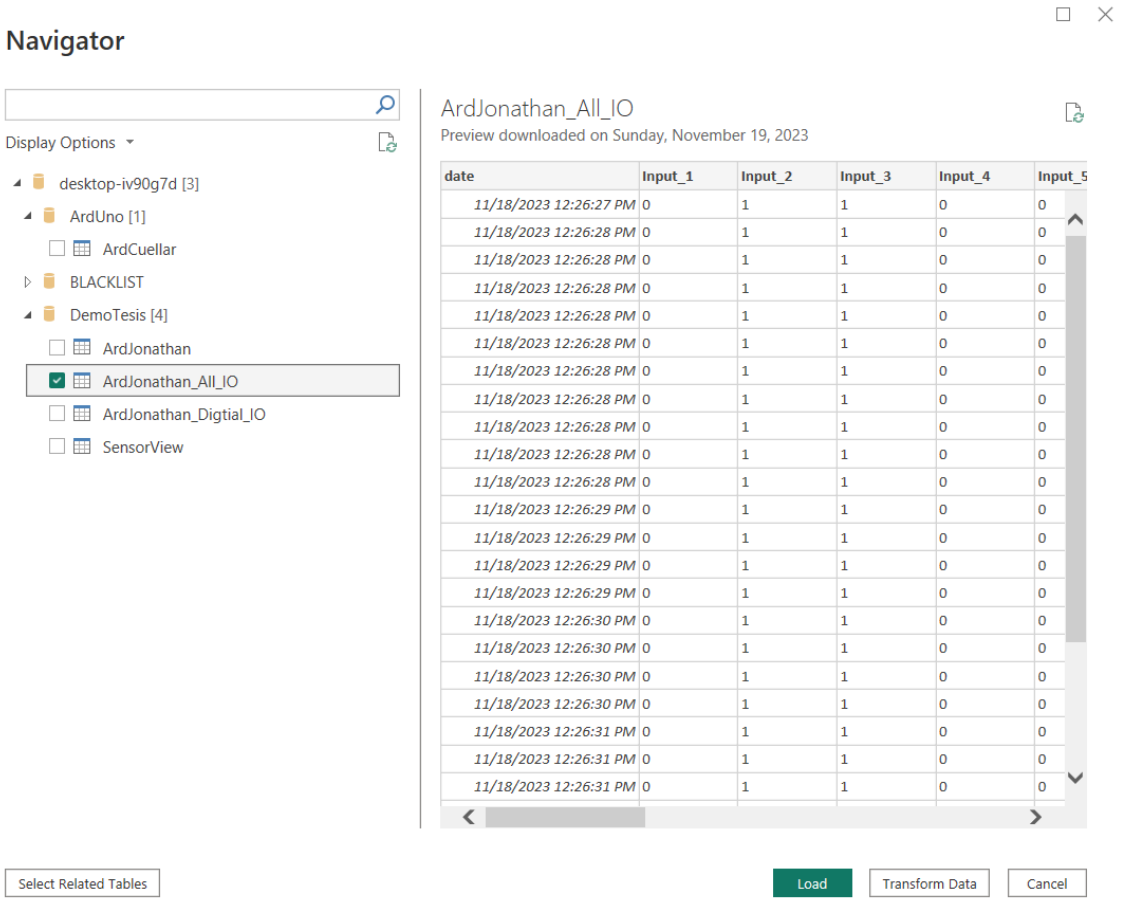


Figura 23: Selección de tabla para extracción

Con los pasos a seguir anteriormente es como se puede empezar a trabajar de manera inmediata con datos capturados de un sistema embebido directamente a una de las herramientas de visualización de datos mas potentes y utilizadas en la industria. Esta misma puede ser implementada para el proyecto, a continuación se ejemplifica lo que se puede llegar a hacer con la data de Arduino.

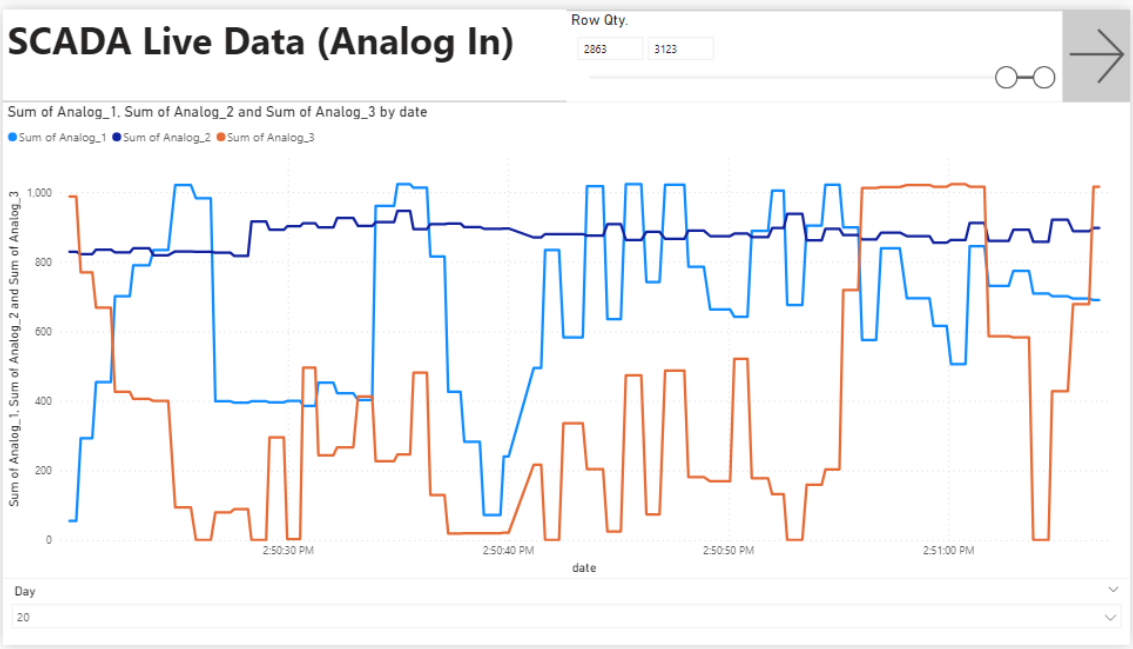


Figura 24: Dashboard utilizando pySQLino

9.4.2. Documentación

Todo lo anteriormente descrito en cuanto a la integridad de la librería (códigos y ejemplo), se puede encontrar en un repositorio público de *github*, dentro del siguiente enlace: https://github.com/Cue19275/pySQLino.git/. En dicho repositorio se puede encontrar el respectivo archivo *README.md* donde se explica a detalle como se puede utilizar la librería, pasos a seguir y software previo del cuál la librería depende para funcionar. Este repositorio público permite que los avances realizados en este trabajo pueda ser aprovechado e incluso mejorado por toda la comunidad concerniente al sector.

9.5. Planteamiento de proyecto, Electrónica Digital 3

Esta herramienta de software tiene mucho potencial didáctico, debido a su pertenencia a dos mundos muy importantes de la industria actual: bases de datos y sistemas embebidos. El uso de la librería podría ser la primer ventana a los estudiantes para observar la ventana de potencial que tiene la interacción entre estos dos sectores. Por lo tanto, se propone replantear el proyecto de curso de Electrónica Digital 3, para poner en práctica el potencial didáctico de la herramienta. Para esto se sugiere realizar un sistema *SCADA* (siendo fieles al proyecto original), pero en donde el historiador del sistema pasa a ser reemplazado totalmente por una base de datos. De esta manera la resolución del proyecto estaría plasmada en el extracto de código *Listing 9.3*, que esencialmente es la integración total de un *board* de sistemas embebidos con una base de datos, controlado por un mediador (el sistema corriendo el código de Python con el que se comunica el Arduino Uno) que en terminología del sistema *SCADA*, dicho mediador pasaría a ser el *RTU*. A continuación, se puede apreciar una imagen

donde se muestra la topología de conexión para la implementación.

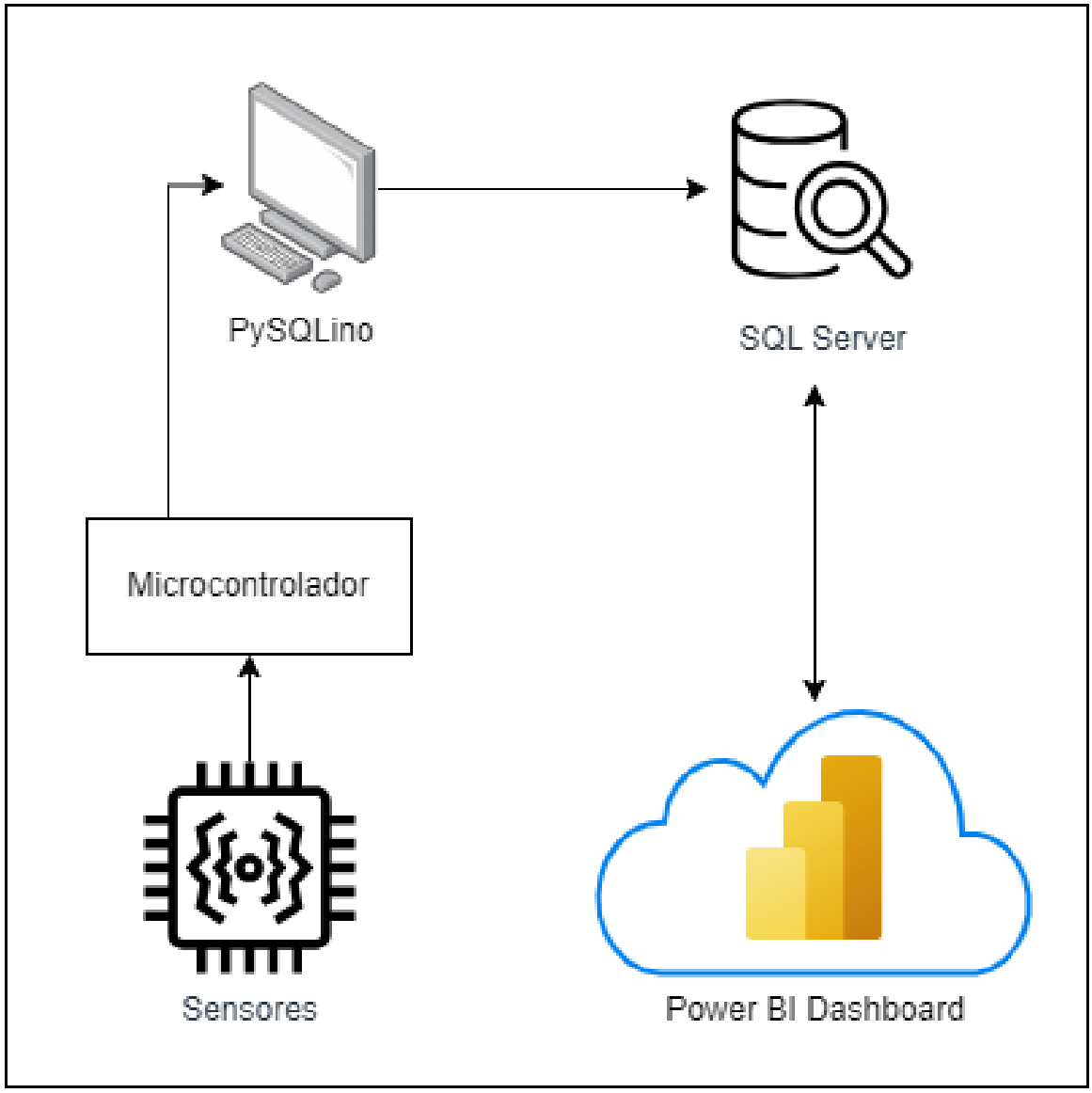


Figura 25: Topología de conexión para implementación

CAPÍTULO 10

Conclusiones

- Dado a que Python es un lenguaje interpretado, no puede gozar de un paralelismo genuino. Sin embargo, esto no impide que se pueda implementar el concepto de hilos y multiprocesamiento, por medio de algoritmos de escalonamiento internos. Esta cualidad de Python permitió que se pudiera desarrollar una librería con una función alternativa a los *forks* para generar códigos con multiprocesamiento en Windows.
- Se demuestra la eficiencia operativa del interpretador de Python por medio de la realización de benchmarks. Este es capaz de mantener un rendimiento constante a través de sistemas operativos con arquitectura de la misma familia.
- Se demostró, a través de herramientas desarrolladas por Microsoft, especializadas para el ecosistema de SQL Server, la capacidad de Python de manipular bases de datos de manera directa.
- La simpleza de sintaxis de Python, no imposibilita su aplicación para el campo educativo. La replicación exitosa de los laboratorios es muestra de esto.
- Fue posible el desarrollo de una librería capaz de integrar de manera automática sistemas embebidos con bases de datos.

CAPÍTULO 11

Recomendaciones

- A pesar del desarrollo de una librería para la utilización de una alternativa para los *forks* en Windows, se sugiere que el desarrollo de los laboratorios sea ejecutado en sistemas UNIX para poder experimentar con las funciones multiproceso de Python de una manera más enriquecedora desde el punto de vista didáctico.
- Un complemento de valor para la herramienta integradora de sistemas embebidos y bases de datos, sería expandir la gama de microcontroladores con la que esta puede interactuar, para no reducir su uso únicamente a *boards* de Arduino.
- Para brindar de mayor versatilidad la herramienta integradora de sistemas embebidos y bases de datos, se recomienda ampliar el tipos de bases de datos con los que esta puede comunicarse, para no reducir su funcionamiento únicamente a SQL Server.
- Desde un punto de vista didáctico, las implementaciones de algoritmos de escalonamiento en Python no ofrecen tanta libertad, como si lo ofrece el lenguaje C. Por lo tanto, es un tema cuya enseñanza se sugiere impartir en un lenguaje de bajo nivel como lo es C.
- Los laboratorios donde se manejan *sockets* y protocolos de comunicación (UDP, TCP) pueden ser adaptados para utilizarse en los cursos de telecomunicaciones. Esto por medio de una adaptación en donde, por medio de la terminal, se puedan desplegar detalles sobre los envíos de paquetes, en donde se despliegue información sobre los puertos y direcciones de destino según cada protocolo lo amerite.

CAPÍTULO 12

Bibliografía

[1] W. Cheah, J. Jegatheesan y S. Chee, *Exploring IOT application using raspberry pi*, https://ijcna.org/Manuscripts/Volume-2/Issue-1/Vol-2-issue-1-M-04.pdf, Accessed: 2023-4-24.

[2] M. S. D. Gupta, V. Patchava y V. Menezes, "Healthcare based on IoT using Raspberry Pi," en *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, 2015, págs. 796-799. DOI: 10.1109/ICGCIoT.2015.7380571.

[3] K. K. G. Punit Khatri y R. K. Gupta, *Raspberry Pi-based smart sensing platform for drinking-water quality monitoring system: a Python framework approach*, jul. de 2019.

[4] C. E. Wills, "Process synchronization and IPC," en, *ACM Comput. Surv.*, vol. 28, n.º 1, págs. 209-211, mar. de 1996.

[5] Z. A. Aziz, D. Naseradeen Abdulqader, A. B. Sallow y H. Khalid Omer, "Python parallel processing and multiprocessing: A rivew," *Acad. J. Nawroz Univ.*, vol. 10, n.º 3, págs. 345-354, ago. de 2021.

[6] A. Singh, P. Goyal y S. Batra, "An Optimized Round Robin Scheduling Algorithm for CPU Scheduling," */ (IJCSE) International Journal on Computer Science and Engineering*, vol. 02, n.º 07, págs. 2383-2385, 2010. dirección: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=780d07fbf09484a2490868d45255a23762a7e21d.

[7] N. Matloff y F. Hsu, *Tutorial on threads programming with python*, http://dominiquethiebaut.com/dftwiki3/images/5/58/Matlof_PythonTutorial.pdf, Accessed: 2023-5-22.

[8] B. Barney, *POSIX Threads Programming*, https://cse.iitkgp.ac.in/~abhij/course/lab/CompLab-I/sample/pthread.pdf, Accessed: 2023-5-22.

[9] V. Tomar y V. Kanaujia, *Python threading and its caveats*, en, https://www.opensourceforu.com/2011/01/python-threading-and-its-caveats/, Accessed: 2023-5-22, dic. de 2010.

[10] M. Liu, Z. Lu, W. Kuehn y A. Jantsch, "Inter-process communication using pipes in FPGA-based adaptive computing," en *2010 IEEE Computer Society Annual Symposium on VLSI*, Lixouri, Greece: IEEE, jul. de 2010.

[11] L. Kalita, *Socket Programming*. dirección: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=54dc78a981c2e3dcd2f9aacb59491831cf7d5401.

[12] J. Geerling, *Raspberry Pi 3 B+ Review and Performance Comparison*, https://www.jeffgeerling.com/blog/2018/raspberry-pi-3-b-review-and-performance-comparison, abr. de 2018.

[13] A. Pini, *Por qué y cómo usar la interfaz periférica serial para simplificar las conexiones entre distintos dispositivos*, feb. de 2019. dirección: https://www.digikey.com/es/articles/why-how-to-use-serial-peripheral-interface-simplify-connections-between-multiple-devices.

[14] 2021. dirección: https://www.analog.com/en/analog-dialogue/articles/i2c-communication-protocol-understanding-i2c-primer-pmbus-and-smbus.html.

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

[16] Ago. de 2022. dirección: https://www.spiceworks.com/tech/networking/articles/user-datagram-protocol-udp/.

[17] B. Nuttall, *GPIO Zero Documentation Release 1.6.2*. 2021. dirección: https://gpiozero.readthedocs.io/_/downloads/en/stable/pdf/.

[18] M. Dancuk, *Linux Commands All Users Should Know Ultimate List*, oct. de 2022. dirección: https://phoenixnap.com/kb/linux-commands.

[19] 2013. dirección: https://www.raspberrypi.com/documentation/computers/os.html#introduction.

CAPÍTULO 13

Anexos

Facultad de Ingeniería
Departamento de Ingeniería Electrónica, Mecatrónica y Biomédica

Luis Alberto Rivera

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 **`hilos`**, la 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. Se deben de correr los programas **L3_Hello.py** y **L3_World.py**. Esto se puede hacer desde la terminal usando la siguiente estructura de comando **python <nombre_del_archivo.py>.** Vale la pena resaltar el hecho que dado a que no se está corriendo un ejecutable, a la hora de listar los programas que están corriendo no se reflejará el nombre del archivo. En su lugar se enlistará el nombre del ejecutable de Python (y la respectiva versión de este). Esto se explica debido a la naturaleza de Python de ser un lenguaje interpretado, ya que lo que está corriendo es dicho interpretador, que lee el archivo y después decide con que recursos ejecutar las líneas del código.
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, en sistemas unix:
   python L3_Hello &
   En Windows se puede ejecutar de la siguiente manera:
   START /B python L3_Hello.py
   Eso permitirá seguir usando la consola, por lo que podrán ejecutar el segundo programa:
   python 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. Mientras los programas anteriores esténejecutándose, ingresen (en sistemas unix) el comando: **ps ax**
   A continuación, ingresen el comando: **ps ax | grep python**
   En caso se trabaje con Windows pueden hacer lo siguiente:
   tasklist

tasklist | more | findstr python

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). En caso se trabaje en Windows, ingresar lo siguiente **taskkill /F /PID <número de pid correspondiente>** **¿Qué ocurre?**

5. A continuación, estudien el programa **L3_Hilos_Ej1.py**.
   Atiendan las explicaciones del catedrático sobre este programa y el concepto de hilos POSIX (POSIX threads o pthreads).

6. Ahora estudien, compilen y ejecuten el programa **L3_Hilos_Ej2.py** en una de las consolas. **¿Cómo se compara lo observado aquí con lo observado al ejecutar los programas de los incisos 1 y 2?**
   En la otra consola, ingresen el comando **ps ax | grep python**. O en su defecto **tasklist | more | findstr python**

7. **¿Qué observan ahora? Tomen una captura de pantalla de la consola.**

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.py**. Para poder hacerlo deben de importar la librería portable_forks como se hace en la primer línea del programa. Para asegurarse de que esto funcione, se deben de asegurar que los archivos provistos correspondientes a esa librería se encuentren en el directorio de trabajo donde estén ejecutando. **Brevemente mencionen las similitudes que encuentrencon los programas de la Primera Parte.**

   El instructor explicará el funcionamiento de la función fork() y como la librería provista es capaz de usar esta funcionalidad tanto en sistemas UNIX y WINDOWS.

2. En una terminal, corran el programa. Sólo deben invocar el programa una vez:
   pyton 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 python**. O en su defecto **tasklist | more | findstr python**. **¿Qué observan ahora? ¿Cuántos procesos L2_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. Ejecuten los programas.

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 python**. O en su defecto **tasklist | more | findstr python** **¿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.py** Antes que termine la ejecución,corran **ps ax | grep python**. O en su defecto **tasklist | more | findstr python _** en otra consola. **¿Hace sentido lo que observan? Expliquen claramente. Tomen una captura de pantalla.**

2. **Modifiquen el programa** **L3_Hilos_Ej2.py** para que tenga tres hilos en lugar de dos. Para ello, deberán asignar a más variables el objeto de threading.Thread() de la librería de threading (esta es nativa de Python) Pueden crear nuevas funciones que usen distintos tipos de argumentos como los keyword arguments.
   a. El tercer hilo debe ejecutar el mismo código **My_Thread**, pero debe desplegar un mensaje distinto. Es decir, al llamar a la función My_Thread se le debe de ingresar un valor distinto al argumento de entrada.
   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 Python). Al llamar a **threading.Thread()** en el *main*, el primer argumento debe tener elnombre de su función nueva, los siguientes argumentos corresponderán al diseño de su función en caso sea necesario.

Evaluación:

Deberán subir a Canvas un documento (**.pdf**) con las respuestas a **todas las preguntas planteadas en la guía y las capturas de pantalla solicitadas**. Asegúrense de identificar/numerar claramente sus respuestas y resultados, según la parte/inciso de la pregunta correspondiente.
Además, deberán subir a Canvas **su código correspondiente al inciso 2 de la Tercera Parte**. Asegúrense de incluir su nombre, carné, curso y número de laboratorio como comentarios al inicio del archivo fuente.

La fecha/hora de límite de entrega aparecerá en Canvas.

Asistencia y trabajo en el lab:	20%
Reporte (.pdf subido a Canvas):	40%
Programa (archivo .c subido a Canvas)	40%

Universidad Del Valle de Guatemala **1**
Facultad de Ingeniería
Departamento de Ingeniería Electrónica, Mecatrónica y Biomédica
IE3059 – Electrónica Digital 3, Segundo Ciclo, 2023
Luis Alberto Rivera

Laboratorio 5: Conexión PC – RPi; Puertos de E/S de Uso General de la RPi (Programas en Python)

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 Python.

Duración: 1½ sesiones

Primera Parte: Conexión PC – RPi

Muchas veces es conveniente conectarse a dispositivos como la RPi desde otras computadoras. A continuación, aprenderá a hacerlo desde terminales y desde el programa VNC Viewer. Este último permite visualizar la interfaz gráfica de la RPi en la otra computadora. También aprenderá a transferir archivos entre dispositivos.

1. Arranque la RPi. Si la fecha y hora están mal, las configurará manualmente. Esto se puede hacer desde una terminal con el comando **date**, así:

   sudo date -s 'YYYY-MM-DD HH:MM:SS'

   Verifique que la fecha está bien ingresando **date** en la terminal. La hora también se ajustará en la esquina superior derecha de la interfaz gráfica, aunque eso puede tomar aprox. 1 minuto.

   Nota: Esto no debería ser necesario si la RPi se conecta a Internet a través de una red sin restricciones (la red UVG tiene algunas). Si prueba conectar la RPi en la red de su celular (usado como hotspot), la fecha y hora deberían corregirse automáticamente.

2. Conecte tanto su laptop como su RPi a la red inalámbrica **NETGEAR_iemtbmUVG** (*Router* que usaremos para el curso). La contraseña es: **iemtbmUVG**

   Nota: en su laptop, primero desconéctese de la red UVG (si estaba conectado a dicha red). Luego, al seleccionar la red NETGEAR_iemtbmUVG, marque la opción de conectarse automáticamente. Esto es para prevenir que su laptop se vuelva a conectar a la red UVG. Más adelante podrá "olvidar" la red NETGEAR_iemtbmUVG y reconectarse a la red UVG. Note que NETGEAR_iemtbmUVG NO da acceso a Internet.

Guía Laboratorio 5 (Página 2)

3. Averigüe la dirección IP de la RPi. Recuerde que lo puede hacer en una terminal con el comando **ifconfig** (busque la inet de la wlan0).

   Nota: La dirección IP no necesariamente será la misma cada vez que se conecte a la red, ya que la asignación no es estática. Es posible definir IPs estáticas, cambiando configuraciones en la red. Puede investigar esto y aplicarlo en la red de su casa, si lo desea. Más adelante verá la ventaja de hacer esto, aunque no es indispensable.

4. Abra una terminal de Cygwin en su laptop. Ingrese el comando **ipconfig**. Verifique que la dirección IP de la LAN inalámbrica Wi-Fi sea "similar" a la de de su RPi (busque la dirección IPv4). "Similar" se refiere a que los tres primeros números de la IP deben ser iguales, y sólo el último debe ser distinto.

5. Ahora se conectará a la RPi desde Cygwin. Esto se hace con el comando **ssh**, así:

   ssh pi@dir.ip.dela.RPi ejs.: ssh pi@10.0.0.5 o ssh pi@192.168.1.20

   Es posible que la primera vez que haga lo anterior le aparezca un mensaje preguntando si realmente desea hacer lo que está a punto de hacer. Ingrese **yes**. Ya no debería necesitar hacerlo nuevamente. Se le pedirá que ingrese la contraseña del usuario pi. Como se indicó en un laboratorio anterior, la contraseña es: 1. Note que no aparece nada al ingresar la contraseña.

6. Si todo salió bien, notará que la línea de comandos ahora se ve como cuando abrió una terminal en la RPi anteriormente. Pruebe comandos como **ls** y **cd**, y convénzase de que está conectado a la RPi. Desde esta terminal, se puede hacer todo lo que ha hecho en laboratorios anteriores que no requiera interfaz gráfica. En particular, se puede compilar y correr programas. Incluso se puede editar programas con algunos editores básicos como **vi** o **nano** (estos no los veremos en el curso, aunque puede investigarlos, si tiene curiosidad).

7. Para terminar la conexión entre la computadora y la RPi, puede usar tanto el comando **exit** como el comando **logout**. Pruebe cualquiera de los dos. Notará cómo la línea de comandos vuelve a verse como cuando abrió Cygwin.

8. Es posible establecer más de una conexión entre su computadora y la RPi. Abra otra terminal de Cygwin, y conéctese nuevamente a la RPi desde ambas terminales.

9. También es posible conectarse a través del PowerShell o el cmd de Windows. Abra una ventana de cada uno, y conéctese a la RPi. Se hace de la misma forma:

   ssh pi@dir.ip.dela.RPi

   Nota: la conexión también se puede hacer desde terminales de sistemas Linux o Mac.

10. Es posible transferir archivos entre la RPi y una PC (o entre distintas RPis) desde una terminal. Esto se puede hacer con el comando **scp**. La sintaxis es la siguiente:

    scp archivo(s)_a_transferir destino

En Cygwin (en su computadora, no conectado a la RPi), vaya a algún directorio que tenga varios archivos .txt o .c (por ejemplo, donde tenga los programas del lab 1 o del lab 3). Si no tiene ningún folder así, cree uno con un par de archivos de texto. Ingrese lo siguiente:

scp un_archivo.txt pi@dir.ip.dela.RPi:

Se le pedirá ingresar la contraseña del usuario pi. Luego de ingresarla, deberá ver una línea indicando que el archivo se transfirió.

Nota 1: luego de la dirección IP, siempre se debe escribir el caracter dos puntos ':'. Si no se escribe nada después de los dos puntos, el archivo se transfiere al directorio **/home/pi** en la RPi. Si desea transferir el archivo a otro directorio, debe ingresar la ruta después de los dos puntos. Por ejemplo, si quisiera transferir el archivo a un folder llamado Lab5, dentro del folder Documents (la ruta completa sería /home/pi/Documents/Lab5; el folder ya debe existir en la RPi), debería ingresar lo siguiente:

scp un_archivo.txt pi@dir.ip.dela.RPi:Documents/Lab5

Nota 2: si el nombre de su archivo tiene espacios en blanco, y/o el folder destino en la RPi tiene espacios en blanco, debe usar comillas simples y el caracter '\ ', así:

scp 'otro archivo.txt' 'pi@dir.ip.dela.RPi:ruta_al/folder\ con\ espacio'

Por cosas como lo anterior, es preferible evitar usar nombres de archivos o directorios con espacios en blanco.

Nota 3: se puede transferir cualquier tipo de archivos. No tienen que ser archivos de texto.

En la RPi, vaya al folder destino. Deberá ver el archivo transferido.

11. Se pueden transferir varios archivos a la vez. Todos los archivos que se escriban antes del último argumento (si existen), serán transferidos. Por ejemplo:

    scp archivo1 archivo2 archivo3 pi@dir.ip.dela.RPi:Documents/Lab5

    También es posible usar el caracter de comodín, '*', para transferir múltiples archivos a la vez. Por ejemplo, si quisiera enviar todos los archivos con extensión .c, podría hacer lo siguiente:

    scp *.c pi@dir.ip.dela.RPi:Documents/Lab5

    Nota: el comodín también funciona para otros comandos, no sólo para **scp**, y puede estar en cualquier parte del argumento de los comandos. Por ejemplo, si quisiera eliminar todos los archivos que comiencen con L1_, o mover todos los archivos que terminen con .txt de un folder a otro, puede ingresar lo siguiente, respectivamente:

    rm L1_*

    mv *.txt ruta/al/folder/destino

    Pruebe transferir todos los archivos .txt (o .c) de Cygwin a la RPi. Verifique que se hayan transferido exitosamente. Luego, en la RPi, borre todos los archivos que transfirió.

12. El comando **scp** también funciona en el PowerShell o en el cmd. Abra alguno de los dos, vaya a un directorio de su computadora donde tenga varios archivos, y pruebe transferirlos a la RPi. Pruebe transferir uno por uno, y pruebe transferir varios a la vez.

13. También es posible transferir archivos de la RPi a su computadora. Esto se puede hacer desde Cygwin (o PowerShell o cmd) en su computadora, invirtiendo el origen y el destino. Suponga que hay un archivo en la RPi en el directorio /home/pi/Documents que quiere transferir al directorio en el que se encuentra en la terminal de su computadora. Para hacer la transferencia, ingresaría:

    scp pi@dir.ip.dela.RPi:Documents/su_archivo .

    El punto al final significa "aquí". Si quisiera transferir a otro directorio, no a donde está actualmente, ingresaría:

    scp pi@dir.ip.dela.RPi:Documents/su_archivo ruta/al/directorio/destino

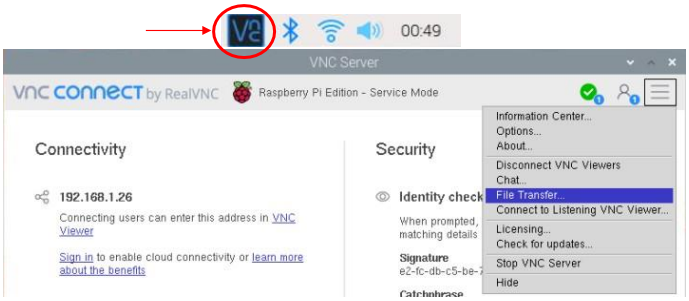
    Pruebe transferir algo desde la RPi a su computadora, desde Cygwin. Use el comando **ls** para verificar que efectivamente se transfirió el archivo. Luego, transfiera algo más, pero desde PowerShell o cmd. Verifique la transferencia. Recuerde que **ls** funciona en PowerShell, pero no en cmd. En lugar de **ls**, puede usar el comando **dir**.

14. Muchas veces es suficiente establecer conexiones a través de la terminal. Sin embargo, también es útil poder ver la interfaz gráfica de la RPi en nuestra computadora (o tableta, o incluso celular). Una forma de hacerlo es a través del programa VNC Viewer. La RPi necesita tener el VNC Server (el cual ya está instalado y activo en las RPis que le prestaron).
    En su computadora, abra VNC Viewer. En la parte superior, ingrese la dirección IP de la RPi (que es la dirección del servidor VNC al que se desea conectar). Si le aparece un mensaje de advertencia, acepte proseguir. Le deberá aparecer una ventana para ingresar el nombre de usuario y la contraseña. Ingrese pi y 1, respectivamente. Si todo está bien, le aparecerá una ventana con la interfaz gráfica de la RPi.

    Ya estando conectado, usted puede trabajar en la RPi usando el teclado y ratón de su computadora, y visualizando todo en su pantalla. Esto es muy útil, si no cuenta con monitor, otro teclado y otro ratón para conectarlo a la RPi. Lo único que necesita es que ambas computadoras estén en la misma red local (y que esta no tenga muchas restricciones), y conocer la dirección IP de la RPi.

15. Si coloca el puntero del ratón por la parte superior de la ventana del VNC Viewer, le aparecerá un menú como el que se muestra en la figura a continuación. Con uno de los íconos abrirá una herramienta para transferencia de archivos. Podrá seleccionar qué archivos quiere transferir de su computadora a la RPi, y dónde, en su computadora, se guardarán archivos copiados desde la RPi.

Para seleccionar archivos a transferir de la RPi a su computadora, y seleccionar dónde, en la RPi, se guardarán archivos transferidos desde su computadora, deberá hacer clic en el ícono del VNC Server. Busque la opción "File Transfer…" en el menú de la esquina superior derecha. Vea las figuras abajo.



Pruebe transferir archivos desde su computadora a la RPi, y vice-versa. Pruebe cambiar las ubicaciones de destino, tanto en la RPi como en su computadora.

16. Para finalizar la sesión del VNC Viewer, puede cerrar la ventana, o hacer clic en el último ícono del menú de la parte superior.

Todo lo anterior se puede hacer sin la necesidad de una red Wi-Fi. Basta usar un cable de red (Ethernet) para conectar la PC con la RPi.

17. Tome un cable de red y conéctelo al puerto Ethernet de la RPi y al de su computadora. Si su computadora no cuenta con puerto Ethernet, deberá observar lo que hace algún compañero, siempre manteniendo el debido distanciamiento.

    Nota: Con un adaptador USB – Ethernet se puede realizar la conexión y todo lo descrito a continuación. **No es necesario que compre uno.**

18. Luego de unos instantes después de conectar el cable, averigüe la dirección IP de la RPi asociada a Ethernet. Use el comando **ifconfig** y busque la inet de la eth0. Esta dirección también debe aparecer en el ícono de red de la esquina superior derecha de la interfaz gráfica de la RPi.

    Nota: Esta dirección IP de la RPi sí es muy probable que sea la misma cada vez que se conecte a su computadora, así que apúntela. Esto es muy conveniente, ya que podrá conectarse a la RPi sin tener que averiguar la IP cada vez.

19. Pruebe conectarse desde una terminal de Cygwin usando la IP de Ethernet. Note que puede mantener las conexiones anteriores (Wi-Fi), o no. No importa. De hecho, no es necesario que el Wi-Fi esté habilitado en la RPi para conectarse por Ethernet.

Nota: La IPv4 de Ethernet de su computadora también la puede encontrar usando **ipconfig** desde Cygwin o PowerShell. Los primeros dos números deberían coincidir con los de la IP de la RPi.

20. Pruebe conectarse desde PowerShell y/o cmd.
21. Pruebe conectarse por medio de VNC Viewer. En este caso, sí es mejor cerrar la sesión que tenía por Wi-Fi, por la cantidad de datos que se transfieren con las interfaces gráficas.

Tercera Parte: Manejo de Puertos GPIO con Programas Escritos en Python

A continuación, deberá crear programas en Python que manipulen los puertos GPIO. Tendrá la libertadde crearlos/editarlos usando KWrite y correrlos desde la terminal o su IDE de preferencia.

1. Encendido y Apagado de LEDs: Escriba un programa en Python 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 las funciones **random.uniform** de la librería random.

   Considere las siguientes clases de gpiozero:
   - LED(int)

   Nota: Necesitará incluir el encabezado **import gpiozero import LED** en su código.

   Deberá mostrar el funcionamiento del programa al instructor o auxiliar.

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 Python que genere una señal cuadrada para activar una bocina. La señal no debe empezar de inmediato al correr el programa. Deberáesperar 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).
   Ayuda: Use hilos para poder realizar cada una de las partes de la actividad, especialmente para los inputs. De esta manera puede usar la función **input()** sin que se bloquee todo el código.

   Además de las clases de gpiozero mencionadas en el inciso 1 de la Tercera Parte, considere la clase siguientes (investigue el método wait_for_press()):
   - Button()

   También deberá mostrar el funcionamiento de este programa al instructor o auxiliar.

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.

Al terminar de usar la RPi, recuerde apagarla correctamente. Pruebe apagarla desde una terminal de Cygwin o PowerShell, estando conectado a la RPi.

Evaluación:

Además de la demostración de sus programas de la Tercera Parte, deberá subir a Canvas **los archivos fuente (archivos .c)**. Asegúrese de incluir su nombre, carné, curso y número de laboratorio, como comentarios.

Trabajo en el laboratorio:	40%
Demostración:	30%
Programas (.c):	30%

Fecha límite de entrega: consulte la tarea de Canvas.

Universidad Del Valle de Guatemala **1**
Facultad de Ingeniería
Departamento de Ingeniería Electrónica, Mecatrónica y Biomédica
IE3059 – Electrónica Digital 3, Segundo Ciclo, 2022
Luis Alberto Rivera

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.

Duración: 1½ sesiones

Antes de comenzar los experimentos descritos abajo, presten atención a la "IE3059 - Laboratorio 6, presentación". Estudien los archivos Lab6_files_y_strings.c y Lab6_Timer_functions.c (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 adicionales al hilo principal (*main*), y utilizará un *buffer* común a los hilos. El *buffer* será 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 hilo adicionales se encargarán de leer los archivos, una línea a la vez (el primerhilo abrirá y leerá el archivo **Lab6_primero.txt** y el segundo hilo 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 hilo 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 arreglosí 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*.

Guía Laboratorio 6 (Página 2)

- Todos los hilos 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 *pthreads* 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.

- 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 apropiados en sus programas según las librerías a utilizar.

Evaluación:

Deberán subir a Canvas un **reporte (.pdf), el cual deberá incluir los tiempos que usaron, así como la captura de pantalla. Incluyan también una breve discusión de lo realizado, y mencionen ventajas y desventajas del método de sincronización utilizado en esta práctica**. Asegúrense de que el documento esté bien identificado (nombre, carné, curso, número de laboratorio, fecha).
También **deberán subir su código final a Canvas (archivo.py)**. Recuerden incluir sus datos como comentarios al inicio de su código.

Asistencia y trabajo en el lab:	40%
Reporte (.pdf):	30%
Programa (.c):	30%

Fecha límite de entrega: revisar la tarea "Lab 6" en Canvas.

Facultad de Ingeniería
Departamento de Ingeniería Electrónica, Mecatrónica y Biomédica
IE3059 – Electrónica Digital 3, Segundo Ciclo, 2024
Carlos Cuellar

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.

Duración: 2 sesiones

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. 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 0, y cuando no esté presionado, en el puerto se lea el valor de 1** (consideren un resistor de *pull up*). Pueden verificar que su conexión es correcta usando los comandos para una terminal de la herramienta **gpio**.

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 **`sleep`**. Sugerencia, separar todas los escenarios posibles en funciones y que el hilo principal sol use una serie de condicionales para llamar a cada función cuando sea pertinente.

¿Qué limitaciones tiene este método?
Deberán mostrar el funcionamiento del programa al instructor o al auxiliar.

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

Investigue la incompatibilidad de priorizar los hilos como se hace en C que esta acción representa en Python.

Asegúrense de reportar sus observaciones y explicar/discutir sus resultados en el reporte.
Deberán mostrar el funcionamiento del programa al instructor o al auxiliar. Se les pedirá que muestren dos o tres de las configuraciones.

Evaluación:

Además de la **demostración de sus programas**, deberán subir a Canvas el **reporte (.pdf). Incluyan sus resultados y observaciones, así como las respuestas a las preguntas planteadas en la guía.** Asegúrense de que el documento esté bien identificado (nombre, carné, etc.). **También deberán subir sus códigos (.c)**. Recuerden incluir su nombre, carné, número de laboratorio, curso, etc. como comentarios.

Trabajo en el lab: 20%, Demostración: 20%, Reporte (.pdf): 30%, Programas (.py): 30%

Fecha límite de entrega: revisar tarea de Canvas.

Universidad Del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ingeniería Electrónica, Mecatrónica y Biomédica
IE3059 – Electrónica Digital 3, Segundo Ciclo, 2024
Carlos Cuellar

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*).

Duración: 1 semanas

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.

Guía Laboratorio 8 (Página 2)

Especificaciones:

1.) Los votos son números enteros aleatorios en el intervalo [1, 15].

2.) Pueden usar el programa **client.py** como punto de partida para sus programas.

3.) 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.**

Pueden encontrar el programa cliente en Canvas (*Módulos > Modelos de Dispositivo Local y Remoto > Laboratorio*). Este programa solo puede mandar los mensajes "QUIEN ES" y "VOTE" al broadcast del puerto. Será el programa base para evaluar el rendimiento de todos los clientes desarrollados por la clase.

Inicialmente, pueden probar sus programas corriendo un servidor en su máquina local, otro en una máquina virtual con un adaptador de red puenteado y el cliente desde la RPi. La cantidad de maneras en las que pueden hacer pruebas es ilimitada. **En la evaluación final de este laboratorio, el instructor correrá el cliente y todos deberán correr sus servidores al mismo tiempo**.

Funciones útiles:

Favor leer la documentación de la librería socket de Python. Esta es una librería nativa, no implica una instalación por medio de pip. Funciones útiles:

- `socket.gethostbyname()`
- `socket.socket()`
- `socket.bind()`
- `socket.setscockopt()`

Evaluación:

El reporte consistirá en una discusión de sus observaciones y experiencias en este laboratorio. Mencionen los problemas y mayores dificultades que hayan tenido para completar su programa, y los aspectos que más les hayan llamado la atención. El reporte debe ser un archivo **.pdf**. Asegúrense de que el documento esté bien identificado (nombre, carné, etc.). También deberán subir su código a Canvas. Recuerden incluir su nombre, carné, número de laboratorio, curso, etc. como comentarios al inicio de su código.

Asistencia y trabajo en el lab:	20%
Funcionamiento de su programa:	40%
Reporte (**.pdf**):	20%
Programa (**.py**):	20%
Determinación automática de IP:	10% extra

Universidad Del Valle de Guatemala **1**
Facultad de Ingeniería
Departamento de Ingeniería Electrónica, Mecatrónica y Biomédica
IE3059 – Electrónica Digital 3, Segundo Ciclo, 2024
Carlos Cuellar

Proyecto de Curso

Sistema SCADA

Integración entre sistemas embebidos y bases de datos.

Objetivos

En este proyecto, los estudiantes deben de poder establecer un sistema de monitoreo tipo SCADA, por medio de todas las herramientas que se han visto a través del curso:

- Como mínimo el sistema debe de poder tener dos RTU's.
- Mandatorio el monitoreo de 3 entradas analógicas, 3 entradas digitales, 3 salidas digitales.
- Levantar una base de datos (MySQL, PostgreSQL, SQLServer…) que registre los eventos del sistema SCADA de manera periódica.
- Realizar una herramienta de visualización para la progresión de los datos recopilados.

Duración: 3 semanas

Procedimiento

El sistema SCADA (Supervisory control and data acquisition), consiste en la comunicación entre estaciones remotas (RTU), hacia una estación supervisora. Dicha estación es la encargada de gestionar la operación de las RTU's según la información provista anteriormente por estas, al igual que por la interacción humana. A partir de la estación supervisora se pueden limitar las funcionalidades de las RTU'S.

Para los objetivos planteados, el control de la base de datos, pasara a estar a disposición de la base de datos. Siendo esta el único elemento capaz de poder tener acceso a la data de todos los RTU's. El sistema escada tiene que ser escalable, ya que por la utilización de base de datos, se espera que se pudean agregar RTU's de manera eficiente y rápida.

La herramienta de visualización es libre hacia el diseño de cada estudiante. Sin embargo se recomienda utilizar PowerBI para la realización de un dashboard bastante profesional. También podría existir la posibilidad de realizar la herramienta de visualización por medio de una interfaz gráfica usando librerías de Python como PyQT. Sin embargo, cualquier sea la solución seleccionada, le herramienta de visualización tiene que estar estrictamente alimentada por la base de datos levantada.

El estudiante tendrá a su libertad tener la opción de elegir cualquier tipo de sistema de base de datos. Sin embargo se recomienda fuertemente el uso de SQLServer, para poder aprovechar la gran gama de herramientas que está tiene a su disposición en el entorno de sistemas Widnows (Como lo es SQL Server Manager, SQL Server Integration Services, entre otros) que pueden ser de suma utilidad para el desarrollo del proyecto.

Guía Proyecto (Página 2)

Especificaciones:

1.) De las 3 entradas analógicas usar como mínimo algún tipo de sensor analógico (temperatura, fotoresistor, humedad…).

2.) Pueden usar la librería pySQLino como plataforma para integrar sistemas embebidos a la base de datos.

3.) 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.**

Pueden encontrar la librería recomendada dentro de un repositorio en el siguiente ***enlace:*** https://github.com/Cue19275/pySQLino.git/. Esta librería usa como base ArduinoUno para la interacción con circuitos y realiza conectividad automática a bases de datos SQLServer.

.

En la evaluación final de este proyecto, el instructor correrá el sistema con un RTU y se debe de demostrar en tiempo real la conectividad de este hacia la base de datos, de igual manera se pondrá a prueba la herramienta de visualización y su enlace hacia la base de datos. Se evaluará el diseño de la base de datos, en función en que las opciones de visualización que tengan presenten la información de manera útil, no solo en tablas por ejemplo.

Guía Proyecto (Página 3)

Evaluación:

El reporte consistirá en una discusión de sus observaciones y experiencias en este proyecto. Mencionen los problemas y mayores dificultades que hayan tenido para completar su programa, y los aspectos que más les hayan llamado la atención. Adjuntar al reporte TODOS los códigos realizados, debidamente documentados. El reporte también debe de tener una sección donde se indique paso a paso como levantar el sistema a SCADA planteado por el estudiante. El reporte debe ser un archivo **.pdf**. Asegúrense de que el documento esté bien identificado (nombre, carné, etc.). También deberán subir su código a Canvas. Recuerden incluir su nombre, carné, número de laboratorio, curso, etc. como comentarios al inicio de su código.

Herramienta de visualización:	20%
Funcionamiento de su programa:	40%
Reporte (**.pdf**):	20%
Programa (**.py**):	20%
Determinación automática de IP:	10% extra