# EX2

## Teoría

### Multiprocesos en Python

La programación multiproceso en Python implica la ejecución simultánea de múltiples procesos para lograr paralelismo y mejorar el rendimiento de una aplicación.

La programación multiproceso en Python permite lograr el paralelismo real al ejecutar múltiples procesos en múltiples núcleos de CPU o en diferentes máquinas. Esto puede acelerar tareas intensivas en CPU y procesamiento en lotes.

### Multihilos en Python

La programación multihilo en Python implica ejecutar varias partes de un programa de forma concurrente, lo que puede mejorar la eficiencia y la capacidad de respuesta.

Un hilo es una unidad de ejecución ligera dentro de un proceso. En Python, se pueden crear hilos utilizando el módulo `threading` de la biblioteca estándar. Los hilos comparten la misma memoria y recursos dentro de un proceso y se ejecutan de forma concurrente.

#### GIL (Global Interpreter Lock)

Es un concepto importante en Python. El GIL es un mecanismo de bloqueo utilizado por la implementación estándar de CPython (la implementación de referencia de Python) para garantizar que solo un hilo de ejecución pueda ejecutar código de Python a la vez.

### Concurrencia en Python

La concurrencia se refiere a la capacidad de ejecutar varias tareas aparentemente al mismo tiempo, mientras que el paralelismo implica la ejecución real simultánea de múltiples tareas. En Python, debido al Global Interpreter Lock (GIL), que limita la ejecución de hilos de Python en la implementación estándar (CPython), el paralelismo puro no es posible en hilos de Python. Sin embargo, la concurrencia aún se puede lograr para ciertos tipos de tareas, como aquellas que implican operaciones de E/S bloqueantes.

### Race Conditions en Python

Una condición de carrera (race condition) en Python ocurre cuando múltiples hilos de ejecución acceden a un recurso compartido y tratan de modificarlo simultáneamente, lo que puede llevar a resultados inesperados o incorrectos.

Esto puede suceder cuando el orden de ejecución de los hilos no está determinado y depende de factores como la planificación del sistema operativo.

#### Ejemplo de Race Condition

Cada hilo intenta acceder a la misma variable global `counter` y todas están intentando incrementarla. Por lo que la variable no está protegida (thread-safe).

```
import concurrent.futures

counter = 0

def increment_counter(fake_value):
  global counter
  for _ in range(100):
  counter += 1

if _name_ == "_main_":
  fake_data = [x for x in range(5000)]
```

```
  counter = 0
  with concurrent.futures.ThreadPoolExecutor(max_workers=5000) as executor:
    executor.map(increment_counter, fake_data)
```

## Web Scraper

El multiprocess al crear los procesos, la creación de procesos es lenta, porque se usa un interprete de Python por cada proceso (por eso suele lenta y no tiene algún beneficio). Asyncio al usar un solo hilo de ejecución genera un tiempo más rápido (evita el cambio de hilos).

Si se quiere instalar el memory profiler.

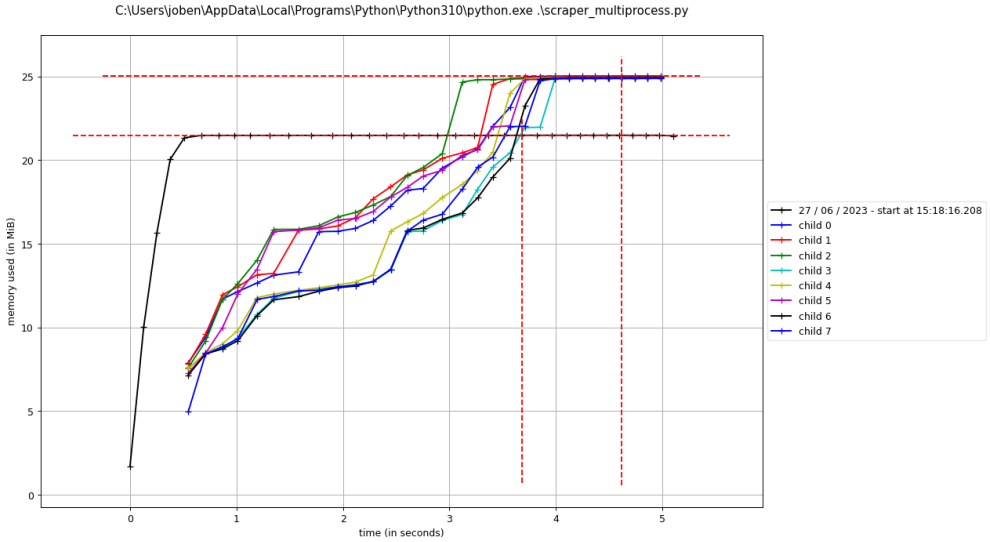
```
pip install memory_profiler
```

Si se quiere ver gráficamente información del proceso.

```
python -m mprof run scraper_tipo.py
python -m mprof plot
```

Si se quieren tomar en cuenta todos los procesos (se coloca `-M` en el último).

```
python -m mprof run -M scraper_tipo.py
python -m mprof plot
```




## Scraper sincrono (sync)

```
import requests
import time


def download_site(url, session):
    with session.get(url) as response:
        pass
        # print(f"Lei {len(response.content)} bytes de {url}")


def download_all_sites(sites):
    with requests.Session() as session:
        for url in sites:
            download_site(url, session)
```

```
if __name__ == "__main__":
    sites = [
        "https://www.jython.org",
        "http://olympus.realpython.org/dice"
    ] * 80

    inicio = time.perf_counter()
    download_all_sites(sites)
    fin = time.perf_counter()

    print(f"Descargado {len(sites)} de manera sincrona en {fin - inicio} segundos")
```

## Scraper asíncrono (async)

```
import time
import asyncio
import aiohttp

async def download_site(url, session):
    async with session.get(url) as response:
        print(f"Lei {len(await response.read())} bytes de {url}")


async def download_all_sites(sites):
    async with aiohttp.ClientSession() as session:
        tasks = list()
        for url in sites:
            task = asyncio.ensure_future(download_site(url, session))
            tasks.append(task)
    await asyncio.gather(*tasks, return_exceptions=True)


if __name__ == "__main__":
    sites = [
        "https://www.jython.org",
        "http://olympus.realpython.org/dice"
    ] * 80

    loop = asyncio.new_event_loop()
    asyncio.set_event_loop(loop)
    inicio = time.perf_counter()
    loop.run_until_complete(download_all_sites(sites))
    fin = time.perf_counter()

    print(f"Descargado {len(sites)} de manera asincrona en {fin - inicio} segundos")
```

## Scraper multihilo (multithread)

```
import concurrent.futures
import requests
import threading
import time

thread_local = threading.local()

def get_session():
    if not hasattr(thread_local, "session"):
        thread_local.session = requests.Session()
    return thread_local.session


def download_site(url):
    session = get_session()
    with session.get(url) as response:
        pass
        # print(f"Lei {len(response.content)} bytes de {url}")


def download_all_sites(sites):
    with concurrent.futures.ThreadPoolExecutor(max_workers=8) as executor:
        executor.map(download_site, sites)


if __name__ == "__main__":
    sites = [
        "https://www.jython.org",
        "http://olympus.realpython.org/dice"
    ] * 80
```

```
    inicio = time.perf_counter()
    download_all_sites(sites)
    fin = time.perf_counter()

    print(f"Descargado {len(sites)} de manera sincrona en {fin - inicio} segundos")
```

### Scraper multiproceso (multiprocess)

```
import requests
from multiprocessing import Pool, cpu_count
import time

session = None

def set_global_session():
    global session
    if not session:
        session = requests.Session()


def download_site(url):
    with session.get(url) as response:
        pass
        # print(f"Lei {len(response.content)} bytes de {url}")


def download_all_sites(sites):
    with Pool(processes=cpu_count(), initializer=set_global_session) as pool:
        pool.map(download_site, sites)


if __name__ == "__main__":
    sites = [
        "https://www.jython.org",
        "http://olympus.realpython.org/dice"
    ] * 80

    inicio = time.perf_counter()
    download_all_sites(sites)
    fin = time.perf_counter()

    print(f"Descargado {len(sites)} de manera sincrona en {fin - inicio} segundos")
```

# Preguntas y problemas

| Concepto | Fórmula | Explicación |
|---|---|---|
| Velocidad de transferencia | Velocidad = Tamaño / Tiempo | Esta fórmula calcula la velocidad de transferencia dividiendo el tamaño de los datos transferidos entre el tiempo requerido. |
| Tiempo de transferencia | Tiempo = Tamaño / Velocidad | Esta fórmula calcula el tiempo requerido para realizar una transferencia de datos, dividiendo el tamaño de los datos entre la velocidad de transferencia. |
| Porcentaje de tiempo de procesador utilizado para E/S | Porcentaje = (Tiempo de E/S / Tiempo total) * 100 | Esta fórmula calcula el porcentaje de tiempo de procesador utilizado para realizar operaciones de E/S, dividiendo el tiempo de E/S entre el tiempo total y multiplicándolo por 100. |
| Ciclos de reloj necesarios para una transferencia DMA | Ciclos de reloj = (Transferencias DMA * Tamaño por transferencia) + (Transferencias DMA * Ciclos de reloj de configuración) | Esta fórmula calcula el número total de ciclos de reloj necesarios para realizar una transferencia DMA, considerando el tamaño de los datos transferidos y los ciclos de reloj requeridos para la configuración. |

| Concepto | Fórmula | Unidades | Ejemplo de problema |
|---|---|---|---|
| Velocidad de transferencia | Velocidad = Tamaño / Tiempo | Tamaño: bytes, Tiempo: segundos | Si se requiere transferir un archivo de 2 MB en 10 segundos, ¿cuál es la velocidad de transferencia en MB/s? |
| Tiempo de transferencia | Tiempo = Tamaño / Velocidad | Tamaño: bytes, Velocidad: bytes/segundo | Si se transfieren 1000 bytes a una velocidad de 200 bytes/segundo, ¿cuánto tiempo tomará la transferencia? |
| Porcentaje de tiempo de procesador utilizado para E/S | Porcentaje = (Tiempo de E/S / Tiempo total) * 100 | Tiempo: segundos | Si durante un período de 1 minuto se utilizan 10 segundos para E/S, ¿cuál es el porcentaje de tiempo de procesador utilizado para E/S? |
| Ciclos de reloj necesarios para una transferencia DMA | Ciclos de reloj = (Transferencias DMA * Tamaño por transferencia) + (Transferencias DMA * Ciclos de reloj de configuración) | Tamaño por transferencia: bytes, Ciclos de reloj: número de ciclos | Si se requiere transferir un bloque de datos de 512 bytes utilizando DMA con 8 transferencias y 5 ciclos de reloj de configuración, ¿cuántos ciclos de reloj se necesitan en total? |

## Preguntas de revisión

1. Enumera tres clasificaciones amplias de dispositivos externos o periféricos.
   - Dispositivos de entrada.
   - Dispositivos de salida.
   - Dispositivos de entrada/salida.
2. ¿Qué es el Alfabeto de Referencia Internacional?
   - El Alfabeto de Referencia Internacional (International Reference Alphabet) es un conjunto de símbolos y códigos utilizados para representar caracteres y símbolos en sistemas de comunicación y dispositivos de entrada/salida.
3. ¿Cuáles son las funciones principales de un módulo de E/S?
   - Las funciones principales de un módulo de E/S incluyen la interfaz entre el bus del sistema y los dispositivos periféricos, la gestión del flujo de datos entre el procesador y los dispositivos periféricos, y el control y monitoreo de las operaciones de E/S.
4. Enumera y define brevemente tres técnicas para realizar E/S.
   - E/S programada: El procesador realiza las transferencias de datos de forma secuencial utilizando instrucciones de E/S específicas.
   - E/S por interrupción: El procesador responde a las solicitudes de E/S mediante interrupciones generadas por los dispositivos periféricos.
   - Acceso directo a memoria (DMA): El controlador DMA toma el control del bus del sistema y realiza transferencias de datos directamente entre los dispositivos periféricos y la memoria principal sin la intervención del procesador.
5. ¿Cuál es la diferencia entre E/S mapeada en memoria y E/S aislada?
   - En la E/S mapeada en memoria (memory-mapped I/O), los dispositivos periféricos se mapean en la memoria principal y se acceden utilizando instrucciones de lectura/escritura de memoria. En la E/S aislada (isolated I/O), se utilizan instrucciones de E/S dedicadas para acceder a los dispositivos periféricos que tienen direcciones de E/S separadas.
6. Cuando ocurre una interrupción de dispositivo, ¿cómo determina el procesador qué dispositivo emitió la interrupción?
   - El procesador utiliza mecanismos de identificación de interrupciones, como las líneas de interrupción específicas asignadas a cada dispositivo, para determinar qué dispositivo emitió la interrupción. Cada dispositivo tiene su propia línea de interrupción y se activa cuando se produce una interrupción.
7. Cuando un módulo DMA toma el control del bus y mientras lo retiene, ¿qué hace el procesador?

- Cuando un módulo DMA toma el control del bus, el procesador se mantiene inactivo y espera hasta que el módulo DMA complete la transferencia de datos. Durante este tiempo, el procesador puede aprovechar para realizar otras tareas o esperar hasta que se libere el bus.

## Problemas

1. En un microprocesador típico, se utiliza una dirección de E/S distinta para referirse a los registros de datos de E/S y una dirección distinta para los registros de control y estado en un controlador de E/S para un dispositivo determinado. Estos registros se conocen como "puertos". En el Intel 8088, se utilizan dos formatos de instrucción de E/S. En un formato, el opcode de 8 bits especifica una operación de E/S, seguido de una dirección de puerto de 8 bits. Otros opcodes de E/S implican que la dirección de puerto está en el registro DX de 16 bits. ¿Cuántos puertos puede direccionar el 8088 en cada modo de direccionamiento de E/S?
   - En el formato de instrucción de E/S en el que se especifica una dirección de puerto de 8 bits, el 8088 puede direccionar 2^8 = 256 puertos. En el formato de instrucción de E/S en el que la dirección de puerto está en el registro DX de 16 bits, el 8088 puede direccionar 2^16 = 65536 puertos.
2. Un formato de instrucción similar se utiliza en la familia de microprocesadores Zilog Z8000. En este caso, existe una capacidad de direccionamiento directo de puerto, en la cual una dirección de puerto de 16 bits es parte de la instrucción, y una capacidad de direccionamiento indirecto de puerto, en la cual la instrucción referencia uno de los registros de propósito general de 16 bits, que contiene la dirección del puerto. ¿Cuántos puertos puede direccionar el Z8000 en cada modo de direccionamiento de E/S?
   - En el modo de direccionamiento directo de puerto, el Z8000 puede direccionar 2^16 = 65536 puertos. En el modo de direccionamiento indirecto de puerto, el Z8000 puede direccionar la misma cantidad de puertos, ya que la dirección del puerto se almacena en uno de los registros de propósito general de 16 bits.
3. El Z8000 también incluye una capacidad de transferencia de bloque de E/S que, a diferencia del DMA, está bajo el control directo del procesador. Las instrucciones de transferencia de bloque especifican un registro de dirección de puerto (Rp), un registro de conteo (Rc) y un registro de destino (Rd). Rd contiene la dirección de la memoria principal donde se almacenará el primer byte leído del puerto de entrada. Rc puede ser cualquiera de los registros de propósito general de 16 bits. ¿Qué tamaño máximo de bloque de datos se puede transferir?
   - El tamaño máximo de bloque de datos que se puede transferir en el Z8000 está determinado por el valor máximo que se puede almacenar en el registro de conteo (Rc), que es de 16 bits. Por lo tanto, se puede transferir un bloque de datos de hasta 2^16 bytes (64 KB).
4. Para calcular el aumento en velocidad con la instrucción de transferencia de bloque, restamos el número de ciclos de reloj requeridos por la instrucción no bloqueante al número de ciclos de reloj requeridos por la instrucción de transferencia de bloque.
   - Número de ciclos de reloj ahorrados = Ciclos de reloj no bloqueantes - Ciclos de reloj de transferencia de bloque.
   - Número de ciclos de reloj ahorrados = 20 - 5 = 15 ciclos de reloj.
   - La instrucción de transferencia de bloque ahorra 15 ciclos de reloj en comparación con la instrucción no bloqueante.
5. Un sistema se basa en un microprocesador de 8 bits y tiene dos dispositivos de entrada/salida (E/S). Los controladores de E/S para este sistema utilizan registros de control y estado separados. Ambos dispositivos manejan datos de a 1 byte a la vez. El primer dispositivo tiene dos líneas de estado y tres líneas de control. El segundo dispositivo tiene tres líneas de estado y cuatro líneas de control.
   a. ¿Cuántos registros de módulo de control de E/S de 8 bits necesitamos para la lectura de estado y el control de cada dispositivo?
      - Para el primer dispositivo que tiene 2 líneas de estado y 3 líneas de control, necesitamos 2 registros de control y 2 registros de estado, ya que ambos dispositivos manejan datos de 1 byte a la vez.
   b. ¿Cuál es el número total de registros de módulo de control necesarios dado que el primer dispositivo es solo de salida?
      - Dado que el primer dispositivo es solo de salida, solo necesitamos registros de control, por lo que necesitamos 2 registros de control en total.
   c. ¿Cuántas direcciones distintas se necesitan para controlar los dos dispositivos?

- Dado que cada dispositivo requiere 2 registros de control, necesitamos un total de 2 * 2 = 4 registros de control para controlar los dos dispositivos.

6. En la E/S programada, la Figura 7.5 indica que el procesador queda atrapado en un bucle de espera realizando verificaciones de estado de un dispositivo de E/S. Para aumentar la eficiencia, el software de E/S podría estar escrito de manera que el procesador verifique periódicamente el estado del dispositivo. Si el dispositivo no está listo, el procesador puede pasar a otras tareas. Después de un intervalo de tiempo determinado, el procesador vuelve a verificar el estado nuevamente.
   a. Si se escanea el estado de la impresora cada 200 ms y la impresora opera a 10 cps (caracteres por segundo), podemos calcular el número de caracteres impresos durante ese intervalo de tiempo:
      - Número de caracteres impresos = Velocidad de la impresora * Tiempo.
      - Número de caracteres impresos = 10 cps * 0.2 segundos = 2 caracteres.
      - En este caso, se imprimirían 2 caracteres durante el intervalo de 200 ms.
   b. Dado que el tiempo entre dos pulsaciones de tecla consecutivas puede ser tan corto como 60 ms y el promedio de entrada de caracteres es de 10 cps, podemos calcular la frecuencia de escaneo requerida para el teclado:
      - Frecuencia de escaneo = 1 / (Tiempo entre pulsaciones de tecla).
      - Frecuencia de escaneo = 1 / 0.06 segundos = 16.67 escaneos por segundo.
      - Por lo tanto, el teclado debe ser escaneado aproximadamente 16.67 veces por segundo para manejar la entrada de caracteres en promedio.
7. Para calcular el tiempo que lleva escanear y atender al dispositivo, multiplicamos el tiempo de un ciclo de reloj por el número de ciclos requeridos para escanear el estado del dispositivo.
   - Tiempo de escaneo y servicio = Ciclos de reloj * Tiempo de un ciclo de reloj.
   - Tiempo de escaneo y servicio = 20 ms * (1 segundo / 8 MHz) = 2.5 µs.
   - El tiempo necesario para escanear y atender al dispositivo es de 2.5 µs.
8. Ventajas de la E/S mapeada en memoria en comparación con la E/S aislada:
   - Mayor simplicidad en el diseño del sistema debido a que utiliza direcciones de memoria estándar para acceder a los dispositivos periféricos.
   - Permite una comunicación más rápida y eficiente entre el procesador y los dispositivos periféricos, ya que no requiere instrucciones especiales de E/S.
   - Desventajas de la E/S mapeada en memoria en comparación con la E/S aislada:
     - Puede haber conflictos de acceso a la memoria si tanto el procesador como los dispositivos periféricos intentan acceder a la misma dirección de memoria al mismo tiempo.
     - El direccionamiento y acceso a los dispositivos periféricos pueden ser más complicados debido a la necesidad de considerar la asignación de direcciones de memoria.
9. 
   a. Para calcular el número de veces que se revisará el teclado en un período de 8 horas, primero debemos convertir el intervalo de tiempo de 8 horas a milisegundos:
      - Tiempo total = 8 horas * 60 minutos/hora * 60 segundos/minuto * 1000 ms/segundo = 28,800,000 ms.
      - Número de veces que se revisará el teclado = Tiempo total / Tiempo de escaneo.
      - Número de veces que se revisará el teclado = 28,800,000 ms / 100 ms = 288,000 veces.
      - El teclado se revisará aproximadamente 288,000 veces en un período de 8 horas.
   b. Si se utiliza la E/S por interrupción, el número de visitas del procesador al teclado se reducirá a cada vez que se produzca una interrupción, es decir, cada vez que se presione una tecla. Por lo tanto, el número de visitas del procesador se reducirá a la cantidad de teclas ingresadas durante el período de 8 horas, que es de 60 comandos en promedio. El uso de la E/S por interrupción reducirá las visitas del procesador en un factor de 60 veces.

## Problemas de E/S y DMA (William Stallings)

Enunciados de problemas relacionados con E/S (Entrada/Salida) y DMA (Acceso Directo a Memoria) que podrías encontrar al final del capítulo 7 del libro "Computer Organization and Architecture" de William Stallings, junto con sus respectivas soluciones:

1. **E/S de transferencia programada**:
   En un sistema de E/S de transferencia programada, se tiene un procesador con una velocidad de ejecución de 2 GHz y una instrucción de transferencia de datos que tarda 50 ciclos de reloj. Si se requiere transferir 1000 bytes de datos, ¿cuánto tiempo tomará la transferencia completa de E/S?
   - La instrucción de transferencia de datos toma 50 ciclos de reloj. Dado que la velocidad de ejecución del procesador es de 2 GHz, se ejecutan 2 mil millones de ciclos de reloj por segundo. Por lo tanto, el tiempo necesario para transferir 1000 bytes de datos será:
     - Tiempo = (50 ciclos de reloj / 2 mil millones de ciclos por segundo) * 1000 bytes = 25 nanosegundos.
2. **E/S de interrupción**:
   En un sistema de E/S de interrupción, se tiene un procesador que tarda 100 ciclos de reloj en manejar una interrupción. Si se produce una interrupción cada 5000 ciclos de reloj, ¿cuál es el porcentaje de tiempo de procesador utilizado para manejar las interrupciones?
   - El procesador tarda 100 ciclos de reloj en manejar una interrupción, y se produce una interrupción cada 5000 ciclos de reloj. Por lo tanto, el porcentaje de tiempo de procesador utilizado para manejar las interrupciones será:
     - Porcentaje = (100 ciclos de reloj / 5000 ciclos de reloj) * 100 = 2%
3. **DMA con memoria compartida**:
   En un sistema con DMA y memoria compartida, se requiere transferir un bloque de datos de 512 bytes desde un dispositivo de E/S a la memoria principal. Si el controlador DMA puede transferir 64 bytes a la vez y tarda 10 ciclos de reloj para configurarse antes de cada transferencia, ¿cuántos ciclos de reloj se necesitan para completar la transferencia completa?
   - El controlador DMA puede transferir 64 bytes a la vez y tarda 10 ciclos de reloj para configurarse antes de cada transferencia. Dado que el bloque de datos tiene 512 bytes, se necesitarán 8 transferencias DMA para completar la transferencia completa. Además, cada transferencia DMA tomará 10 ciclos de reloj para configurarse. Por lo tanto, el número total de ciclos de reloj necesarios será:
     - Ciclos de reloj = (8 transferencias DMA * 64 bytes por transferencia) + (8 transferencias DMA * 10 ciclos de reloj por configuración) = 512 ciclos de reloj + 80 ciclos de reloj = 592 ciclos de reloj.

## Ejemplo de problema

En un sistema de E/S de transferencia programada, se necesita transferir un archivo de 4 MB desde un dispositivo de almacenamiento a la memoria principal. El dispositivo de almacenamiento tiene una velocidad de transferencia de 10 MB/s. ¿Cuánto tiempo tomará la transferencia completa de E/S y cuál es la velocidad de transferencia en MB/s?

- Dado que el tamaño del archivo es de 4 MB y la velocidad de transferencia del dispositivo de almacenamiento es de 10 MB/s, podemos utilizar la fórmula de tiempo de transferencia:
  - Tiempo = Tamaño / Velocidad
  - Tiempo = 4 MB / 10 MB/s = 0.4 segundos
- Por lo tanto, la transferencia completa de E/S tomará 0.4 segundos.
- Para calcular la velocidad de transferencia en MB/s, simplemente utilizamos la fórmula de velocidad de transferencia:
  - Velocidad = Tamaño / Tiempo
  - Velocidad = 4 MB / 0.4 segundos = 10 MB/s
- Por lo tanto, la velocidad de transferencia es de 10 MB/s.

## PC3 (2023-1)

**Pregunta 1**: ¿Cual es el protocolo utilizado para transmisión de datos en buses asíncronos? Describa cada una de las etapas. En una tabla describa brevemente las diferencias entre los métodos de control y transmisión de datos para dispositivos I/O (V o F).

- El sistema I/O no afecta el rendimiento global de un computador.

- En una operación de Input, sólo se escriben datos en memoria principal. En una operación de Output, se leen datos de memoria.
- El acceso a un periférico consistente en seleccionar o identificar un dispositivo sobre el que se va a realizar una operación I/O.
- El rendimiento del sistema I/O depende exclusivamente de la velocidad del procesador.

**Pregunta 2**: Considere un sistema cuyo ciclo de bus asciende 550ns. La transferencia de control de del bus en cualquier dirección demora 275 más. Uno de los dispositivos conectados al bus transfiere data a 55 kB/s y emplea DMA La data es transferida 1 byte a la vez.

1. Si utilizamos DMA en modo ráfaga; es decir, la interfaz DMA gana el control de bus antes de la transmisión del bloque y mantiene el control hasta que la transferencia del bloque total haya terminado, Por cuanto tiempo en milisegundos el dispositivo tendrá ocupado el bus si se necesita transferir un bloque de 128 bytes.
2. Si cambiamos el modo de operación del DMA a robo por ciclo, Por cuanto tiempo en milisegundos el dispositivo tendrá ocupado el bus si se necesita transferir un bloque de 128 bytes.
3. Si abre el mismo bus se conecta un dispositivo que transfiere data a 8 kB/s de modo continuo, determinar la fracción de tiempo de procesador que ocupa el dispositivo I/O. Considerar que la transferencia de datos de este dispositivo se realiza por interrupciónes y está demora 100us. Además se genera una interrupción por cada byte que se quiera transferir

**Pregunta 3**: Justificar verdadero o falso.

- El forwarding no es una solución al riesgo de datos a nivel de compilador.
- Al utilizar calling conventions, sólo se tiene un máximo de 6 parámetros enteros para pasar como argumentos.
- Cuando se hace referencia a un espacio de la memoria caché, esta información siempre se transfiere por bloques de la memoria principal.
- El sistema de Entrada/Salida no afecta al rendimiento global de un computador si solo se usan sistemas de Salida.
- En una operación de ENTRADA solo se escriben datos en la memoria principal mientras que en una operación de SALIDA se leen datos de la memoria.
- Un método de acceso a un periférico consiste en seleccionar o identificar un dispositivo sobre el que se va a realizar una operación de E/S.
- El rendimiento del sistema de E/S es fácil evaluar debido a que únicamente depende de la velocidad del procesador.
- La principal misión de un módulo de E/S es aislar a la CPU y a la memoria de los detalles de operacion del periférico
- Sólo existen interrupciones por hardware.

## PC4 (2023-1)

**Pregunta 1**: Verdadero o falso.

- Es posible que dos hijos de un mismo programa se ejecuten en distintos procesadores.
- Todos los hilos creados por un mismo proceso pertenecen al mismo proceso. La comunicación entre procesos en el multiprocessing es más rápida en comparación con la comunicación entre lo hilos en el multithreading.
- La programación con los hilos es más adecuada para aplicaciones con cálculos intensivos.
- La concurrencia no siempre implica paralelismo.
- Las funciones asíncronas se ejecutan dentro de un mismo hilo.
- Los hilos pueden ser interrumpidos en cualquier punto de ejecución debido a la planificación del sistema operativo.
- El que decide en qué momento se deja de ejecutar una corutina para pasar a otra es el sistema operativo.
- Se obtiene un mayor Speed-up al descargar una base de datos con multipes procesos que con múltiples hilos.
- En multiprocessing, cada proceso tiene su propio espacio de memoria.

**Pregunta 2**: Sobre implementación en Python:

1. ¿Que es un bucle de eventos(event LOOP) y cuál es su papel en la programación asíncrona?
2. ¿Que es el Global Intérprete Loco y como afecta a la ejecución de hilos?

3. Si se generan varios procesos desde un mismo programa, cada subproceso tiene su propio intérprete? En cualquier caso, por qué es esto necesario?
4. Si es una computadora con 4 procesadores se intentará ejecutar un programa que es un multiprocessing y que tiene que ejecutarse en 8 procesos, que es lo que sucede?

**Pregunta 3**: Que consideraciones especiales se deben tener en cuenta al compartir recursos
(acceso a memoria, acceso a periféricos de E/S) entre:

1. Funciones asíncronas:
2. Hilos:
3. Procesos:

**Pregunta 4**: Responder las siguientes preguntas:

1. Se tiene el siguiente código, el cual se utiliza para que un servidor pueda conectar a múltiples cliente y reenviar toda la información que recibe ¿Es posible que se generen race conditions en algún punto de la ejecución? ¿Por qué? Si su respuesta es afirmativa, indicar que instrucción o instrucciones podrían ser problemáticos.

```
import socket
from threading import Thread

SOCK_BUFFER = 1024
num_clientes = 0

def client_handler(conn,client_adress):
    global num_clientes
    num_clientes += 1
    print(f"Numero de clientes conectados: {num_clientes}")
    try:
        print(f"Conectado desde {client_adress[0]}")

        while True:
            data = conn.recv(SOCK_BUFFER)
            print(f"Recibi: {data}")
            if data:
                print(f"Enviando: {data}")
                conn.sendall(data)
            else:
                print("No hay datos")
                break
    except ConnectionResetError:
        print("El cliente ha cerrado la conexion de manera abrupta")
    finally:
        print("Cerrando la conexcion con el cliente")
        num_clientes -= 1
        conn.close()

    if __name__=='__main__':
        sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

        server_adress = ('localhost',5000)
        print(f"Iniciado Conexion en {server_adress[0]}:{server_adress[1]}")
        sock.bind(server_adress)

        sock.listen(1)

        while True:
            print("Esperando conexiones")

            conn,client_adress = sock.accept()

            client_thread = Thread(target=client_handler,args=(conn,client_adress))
            client_thread.start()
```

2. El siguiente código debe descargar la respuesta http de una lista de urls de manera asíncrona para luego imprimirla. Para ello se considera una función asíncrona llamada async_fetch() ¿Qué es lo que sucede cuando se llama a cada función asíncrona?¿Se obtendrá un SpeedUp considerable respecto a la versión síncrona? Justifique (2pts).

```
import asyncio
import httpx #para instalar pip install httpx

urls = ("https://www.amazon.com","https://www.google.com","https://www.github.com","https://www.python.org","https://www.microsoft

async def async_fetch(url):
    async with httpx.AsyncClient() as client:
```

```
        response = await client.get(url)
        print(response.text)

async def main():
    for url in urls:
        await async_fetch(url)

if __name__=='__main__':
    asyncio.run(main())
```

3. Se desea encontra un PIN de seguridad de 4 digitos (0 al 9) por el método de fuerza bruta. Considere que se tiene en Python una función CPU-bound denomicada comprobar_PIN(PIN: int) que te devuelve verdadero o falso en caso el PIN sea el correcto ¿Cómo se podría paralelizar la búsqueda para N núcleos? Indique que haría la función a paralelizar, así como sus parámetros de entrada y salida (2pts).

## EX2 (2023-0)

Escriba un programa en Python que imprima si un número es primo o no. El algoritmo sugerido para saber si un número es primo es el siguiente:

- Dado un número n, verificar si es divisible entre 2 , 3, 4, … … $\sqrt{n}$
- Si es divisible entre algún número dentro de ese intervalo, no es primo; caso contrario sí es primo.

1. (2 puntos) Escriba una función que reciba como parámetro de entrada el argumento n y retorne True si el número es primo, o False en caso contrario (Implementación serial)

   Imprima el tiempo de ejecución para n = 2 345 678 911 111 111

2. (5 puntos) Escriba una función que permita paralelizar el cálculo de si un número es primo o no. Use 2 procesos.

   Imprima el tiempo de ejecución de esta función para el mismo valor de n de la parte a). Calcule el Speedup. Use la función assert() para verificar que el resultado en la parte a) sea igual a la parte b)

3. (3 puntos) Calcule el tiempo de ejecución para cuando se usa 4, 8, y 16 procesos. Grafique el número de procesos (eje x) vs el tiempo de ejecución (eje y). ¿Aumentar el número de procesos hace que disminuya el tiempo de ejecución, o no necesariamente?

> 💡 Nota: Puede hacer el gráfico en Excel o hacerlo a mano y tomarle foto, pero debe subir la imagen del gráfico para que se le considere puntaje.

### Solución

```
import time
import matplotlib.pyplot as plt
from threading import Thread

rango = list()
valor = list()

def es_primo(n:int):
    max = int(n**(0.5))
    for i in range (2,max+1):
        if n%i == 0:
            return False
    return True

def es_primo_hilo(inicio:int,fin:int):
    fin_ran = int(fin**(0.5))
    for i in range (inicio,fin_ran+1):
        if n%i == 0:
            valor.append(False)
    rango[0] = fin
    rango[1] += fin
    valor.append(True)

def sincrono(n:int):
    if es_primo(n):
        print("Es primo_sincrono")
        return True
    else:
        print("No es primo_sincrono")
        return False
```

```
def hilos(n:int,num_procesos:int):
    global rango
    rango.append(2)
    rango.append(int(n/2))
    global valor
    hilos = list()
    for i in range(num_procesos):
        hilos.append(Thread(target=es_primo_hilo,args=(int(rango[0]),int(rango[1]))))
        hilos[i].start()

    res = True
    for i in range(len(valor)):
        res = res and valor[i]

    if res:
        print("Es primo_hilo")
        return True
    else:
        print("No es primo_hilo")
        return False

if __name__=="__main__":
    #n = int(input("Ingresa un numero: "))
    n = 2_345_678_911_111_111
    #n = 5
    tiempo = list()
    cant_procesos = list()

    inicio_sinc = time.perf_counter()
    val_sincr = sincrono(n)
    fin_sinc = time.perf_counter()

    inicio_hilos = time.perf_counter()
    val_hilos = hilos(n,2)
    fin_hilos = time.perf_counter()
    tiempo.append(fin_hilos-inicio_hilos)
    cant_procesos.append(2)

    inicio_hilos = time.perf_counter()
    val_hilos = hilos(n,4)
    fin_hilos = time.perf_counter()
    tiempo.append(fin_hilos-inicio_hilos)
    cant_procesos.append(4)

    inicio_hilos = time.perf_counter()
    val_hilos = hilos(n,8)
    fin_hilos = time.perf_counter()
    tiempo.append(fin_hilos-inicio_hilos)
    cant_procesos.append(8)

    inicio_hilos = time.perf_counter()
    val_hilos = hilos(n,16)
    fin_hilos = time.perf_counter()
    tiempo.append(fin_hilos-inicio_hilos)
    cant_procesos.append(16)

    assert(val_sincr == val_hilos)
    print(f"tiempo sincrono {fin_sinc-inicio_sinc}")
    print(f"tiempo hilos {tiempo[0]}")

    plt.plot(cant_procesos,tiempo)
    plt.show()
```