Examen parcial- Computación concurrente

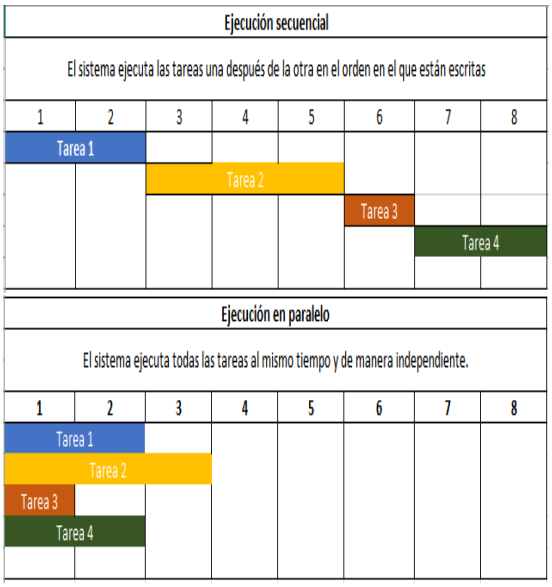
Elsy Camila Silva Velázquez
Luis Octavio Sánchez Hatadi
Guillermo Gerardo Andrés Urbano

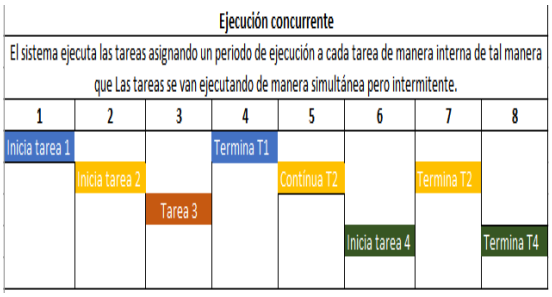
12 de Octubre del 2021

1. Elabora con tus palabras una descripción lo más completa posible del concepto de concurrencia.

   Se puede definir concurrencia como la capacidad de los sistemas de ejecutar varios procesos de manera simultánea e independiente mediante la asignación de intervalos de tiempo creando la ilusión de ejecución en paralelo. Cabe resaltar que dichos procesos pueden o no interactuar entre sí.

2. Utiliza diagramas de tiempos y tareas (diagrama de Gantt) donde muestres los tres tipos de ejecución: secuencial, concurrente y paralela.





3. Menciona lo que son las llamadas al sistema, el estándar POSIX y su uso para implementar concurrencia en linux-C.

   <**sys/types.h**> Encabezado que contiene definiciones de varios tipos, uno de ellos será `pid_t`.

   <**unistd.h**>.Es el archivo de cabecera que proporciona acceso a la API del sistema operativo POSIX.

pid_t. Se utiliza para ID de proceso e ID de grupo de proceso.

fork(). Crea un nuevo proceso duplicando el proceso que lo llamo. El nuevo proceso se denomina proceso hijo. El proceso de llamada se conoce como proceso padre. El proceso hijo y el proceso padre se ejecutan en espacios de memoria separados. En el momento de 'fork()' ambos espacios de memoria tienen el mismo contenido. En caso de éxito, el PID del proceso hijo se devuelve en el padre y se devuelve 0 en el hijo. En caso de error, se devuelve -1 en el padre, no se crea ningún proceso hijo.

exit(estado). La función de biblioteca exit finaliza un proceso, haciendo que todos los recursos (memoria, descriptores de archivos abiertos, etc.) utilizados por el proceso disponible para posterior reasignación por parte del kernel. El argumento de estado es un número entero que determina el estado de terminación del proceso. Usando la llamada al sistema wait(), el padre puede recuperar este estado.

wait(estado). La llamada al sistema wait(estado) tiene dos propósitos. Primero, si un hijo de este proceso aún no ha terminado llamando a exit(), entonces wait() suspende la ejecución del proceso hasta que uno de sus hijos haya terminado. En segundo lugar, el estado de terminación del hijo se devuelve en el argumento de estado de wait().

getpid(). Devuelve el identificador de proceso del proceso actual

getppid(). Devuelve el identificador de proceso del padre del proceso actual.

getuid(). Devuelve el identificador de usuario actual.

pipe(pipefd). Crea una tubería, un canal de datos unidireccional que se puede utilizar para la comunicación entre procesos. La matriz pipefd se utiliza para devolver dos descriptores de archivo que se refieren a los extremos de la tubería.

- pipefd[0] se refiere al extremo leído de la tubería.
- pipefd[1] se refiere al extremo de escritura de la tubería. Los datos escritos en el extremo de escritura de la tubería son almacenados en búfer por el kernel hasta que se leen desde el extremo de lectura de la tubería.

write(fd, *buf, count) . La llamada al sistema write hace que los primeros bytes del buffer sean escritos en el archivo asociado con el descriptor de archivos `fd`.

read(fd, *buf, count). Lee de un descriptor de archivo indicando el numero de bytes y un buffer donde se guardara la lectura.

close(). Cierra un descriptor de archivo, de modo que ya no hace referencia a ningún archivo y se puede reutilizar.

gets(s). Lee una línea desde la entrada estadandar en el búfer al que apunta `s` hasta que termina una nueva línea o EOF.

fflush() Es una función para la limpieza de buffer de escritura, escritura de ficheros o escritura en la pantalla.

usleep() Suspender la ejecución para intervalos de microsegundos.

<**pthread.h**>.Es el archivo de cabecera que proporciona tipos, contantes y funciones para la manipulacion de `threads`.

pthread_create(*thread) Crea un nuevo hilo pasando como argumento un apuntador `thread` donde se guardará.

pthread_join(thread). La función espera a que termine el hilo especificado por `thread`. Si ese hilo ya ha terminado, entonces `pthread_join()` regresa inmediatamente. El hilo especificado por `thread` debe poder unirse.

Ejemplo en C:

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
    int pid;
    // Creacion de un proceso
    pid = fork();
    printf("ID proceso: %d\n", pid);
    if (pid)
         printf("Hola soy el proceso padre!!\n");
    else
        printf("Hola soy el proceso hijo!\n");
    return 0;
}
```

4. Implementa en Python tres versiones de un programa que calcula el factorial de un número entero proporcionado desde el teclado:

- a) Implementa la versión secuencial.
- b) Implementa la versión en la cual se crean dos procesos, qcada uno un valor entero desde el teclado y calcula el factorial correspondiente.
- c) Implementa la versión en la que se crean cinco hilos que reciben un número real desde el teclado, cada hilo calcula el factorial del número recibido.

0.0.1 Factorial de un numero

[ ]:
```
def factorial(n):
    product = 1
    for i in range(1, n+1):
        product *= i
    print(f'{product}\n')
```

0.1 Vesión secuencial

[39]:
```
n = int(input())
print("-----")
factorial(n)
```

```
5
-----
120
```

0.2 Versión creación de procesos

[40]:
```
import multiprocessing as mp

n1, n2 = int(input()), int(input())
print("-----")

# Creacion de 2 procesos
p1 = mp.Process(target=factorial, args=(n1,))
p2 = mp.Process(target=factorial, args=(n2,))

# Iniciando los procesos
p1.start()
p2.start()

# Haciendo que el padre espere a los dos procesos
p2.join()
p2.join()
```

```
5
7
-----
120
5040
```

0.3 Version creación de hilos

[37]:
```python
import threading as th

nums = []
threads = []
n = 5

for i in range(n):
    nums.append(int(input()))

print("-----")
# Creacion de hilos e iniciando el proceso
for i in range(n):
    thread = th.Thread(target=factorial, args=(nums[i],))
    threads.append(thread)
    threads[i].start()

# Espera el padre a los hilos
for i in range(n):
    threads[i].join()
```

```
5
6
7
2
4
-----
120

720

5040

2

24
```

5. Describe la acción de bloqueo, la razón por la cual se utiliza y los métodos que ofrece python para implementarla.

Existen recursos los cuales no se deben acceder al mismo tiempo por dos procesos o más; por lo que es necesario proteger o bloquear el acceso a estos recursos compartidos: memoria, archivos, bases de datos.

La razon por la cual surgen es por la **condición de carrera** (race condition), ocurre cuando dos o más procesos acceden un recurso compartido sin control, de manera que el resultado combinado de este acceso depende del orden de llegada.

El acceso a un recurso compartido por mas de dos tareas, existe la posibilidad de corromper la información.Cuando existen dos procesos pueden modificar la variables, por lo tanto debe de ser sincronizadas.

Sincronizar es tener cierto orden en los recursos, es un mecanismo para compartir recursos de manera ordenada.

Cuando una tarea solita el acceso para modificarla, pongo un bloqueo, para que el proceso no lea un valor desactualizado

Python ofrece la clase `Lock` para la condición de carrera, ofrecera dos estados `locked` o `unlocked`, de esta manera poder sincronizar los procesos, esta clase continen los metodos:

- **lock.acquire()**: Cuando el estado es *unlocked*, `acquire()` cambia el estado a *locked*. Cuando el estado está *locked*, `acquire()` bloquea hasta que una llamada a `release()` en otro hilo lo cambie a *unlocked*, luego la llamada de `acquire()` lo restablece a bloqueado y regresa.
- **lock.release()**: El metodo `release()`,se utiliza para liberar un bloqueo adquirido, deberia ser solo llamado cuando esta en estado *locked* ,si no lanzará una exepción *a RuntimeError*.

Ejemplo:

[41]:
```
import time
import multiprocessing

def deposit(balance, lock):
    for i in range(100):
        time.sleep(0.01)
        lock.acquire()
        balance.value = balance.value + 1
        lock.release()

def withdraw(balance, lock):
    for i in range(100):
        time.sleep(0.01)
        lock.acquire()
        balance.value = balance.value - 1
        lock.release()


if __name__ == '__main__':
    balance = multiprocessing.Value('i', 200)
    lock = multiprocessing.Lock()
    d = multiprocessing.Process(target=deposit, args=(balance, lock))
    w = multiprocessing.Process(target=withdraw, args=(balance, lock))
    d.start()
    w.start()
    d.join()
    w.join()
    print(balance.value)
```

200

6.- Con tus propias palabras resume los primeros conceptos que has identificado en la lectura del artículo de Dijkstra.

Procesos conectados: Procesos que pueden comunicarse entre ellos y además, intercambiar información.

Procesos débilmente conectados. Los procesos se ejecutan de manera independiente entre sí, a excepción de cuando hay intercomunicación explícita.

Proceso secuencial: Son aquellas tareas en las que la salida de una es la entrada de la siguiente tarea y así sucesivamente hasta terminar con el proceso.

Sección crítica: Es la parte del proceso de comunicación entre procesos procesos en la deben de interactuar de tal forma que se evite inconsistencias al momento de asignar nuevos valores a variables comunes.

Problema de exclusión mutua: Es el mecanismo que tienen los sistemas operativos para evitar que dos o más procesos ingresen de manera simultánea a los datos compartidos.

ALGOL-60: (Algorithmic Language) es un tipo de lenguaje de programación utilizado para la estandarización de la escritura de algoritmos.

Semáforos. Dos números enteros no negativos que pueden ser 0 o 1.

Operación-V. Incrementar el valor del semáforo en 1.

Operación-P. Disminuir en 1 el valor del semáforo.