



Práctico 4

Sincronización de Procesos

4.1. Definir el concepto de proceso cooperativo y condición de carrera (*race condition*).

4.2. Las condiciones de carrera son posibles en muchos sistemas informáticos. Consideremos un sistema bancario que posee las siguientes dos funciones: `depósito(cantidad)` y `retiro(cantidad)`. Supongamos entonces que existe una cuenta bancaria compartida entre un matrimonio y al mismo tiempo el marido ejecuta un `retiro()` y la esposa ejecuta un `depósito()`. Describir cómo es posible una condición de carrera en este caso y qué podría hacerse para evitar que se produzca.

4.3. Describir dos estructuras de datos del kernel donde sean posibles las condiciones de carrera.

4.4. Describir los tres requisitos que debe satisfacer una solución al problema de la sección crítica.

4.5. Determinar si el siguiente algoritmo en pseudocódigo representa una solución al problema de la sección crítica.

```
shared int turn = 1;
int myPID = 0;
int otherPID = 1 - myPID ;

while ( turn != myPID ) do no -op;    // Entry Section
    // Critical Section
turn = otherPID ;                      // Exit Section
```

4.6. ¿Cuál es el significado del término “*busy waiting*” (espera ocupada)? ¿Qué otros tipos de espera hay en un sistema operativo? ¿Puede evitarse por completo el “*busy waiting*”?

4.7. Explicar por qué los *spinlocks* no son apropiados para sistemas de un solo procesador pero sí se utilizan a menudo en sistemas multiprocesador.

4.8. Explicar por qué la implementación de primitivas de sincronización mediante la desactivación de interrupciones no es apropiada en un sistema de un solo procesador, si las primitivas de sincronización son utilizadas en programas a nivel de usuario.

4.9. Considerar un programa que crea dos procesos, P y Q, los cuales se definen de la siguiente manera:

```
void p() {          void q() {
    A;              D;
    B;              E;
    C;              }
}
```

A, B, C, D, y E son declaraciones atómicas (indivisibles). Mostrar a partir de estas declaraciones todas las posibles ejecuciones de los dos procesos.

4.10. ¿Representan los semáforos una solución al problema de la sección crítica? Justificar adecuadamente.

4.11. Mostrar que, si las operaciones de semáforo *wait()* y *signal()* no son ejecutadas atómicamente, entonces la exclusión mutua puede ser violada.

4.12. Los servidores pueden ser configurados para limitar el número de conexiones abiertas. Por ejemplo, un servidor puede tener solamente N conexiones de *socket* en cualquier instante en el tiempo. Tan pronto como se hacen las N conexiones, el servidor no aceptará otra conexión de entrada hasta que se libere una conexión existente. Explicar cómo los semáforos pueden ser utilizados por un servidor para limitar el número de conexiones simultáneas.

4.13. Se ha configurado una impresora como recurso compartido de red, que será utilizada por tres procesos: P1, P2 y P3, que se ejecutan en un servidor. Los procesos pertenecen a un programa que realiza tareas específicas y es necesaria su sincronización. Escribir para cada uno de los siguientes incisos un algoritmo en pseudocódigo, indicando en cada caso la sección crítica, los semáforos utilizados y su respectiva inicialización.

- a) Dados P1 con las instrucciones `print(i1)` y `print(i2)` y P2 con las instrucciones `print(j1)` y `print(j2)`, se solicita que P1 espere por P2 y P2 por P1 de la siguiente manera: que `print(i1)` se ejecute antes que `print(j2)` y que `print(j1)` se ejecute antes que `print(i2)`. El proceso P3 no está incluido en la sincronización.

Nota: Este tipo de problema se denomina “*rendezvous*”; en el cual dos hilos se encuentran en un punto de ejecución y ninguno tiene permitido continuar hasta que ambos hayan llegado.

- b) Dado P2 que sólo puede ejecutar su instrucción `print(X)` si P1 ha ejecutado su instrucción `print(Y)` y P3 ha ejecutado su instrucción `print(Z)`. ¿Se puede resolver el problema con un solo semáforo y con valor inicial 0?

Nota: Este uso de semáforos es del tipo “*signaling*” (señalización), lo que significa que un hilo envía una señal al otro hilo para indicar que algo ha sucedido.

- c) Los tres procesos se deben ejecutar en el siguiente orden: P1, P3 y P2. Una vez finalizado P2, se ejecuta P1 nuevamente y así sucesivamente.
- d) Se ha agregado el proceso P4 al conjunto de procesos creados por el programa, de manera que P4 sólo se pueda ejecutar si P2 y P3 se ejecutaron; a su vez, P2 y P3 sólo se pueden ejecutar si P1 ya se ejecutó.
- e) Los tres procesos que comparten la impresora lo deben hacer de manera exclusiva.

Nota: Este uso de semáforos es para forzar la exclusión mutua. Un “*mutex*” garantiza que solo un hilo acceda a un recurso compartido (por ej.: variable compartida) a la vez.

4.14. Un gimnasio está teniendo un aumento en el número de nuevos socios, motivo por el cual está teniendo dificultades por la poca cantidad de aparatos con los que cuenta en relación a la cantidad de gente que concurre al mismo. El gimnasio cuenta con 5 barras, 3 bancos press y 10 pares de mancuernas. La rutina de ejercitación se realiza en el siguiente orden: sentadillas con barras, pull over sobre el banco y estocadas con mancuernas.

Escribir el algoritmo en pseudocódigo del proceso que ejecutarán los usuarios de manera concurrente utilizando semáforos para la sincronización.

4.15. Modificar el ejercicio anterior de tal manera que al terminar con la rutina el usuario deje su toalla para lavar en un canasto al cual se permite el ingreso de a uno por vez. A continuación, un encargado de limpieza las saca de a una para llevarlas a lavar.

4.16. El problema del Productor-Consumidor. El problema del productor-consumidor (también conocido como el problema del búfer limitado (*bounded-buffer problem*)) es un ejemplo clásico de sincronización de procesos. El problema describe dos procesos, el productor y el consumidor, que comparten un búfer de tamaño fijo. El trabajo del productor es generar datos y colocarlos en el búfer. Al mismo tiempo, el consumidor está consumiendo los datos de uno a la vez. El problema es asegurarse de que el productor no trate de añadir datos en el búfer si está lleno y que el consumidor no trate de eliminar datos de un búfer vacío.

Los procesos productor y consumidor comparten las siguientes estructuras de datos:

```
semaphore mutex;
semaphore empty;
semaphore full;
```

Estructura Proceso Productor	Estructura Proceso Consumidor
<pre>do { . . . item = produceItem(); . . . wait(empty); wait(mutex); . . . putItemIntoBuffer(item); . . . signal(mutex); signal(full); } while (true);</pre>	<pre>do { . . . wait(full); wait(mutex); . . . item = removeItemFromBuffer(); . . . signal(mutex); signal(empty); . . . consumeItem(item); } while (true);</pre>

A partir de la definición anterior responder las siguientes preguntas:

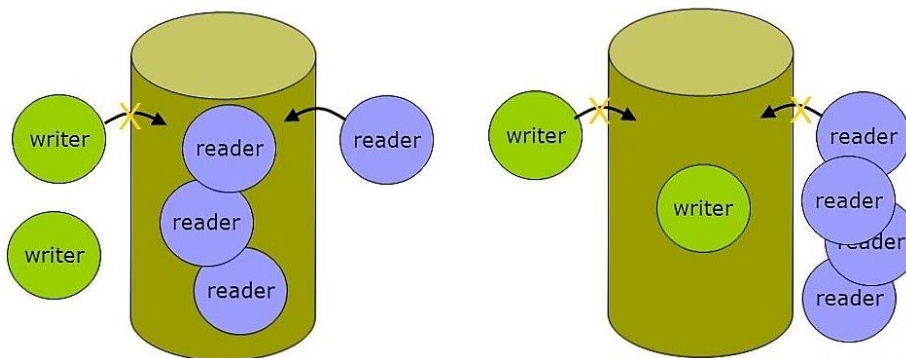
- ¿En qué valor se tiene que inicializar el semáforo `mutex`? ¿Por qué?
- ¿En qué valor se tiene que inicializar el semáforo `empty`? ¿Por qué?
- ¿En qué valor se tiene que inicializar el semáforo `full`? ¿Por qué?
- En caso de que se inicialice `full = 1`, ¿qué proceso entra primero en la sección crítica?

- e) En caso de que se inicialice $full = n - 1$ y $empty = 1$, si se ejecutan tres procesos productores seguidos de un consumidor, ¿cuál es el estado final de los semáforos?

4.17. El problema de los lectores y escritores. Un cierto número de lectores pueden leer un recurso compartido en simultáneo. Sin embargo, sólo un escritor por vez puede escribir sobre el recurso y ningún lector puede leer mientras se está realizando la operación de escritura. El problema de los lectores-escritores tiene diversas variantes, todas relacionadas con prioridades:

- El **primer problema de los lectores-escritores** (*first readers-writers problem*) requiere que ningún lector espere a menos que un escritor ya haya obtenido permiso para usar el recurso compartido. En otras palabras, ningún lector debería esperar a que otros lectores terminen simplemente porque un escritor está esperando.
- El **segundo problema de los lectores-escritores** (*second readers-writers problem*) requiere que, una vez que un escritor está listo, éste realice su escritura tan pronto como sea posible. En otras palabras, si un escritor está esperando para acceder al recurso compartido, ningún nuevo lector puede comenzar a leer.

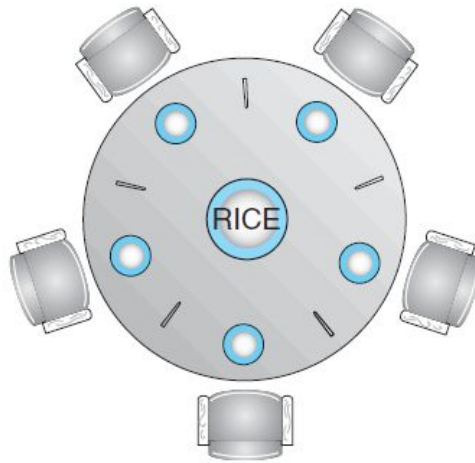
■ Multiple readers or a single writer can use DB.



- a) Escribir un algoritmo en pseudocódigo que permita solucionar el primer problema de los lectores-escritores utilizando semáforos.
- b) Escribir un algoritmo en pseudocódigo que permita solucionar el segundo problema de los lectores-escritores utilizando semáforos.
- c) ¿Cuál será la razón por la cual ha surgido un tercer problema de los lectores-escritores (*third readers-writers problem*)?

Nota: No es necesario realizar el algoritmo.

4.18. El problema de los filósofos cenando. Cinco filósofos están sentados alrededor de una mesa redonda. Cada filósofo posee un plato de arroz y entre cada par de filósofos existe un palillo. Cada filósofo está en uno de tres estados posibles: pensando, hambriento o comiendo. En un determinado momento, un filósofo que estaba pensando siente hambre, por lo que intenta obtener uno de los palillos adyacentes y posteriormente el otro. Si un filósofo es capaz de obtener los dos palillos (es decir que están ambos disponibles), entonces el filósofo come durante una cierta cantidad de tiempo. Luego de comer, el filósofo retorna ambos palillos a la mesa.



- Escribir un algoritmo en pseudocódigo que permita solucionar este problema utilizando semáforos.
- ¿Por qué es tan importante este problema?

4.19. El problema del barbero dormilón. Una barbería está compuesta por una sala de espera con n sillas y una sala de barbería en donde se encuentra la silla del barbero. Si no hay clientes que atender, el barbero se pone a dormir. Si un cliente entra a la barbería y todas las sillas están ocupadas, entonces el cliente se retira. Si el barbero está ocupado pero existen sillas disponibles, entonces el cliente se sienta a esperar en una de las sillas libres. Si el barbero está dormido, el cliente lo despierta.



- Escribir un algoritmo en pseudocódigo que permita sincronizar al barbero con sus clientes utilizando semáforos.

4.20. Monitores

- ¿Cuál es la principal diferencia entre un tipo de dato abstracto genérico y uno del tipo monitor?
- Describir los mecanismos agregados al monitor para poder proveer sincronización y, cuáles son sus problemáticas.