Departamento de Ciencias e Ingeniería de la Computación

Ingeniería en Computación
Ingeniería en Sistemas de Información

SISTEMAS OPERATIVOS

Segundo Cuatrimestre de 2020

Trabajo Práctico Nº 3

1. Problemas: Sincronización de Procesos

1. Hay 3 procesos cooperativos que acceden a 2 secciones críticas, y cada proceso tiene derecho a acceder a las 2 secciones críticas en un instante de tiempo. La estructura de los procesos es la siguiente:

```
P1: while (1) {          P2: while (1) {          P3: while (1) {
...                      ...                      ...
while (T&S(&lock1));     while (T&S(&lock1));     while (T&S(&lock2));
...                      ...                      ...
while (T&S(&lock2));     while (T&S(&lock2));     while (T&S(&lock1));
...                      ...                      ...
...                      ...                      ...
lock2 = 0;               lock1 = 0;               lock2 = 0;
lock1 = 0;               lock2 = 0;               lock1 = 0;
...                      ...                      ...
}                        }                        }
```

   Suponga que los procesos no utilizan otro recurso a parte de los locks. Ambos locks están en cero antes que el tercer proceso comience. ¿Cuál de las siguientes aseveraciones es verdadera?

   a) No hay posibilidades de deadlock con esta inicialización.

   b) Deadlock es posible y puede estar afectado un sólo proceso.

   c) Deadlock es posible y pueden estar afectados dos procesos.

   d) Deadlock es posible y pueden estar afectados los tres procesos.

2. Tweedledum y Tweedledee son threads que están ejecutando su respectivo procedimientos. El código presentado intenta que para siempre estén intercambiando opiniones a través de una variable compartida X en estricta alternancia. Las rutinas de Sleep() y Wakeup() operan de la siguiente manera: Sleep bloquea el thread invocador, y Wakeup desbloquea un thread específico si ese thread está bloqueado, de otra forma su comportamiento es impredecible.

```
void                                    void
Tweedledum()                            Tweedledee()
{                                       {
   while(1) {                              while(1) {
     Sleep();                                x = Quarrel(x);
     x = Quarrel(x);                         Wakeup(Tweedledum thread);
     Wakeup(Tweedledee thread);              Sleep();
   }                                       }
}                                       }
```

a) Muestre un escenario en el cual el código pueda fallar.

b) Muestre como solucionar el problema reemplazando las llamadas Sleep y Wakeup con operaciones de semáforo. No se pueden deshabilitar las interrupciones.

c) Implemente Tweedledum y Tweedledee correctamente utilizando mutex y variable de condición.

3. Explique la diferencia entre la espera ocupada y el bloqueo.

4. Muestre que si *wait* y *signal* no son ejecutadas atómicamente, la exclusión mutua puede ser violada.

5. De una idea de la forma en que un sistema operativo con posibilidades de desactivar las interrupciones podría implantar semáforos.

6. ¿Cuáles son las diferencias entre un semáforo binario y un mutex?

7. Dado el clásico problema de productores y consumidores, se plantea una solución utilizando semáforos con el siguiente código:

```
Typedef int semaforo; .
typedef char* msg;
int N=100; /*Longitud del buffer */
semaforo mutex = 1; /*Da la exclusión mutua */
semaforo lleno = 0; /*Cuenta lugares llenos */
semaforo vacio = N; /*Cuenta lugares vacíos */
```

```
Productor()                     Consumidor()
{                               {
msg mensaje;                    msg mensaje;
while(TRUE)                     while(TRUE)
{                               {
producir(mensaje);              down(&lleno);
down(&mutex);                   down(&mutex);
down(&vacio);                   remover_msg(mensaje);
entrar_msg(mensaje);            up(&mutex);
up(&mutex);                     up(&vacio);
up(&lleno);                     consumir_msg(mensaje);
}                               }
}                               }
```

¿La solución planteada es válida?. En caso de que no lo sea, explique por qué.

8. Se tienen las siguientes secuencias de ejecución:

   a) La secuencia permitida es: ABCDEABCDEABCDEABCDEABCDEABCDE ...

   b) La secuencia permitida es: ACDEBCDEACDEBCDEACDEBCDEACDEBCDE ...

   c) La secuencia permitida es: (A o B)CDE(A o B)CDE(A o B)CDE(A o B)CDE ...

   d) La secuencia permitida es: (AóB)CE(AóB)(AóB)DE(AóB)CE(AóB)(AóB)DE ...

   Realizar la sincronización de los procesos utilizando semáforos para cada uno de los casos especificados.

9. Analice la siguiente solución para el problema de los filósofos. ¿La solución propuesta tiene algún problema? De ser así explique cual es y cómo lo solucionaría.

```
#define   N              6
#define   Izquierdo      (i-1) % N
#define   Derecho        (i+1) % N
#define   Pensando       0
#define   Tiene_Hambre   1
#define   Comiendo       2
Typedef   int            semaforo
int estado[N];
semaforo mutex = 1;
semaforo s[N];

void filosofo (int i)
{
while (TRUE) {
pensar();
tomar_tenedores(i);
comer();
dejar_tenedores(i);
}
}

void tomar_tenedores (int i)
{
down(&mutex);
estado[i] = Tiene_Hambre;
test(i);
up(&mutex);
down(&s[i]);
}

void dejar_tenedores (int i)
{
down(&mutex);
estado[i] = Pensando;
up(&mutex);
test(Izquierdo);
test(Derecho);
}

void test (int i)
{
if (estado[i] == Tiene_Hambre && estado[Izquierdo] != Comiendo && estado[Derecho]
!= Comiendo) {
estado[i] = Comiendo;
up(&s[i]);
}
}
```

10. En la fábrica de bicicletas MountanenBike, tenemos tres operarios que denominaremos OP1, OP2 y OP3. OP1 monta ruedas, OP2 monta el cuadro de las bicicletas, y OP3, el manillar. Un cuarto operario, el Montador, se encarga de tomar dos ruedas, un cuadro y un manillar, y arma la bicicleta. Sincronizar las acciones de los tres operarios y el Montador utilizando semáforos, en los siguientes casos:

    a) Los operarios no tienen ningún espacio para almacenar los componentes producidos pero si tiene espacio para fabricar de a uno, y el Montador no podrá tomar ninguna pieza si esta no ha sido fabricada previamente por el correspondiente operario.

    b) Los operarios no tienen ningún espacio para almacenar los componentes producidos pero si tiene espacio para fabricar de a uno, y en el caso del OP1 tiene espacio para fabricar las dos ruedas y el Montador no podrá tomar ninguna pieza si esta no ha sido fabricada previamente por el correspondiente operario.

    Restricción: el OP1 sólo produce de a una rueda por vez.

11. El problema de los fumadores de cigarrillos (Patil 1971). Considere un sistema con tres procesos fumadores y un proceso agente. Cada fumador está continuamente armando y fumando cigarrillos. Sin embargo, para armar un cigarrillo, el fumador necesita tres ingredientes: tabaco, papel y fósforos. Uno de los procesos fumadores tiene papel, otro tiene el tabaco y el tercero los fósforos. El agente tiene una cantidad infinita de los tres materiales. El agente coloca dos de los ingredientes sobre la mesa. El fumador que tiene el ingrediente restante armaría un cigarrillo y se lo fuma, avisando al agente cuando termina. Entonces, el agente coloca dos de los tres ingredientes y se repite el ciclo. Escriba un programa para sincronizar al agente y los fumadores utilizando semáforos.

12. El problema del barbero dormilón (Dijkstra 1971). Una barbería se compone de una sala de espera con n sillas y la sala de barbería donde se encuentra la silla del barbero. Si no hay clientes a quienes atender, el barbero se pone a dormir. Si un cliente entra y todas las sillas están ocupadas, el cliente sale de la barbería. Si el barbero está ocupado, pero hay sillas disponibles, el cliente se sienta en una. Si el barbero está dormido, el cliente lo despierta. Escriba un programa para coordinar al barbero y sus clientes utilizando semáforos.

13. Escriba un algoritmo con semáforos, que controle el acceso a un archivo, de tal manera que se permita el acceso en lectura a varios procesos o a un solo escritor en forma exclusiva.

14. Considere la siguiente implementación para el problema de los lectores-escritores utilizando monitores para la implementación de las funciones para acceder a los datos compartidos.

    Considere que si c es una variable de tipo *condition*, entonces la función booleana $empty(c)$ retorna $falso$ cuando existan uno o varios procesos esperando en la cola de c, de lo contrario retorna $verdadero$.

a) Explique las políticas de prioridad utilizadas para los lectores y escritores para el diseño de esta solución.

b) ¿Existe la posibilidad de inanición?

```
MONITOR lector-escritor{
    int lecto_cant, escribiendo = 0;
    condition ok_leer, ok_escribir;
INICIO-LEER{
    if (escribiendo || !empty(ok_escribir)) ok_leer.wait;
    lecto_cant = lecto_cant + 1;
    ok_leer.signal;
}
FIN-LEER {
    lecto_cant = lecto_cant - 1;
    if (lecto_cant == 0) ok_escribir.signal;
}
INICIO-ESCRIBIR {
    if ((lecto_cant != 0) || writing) ok_escribir.wait;
    escribiendo = 1;
}
FIN-ESCRIBIR {
    escribiendo = 0;
    if (!empty(ok_leer)) ok_leer.signal;
    else ok_escribir.signal;
}
}
```

15. Escriba un algoritmo que resuelva el problema productor-consumidor, con un tamaño de buffer limitado, utilizando monitores para la sincronización y exclusión.

16. El oso y las abejas. Se tienen n abejas y un hambriento oso. Comparten un tarro de miel. Inicialmente el tarro de miel está vacío, su capacidad es M porciones de miel. El oso duerme hasta que el tarro de miel se llene, entonces se come toda la miel y vuelve a dormir. Cada abeja produce una porción de miel que coloca en el tarro, la abeja que llena el tarro de miel despierta al oso. Escriba un programa para que sincronice a las abejas y al oso.

17. Los servidores pueden diseñarse de modo que limiten el número de conexiones abiertas. Por ejemplo, un servidor puede autorizar sólo N conexiones simultáneas de tipo *socket*. En cuanto se establezcan las N conexiones, el servidor ya no aceptará otra conexión entrante hasta que una conexión existente se libere. Explique cómo puede utilizar un servidor los semáforos para limitar el número de conexiones concurrentes.

18. Suponga tener n usuarios que comparten 2 impresoras. Antes de utilizar la impresora, el U[i] invoca requerir(impresora). Esta operación espera hasta que una de las impresoras esté disponible, retornando la identidad de la impresora libre. Después de utilizar a impresora, U[i] invocan a liberar(impresora).

    a) Resuelva este problema considerando que cada usuario es un thread y la sincronización se realiza utilizando semáforos.

    b) Resuelve este problema considerando que cada usuario tiene una prioridad única. Se otorga la impresora al usuario con mayor prioridad que está esperando.

19. Cómo se modificaría el ejercicio 2 de la sección problemas del Trabajo Práctico Nro. 3 (diagrama de transición de procesos) si se previera el uso de semáforos para sincronizar los procesos entre sí. Qué nuevos estados habría que agregar al diagrama de transiciones y qué nuevas rutinas de manejo de las mismas habría que prever si:

    a) Se incluye un semáforo que controla el uso del canal correspondiente a las unidades de cinta y otro al correspondiente a las unidades de disco.

    b) Se incluye un semáforo que controla la obtención dinámica de espacio, en memoria central por parte de los procesos en ejecución.