Sistemas Operativos.

2019

1. Sincronización de procesos.

En un sistema multiprocesador es posible que durante la ejecución de los procesos, algunos se traslapen desde el punto de vista del uso de recursos. Por ejemplo, si dos procesos necesitan acceder a un recurso de hardware, es posible que los dos intenten acceder al mismo tiempo, o en su defecto, que uno intente acceder mientras el otro ya está haciendo uso de dicho recurso. En éstos casos, es necesario implementar un mecanismo que permita controlar la forma en que los procesos acceden a algunos recursos de la máquina. En éste caso, es necesario un mecanismo de sincronización a fin de que se acceda de forma ordenada a los recursos de la máquina.

Para entender mejor los mecanismos mediante los cuales pueden existir problemas que requieran sincronización, es necesario determinar las interacciones entre los procesos en un sistema operativo:

- Procesos no relacionados. Son procesos que no tienen una tarea específica común, pero que podrían llegar a **competir** por un recurso.
- Procesos relacionados indirectamente. En este caso, los procesos no son emparentados, pero comparten algún objeto o *buffer* de memoria. Éstos procesos presentan **cooperación mediante compartición** de un objeto.
- Procesos relacionados directamente. Se comunican entre sí directamente y están diseñados para realizar una tarea específica. Estos procesos presentan **cooperación**.

A continuación, se presentan los problemas más comunes en cada caso de interacción.

1. Procesos que compiten por un recurso.

   En el caso de los procesos que desean acceder a recursos de la máquina, surgen tres tipos de problemas:

   - Exclusión mutua: define una forma en que los procesos deberían acceder a un recurso. En realidad, define la idea de que si un proceso accede a una sección crítica, los demás procesos no deberían acceder a dicha parte del código.

- *Interbloqueos.* Suponga que dos procesos P1 y P2 necesitan acceder a dos recursos R1 y R2. Cada uno necesita acceder a los dos recursos para completar sus tareas. Si en un momento dado el S.O. asigna R1 a P1 y R2 a P2, cada proceso podría esperar a que el otro desocupe el recurso asignado, a fin de cumplir con el requerimiento. Esta situación es denominada *interbloqueo.*
- Inanición. Es una situación en la que un proceso se podría quedar esperando un recurso por un tiempo indefinido. Suponga que tres procesos P1, P2 y P3 requieren acceder a un recurso R. Si el recurso está ocupado por el proceso P1, los procesos P2 y P3 estarían bloqueados. Una vez P1 libera el recurso (sale de su sección crítica) podría ocurrir que el sistema operativo le otorgue el uso de R a P2, lo que implica que P3 seguiría bloqueado. Pero puede darse una situación en la que P1 requiera de nuevo el recurso R, lo que implica que, si el sistema operativo le concede su uso, P3 se quedaría indefinidamente en estado de bloqueado.

2. Procesos que cooperan mediante recursos compartidos.

   En este caso, los procesos que cooperan mediante variables compartidas pueden enfrentarse al problema de la *coherencia de la memoria.* Por tanto, es necesario que los procesos verifiquen la integridad de la información almacenada en dichas variables compartidas. Dado que la información es almacenada en un recurso (memoria), los problemas de exclusión mutua, interbloqueos e inanición están presentes.

3. Procesos que cooperan mediante comunicaciones.

   Los problemas con procesos que cooperan mediante sistemas de comunicaciones, aparecen cuando: dos procesos están esperando un mensaje desde el otro (interbloqueo), y cuando por ejemplo, dos procesos P2 y P3 desean comunicarse con P1, pero podría ser que P2 y P1 queden en un ciclo repetitivo de intercambio de información, dejando a P3 en estado de inanición.

2. Semáforos.

Los semáforos son variables que almacenan un valor entero y sobre la que se pueden realizar tres operaciones básicas: inicialización, incremento y decremento. Las tres operaciones son atómicas, lo que implica que durante su ejecución el sistema no interrumpirá el proceso. Los semáforos son una herramienta sencilla que permite implementar sicronización entre procesos, a fin de solventar algunos de los problemas generados por la concurrencia.

Las tres operaciones realizadas sobre un semáforo se resumen en:

1. Un semáforo debe ser inicializado, lo que implica que sea cargado con un valor no negativo.

2. Decremento. Esta operación se realiza mediante la función *semWait*. Si el valor del semáforo es negativo esta función será bloqueante, de lo contrario el proceso continuará su ejecución.

3. Incremento. Esta operación se realiza mediante la función *semSignal*. Si el valor del semáforo es menor o igual a cero, un proceso bloqueado por un *semWait* será desbloqueado.

Las funciones para implementar semáforos genéricos con POSIX son:

- int sem_init(sem_t *sem, int pshared, unsigned int value);
- int sem_wait(sem_t * sem);
- int sem_post(sem_t * sem);
- int sem_destroy(sem_t * sem);

Ejemplo de uso de semáforos con hilos en C.

Listing 1: Librerías

```
1
2  #define MAX_PROCESOS 2
3
4  /* Variables para semaforos*/
5  sem_t *semaforo;
6
7  struct datos_tipo
8  {
9          int dato;
10         int p;
11 }
12
13
14 * proceso(void *datos)
15 {
16         struct datos_tipo *datos_proceso;
17         datos_proceso = (struct datos_tipo *) datos;
18         int a, i, j, p;
19 sem_wait(semaforo);
20         a = datos_proceso -> dato;
21         p = datos_proceso -> p;
22
23         for(i=0; i <= p; i++)
24         {
25                 printf(" %i ",a);
26         }
27
28         fflush(stdout);
29         sleep(1);
```

```
30          for(i=0; i<=p; i++)
31          {
32                  printf("- ");
33          }
34          fflush(stdout);
35  sem_post(semaforo);
36  }
37
38
39  main()
40  {
41  int error;
42  char *valor_devuelto;
43
44  /* Variables para hilos*/
45  struct datos_tipo hilo1_datos, hilo2_datos;
46  pthread_t idhilo1, idhilo2, idhilo3, idhilo4;
47
48  hilo1_datos.dato = 1;
49  hilo2_datos.dato = 2;
50  hilo1_datos.p = 10;
51  hilo2_datos.p = 5;
52
53  semaforo= sem_open("semaforo_name", O_CREAT, 0700, MAX_PROCESOS);
54
55  error=pthread_create(&idhilo1, NULL, (void *)proceso, (void *)(&hilo1_datos));
56  if (error != 0)
57  {
58          perror ("No puedo crear hilo");
59          exit (-1);
60  }
61
62  error=pthread_create(&idhilo2, NULL, (void *)proceso, (void *)(&hilo2_datos));
63  if (error != 0)
64  {
65          perror ("No puedo crear thread");
66          exit (-1);
67  }
68  pthread_join(idhilo2, (void **)&valor_devuelto);
69  pthread_join(idhilo1, (void **)&valor_devuelto);
70
71  sem_close(semaforo);
72  sem_unlink("semaforo_name");
73
74  }
```

3. Taller

Para el presente taller se requiere realizar un resumen **a mano** con **letra imprenta** que presente los problemas concurrencia entre procesos, basado en el apartado anterior y la bibliografía del curso. El resumen debe mostrar los problemas descritos mediante texto y **dibujos**. Adicionalmente explique brevemente los elementos de hardware presentes en el sistema para soportar la exclusión mutua (interrupt disabling, instrucciones especiales). Finalmente explique la forma en que funciona un semáforo, explicando el mecanismo mediante el cual implementa la exclusión mutua y soluciona el problema del productor-consumidor.

3.1. Consideraciones.

- Emplea una hoja carta por ambas caras para realizar el resumen.
- Se aconseja el uso de tinta negra para la letra. Los dibujos pueden llevar color. Si lo desea, puede emplear dos columnas por página.
- Hacia el final del resumen explica brevemente el funcionamiento del ejemplo y modifíquelo para operar con N hilos. **No es necesario enviar el código.**
- Un resumen organizado, coherente, con información suficiente y evidencia de consultas bibliográficas, puntuará 2 décimas (0.2) sobre el segundo parcial del curso.