Sistemas Operativos II

Práctica 2 - Sincronización de procesos con semáforos

Objetivo

Conocer el funcionamiento de los semáforos como solución al problema de las carreras críticas y desarrollar habilidades en su manejo.

1. Carreras críticas en el problema del productor-consumidor

- Estudia el problema del productor-consumidor visto en clase.
- Programa el productor y el consumidor para comprobar que se pueden presentar **carreras críticas**. Puedes aumentar la probabilidad de que ocurran **haciendo que la región crítica dure más tiempo**, por ejemplo, con llamadas a la función *sleep()* oportunamente situadas. Añade al código salidas que muestren como evolucionan el productor y el consumidor, y usa comentarios para indicar donde deben situarse los *sleep()* para que se produzcan carreras críticas con alta probabilidad.
- Los códigos del productor y el consumidor deben ser dos programas diferentes llamados *prod.c* y *cons.c* que se ejecuten en terminales distintos. Ten en cuenta lo siguiente:
  - Dado que el consumidor y el productor serán **dos procesos diferentes** (no hilos), es necesario definir zonas de memoria donde almacenar variables compartidas usando, por ejemplo, *mmap()*.
  - El buffer debe ser de **tipo char**, y funcionar como una cola **LIFO** (Last In First Out).
  - El tamaño del buffer definido como una consttante de los códigos, debe ser **N=8**.
  - Las funciones *sleep()* y *wakeup()* las puedes implementar con señales o con semáforos o sustituirlas por espera activa.
  - La función *produce_item()* debe generar una letra mayúscula aleatoria. Deberá, además, ir añadiendola a un string local que contendrá finalmente todos esos caracteres.
  - La función *insert_item()* debe colocar el carácter generado en el buffer.
  - La función *remove_item()* debe retirar del buffer el carácter que corresponda.
  - La función *consume_item()* debe añadir el carácter retirado del buffer a un string local donde los irá almacenando a medida que se leen.

2. Uso de semáforos para resolver las carreras críticas

- Usando como base los códigos del ejercicio anterior, programa la solución vista en clase usando semáforos, y teniendo en cuenta lo siguiente:
  - Las variables semáforo se declaran como punteros a estructuras de tipo *sem_t*. Por ejemplo: sem_t *vacias;

- Para crear un semáforo usa la función *sem_open()*. Un ejemplo de uso es:
  vacias = sem_open("VACIAS", O_CREAT, 0700, N);
- Uno de los procesos debe crear el semáforo e **inicializarlo** con la función *sem_open()*. Los procesos que usen ese semáforo posteriormente deben **abrirlo** sin inicializarlo usando también la función *sem_open()* del siguiente modo:
  vacias = sem_open("VACIAS", 0);
- Para **cerrar** los semáforos y el espacio de memoria compartida, usa las funciones *munmap()* y *sem_close()*.
- Las funciones de **manejo** de los semáforos de POSIX son *sem_wait()* y *sem_post()*.
- Sustituye el lazo infinito por otro con **60 iteraciones** (definido como una constante de los códigos) para no tener que abortar el programa en cada ejecución.
- Añade llamadas a la función *sleep()* oportunas tanto en el productor como en el consumidor, y fuera de la región crítica, para forzar situaciones en las que vayan a diferentes velocidades, de modo que eventualmente el buffer se pueda llenar o vaciar. En la versión entregable, esas llamadas a *sleep()* deben ser de valores aleatorios entre 0 y 3 segundos.
- Al final deben **eliminarse** los semáforos usando la función *sem_unlink()*. Si no se usa esta función, los semáforos quedarán activos en el kernel. Una buena práctica es eliminar los semáforos antes de crearlos al principio del programa.
- Para **compilar** quizás se debe incluir la opción de compilación *-pthread*.
- Los programas deben llamarse *prod_sem.c* y *cons_sem.c*.

3. Ejercicios opcionales

- Puedes realizar algunos de los ejercicios siguientes:

  1. Resuelve el problema con threads en lugar de procesos.
  2. Generaliza el código para un número arbitrario de procesos productores y consumidores que pueda ser seleccionado por el usuario.
  3. Generaliza el ejercicio para P procesos, de manera que el primero produzca carácteres que el segundo consumirá. A su vez, a cada carácter consumido por el segundo se le calcula el siguiente alfabéticamente (teniendo en cuenta que el siguiente a la Z será la A), y que constituirá el valor producido por el segundo para su consumo por el tercero. Similarmente, el tercero actuará como productor para el cuarto, y así sucesivamente hasta el P-ésimo, que actuará como consumidor del (P-1)-ésimo y no actuará como productor de ningún otro. Los P procesos deben actuar concurrentemente en la mayor medida posible.

Entrega

- Se deben entregar dos informes breves (de como mucho 5 páginas) incluyendo información sobre como ejecutar los códigos, comentarios sobre su funcionamiento y las conclusiones obtenidas, tanto para el apartado 1 como para el 2. Adicionalmente deben subirse los códigos en C de ambos ejercicios con comentarios exhaustivos.
- En su caso, para cada uno de los ejercicios opcionales, debe incluirse la misma información que la indicada para los obligatorios.
- La fecha de entrega es la indicada en el Campus Virtual.