Sistemas Operativos I. Enxeñería Informática. Curso 2007/08.

Práctica 2: Concurrencia de procesos: Productores/Consumidores.

En esta práctica se tratará de resolver el problema de los productores/consumidores utilizando semáforos como medio de sincronización y utilizando memoria compartida en los elementos comunes a los distintos procesos.

Cada productor escribirá por pantalla el instante en el que ha introducido un ítem en el búffer, el ítem introducido, y contenido de dicho búffer. Asimismo, cada consumidor indicará el instante en el que obtiene un ítem del búffer compartido, el ítem obtenido, y el estado del búffer.

Para la creación del búffer, se utilizarán las funciones para el manejo de memoria compartida (shmget, shmat, shmdt, shmctl,…). Para el manejo de semáforos, se deben utilizar, entre otras, las funciones: semget, semop y semctl.

Descripción general:

La práctica consistirá en la creación de 4 programas:

Inicializa.c: Crea los semáforos necesarios para la sincronización de productores y consumidores. Además **crea el búffer** en el que éstos añadirán o extraerán items. El tamaño del búffer vendrá indicado por el parámetro –items. Por defecto se asumirá que el búffer tiene capacidad para 10 items (cada ítem es un carácter). En ningún caso, el tamaño del búffer podrá exceder el valor indicado por una constante MAX_ITEMS.

```
$inicializa –items 10
```

Productor.c: Crea *p* procesos productores (parámetro -*p*). Cada proceso creado simula el funcionamiento de un productor, produciendo items (1 de cada vez) y añadiéndolos al búffer compartido. Un parámetro “veces” indica el número de elementos que cada productor añadirá al búffer.

```
$productor –p 3 –veces 30
```

Consumidor.c: Crea *c* procesos consumidores (parámetro -c). Cada proceso extrae items del búffer (1 de cada vez) y muestra su contenido. Un parámetro “veces” indica el número de elementos que cada consumidor quitará del búffer.

```
$consumidor  -c 7 –veces 30
```

Finaliza.c: Una vez que todos los productores y consumidores hayan finalizado su trabajo, se encarga de liberar todos los recursos utilizados (creados por “inicializa.c”). Esto es, los semáforos creados, y el búffer compartido.

```
$finaliza
```

Productores/consumidores

Los programas productor.c y consumidor.c se encargarán respectivamente de crear *p* procesos productores y *c* procesos consumidores. Para la creación de un número concreto de productores o consumidores, puede usarse la función *fork()*, como se muestra a continuación:

```
main() {
   pid_t pid;
   int i;
   int np=10;  //número de procesos a crear.
   for (i=0;i<np;i++) {
      pid=fork();
      if (pid == 0) break;  //un hijo
   }
   printf("\n el padre del proceso %d es %d",getpid(),getppid());
}
```

Cada proceso productor o consumidor, en lugar de un bucle infinito, se ejecuta N veces (en función del parámetro *–veces*) ejecutando el código asociado a un productor o consumidor respectivamente.

Cada vez que el productor "produzca" un ítem, se obtendrá un carácter del siguiente modo: el primer productor siempre dará lugar a una 'A', el segundo a una 'B', el tercero a una 'C', y así sucesivamente (se puede asumir que no habrá más de 20 productores distintos). Esto es, si hay 3 productores, el 1º escribirá siempre 'A's, el 2º escribirá 'B's y el 3º 'C's.

Una vez que cada proceso accede a la sección crítica, las funciones anhadirItem() y obtenerItem() dejan al productor/consumidor en espera un tiempo aleatorio entre 1 y 5 segundos (*sleep)*. Además, se imprime la hora a la que empieza y termina de producir o consumir el correspondiente proceso. También se muestra el estado del búffer (número de items que contiene actualmente y los distintos items).

Todas las salidas de información a pantalla se realizarán mediante la llamada al sistema *write*.

```
char produceItem(int id_proc) {
    ….
    ….
}

void anhadirItem(int id_proc, char item) {
   char mensaje [MAXCADENA];
   time_t t;
   pid_t p;
   p=getpid();
   t=time(NULL);
   sprintf (mensaje,"productor %d (%u): Comienzo anhadir item %c
                    a las %s",id_proc,p,item,ctime(&t));
   write (STDOUT_FILENO,mensaje,strlen(mensaje));
   /*esperar tiempo aleatorio entre 1 y 5 segundos*/

   anhadeCola(item);   //añade el item producido al búffer compartido
   mostrarBuffer();

   sprintf (mensaje,"productor %d (%u): termino de ahadir %c a las %s"
                  ,id_proc,p,item,ctime(&t));
   write (STDOUT_FILENO,mensaje,strlen(mensaje));
}
```

Búffer compartido: Cola circular.
El búffer que productores y consumidores comparten se implementará como una cola circular.

```
Struct tColaChar {
   char items[MAX_ITEMS];  //vector de longitud indicada en "inicializa.c"
   int maxNum, num, ppio, final;
}
Typedef struct tColaChar tBuffer
```

El número máximo de items que caben en el búffer (maxNum) se le indica al programa "inicializa"; nótese que ha de cumplirse que maxNum <=MAX_ITEMS. El valor num, indica el número de elementos que actualmente están almacenados en la cola. Los valores ppio y final indican el principio y final de la cola; al ser una cola circular, al incrementar ppio y final de debe hacer posteriormente una operación de módulo con "maxNum" *(ppio = (ppio +1) % maxNum).*

Al querer que el búffer (la cola de items) sea compartido por todos los procesos consumidores y productores, no es posible declararlo simplemente como:

tBuffer buffer;

pues esto daría lugar a que cada proceso tuviese su propio búffer (una variable de tipo tBuffer llamada buffer), pero no sería compartida entre todos ellos. Esto haría, p ej, que los procesos consumidores no podrían ver los items generados por los productores.

Para poder compartir el búffer, usaremos las funciones de la librería ipc para manejo de memoria compartida ***shmget, shmat, shmdt, shmctl*** del siguiente modo:

1. El proceso padre crea un área de memoria compartida con tamaño suficiente para almacenar el búffer (sizeof(tBuffer)), usando la llamada *shmget*(). Shmget devuelve un número (shmid) que será usado como identificador de la zona de memoria compartida que se ha creado. A la función shmget() debe pasársele una clave (key), que puede obtenerse mediante la función ftok().

   key_t key = ftok("/tmp",'B');
   shmid = shmget(key, …..);

2. Usando el identificador que devuelve shmget, cada proceso podrá obtener un puntero a la zona de memoria compartida (el búffer), y a través de dicho puntero acceder a dicho búffer para añadir o quitar items.

   tBuffer *pbuffer;
   pbuffer=(tBuffer *) shmat(shmid,0,0);

3. A partir de ahí, cada proceso podrá usar pbuffer como si fuese un puntero a tBuffer "normal", pero que se comporta ahora como una estructura compartida.
4. Al finalizar cada proceso productor o consumidor, se "desasigna" el puntero pbuffer usando *shmdt*.

5. Por último, el programa "finaliza.c" destruye el área de memoria compartida usando *shmctl*.

Utilización de semáforos:
La implementación propuesta precisa del uso de 3 semáforos: *vacío, lleno y búffer*.
La creación de semáforos se hará mediante la llamada al sistema *semget()*. Se pueden implementar las operaciones básicas para la manipulación de cada semáforo haciendo uso de la llamadas *semop() y semctl()*. Finalmente la llamada *semctl()* permitirá liberar dicho recurso. Más concretamente:

- semget: Permite obtener un grupo de k semáforos que será identificado por un identificador semid.
- semop: Opera sobre uno de los semáforos (i) del grupo obtenido con semget (grupo identificado por *semid*), y permite entre otras, hacer las operaciones P y V sobre alguno de los semáforos del grupo.
- semctl: se encarga de liberar un grupo de semáforos identificado por *semid* cuando ejecuta el comando IPC_RMID. También permite establecer un valor a un semáforo determinado, cuando a través de semctl se ejecuta el comando SETVAL.

NOTA:
Al igual que shmget(), semget() recibe como parámetro un valor key_t key, que puede obtenerse previamente con la función ftok(). Deberemos asegurarnos de que la clave que devuelva ftok sea distinta que la obtenida para crear el búffer_compartido, y la misma para todos los procesos que han de "compartir" el mismo semáforo.

Básicamente semget() permite obtener un grupo de k semáforos. En nuestro caso será necesario simplemente disponer de un grupo de k=3 semáforos (vacío, lleno y búffer). Como cada uno de estos semáforos ha de ser accedido utilizando un número {0,1,2}, se aconseja definir constantes para identificar cada semáforo de modo que:

- el semáforo "*vacío*" se identificará con un 0.
- el semáforo "lleno" se identificará con un 1.
- el semáforo "mutex" se identificará con un 2.

Más información y comandos de interés.
Ver manual en línea para las funciones comentadas.
Ipcs e ipcrm para ver y eliminar recursos compartidos (semáforos o segmentos de memoria compartida) desde la línea de comandos.

FORMA DE ENTREGA.
Al igual que en la práctica 1, se realizará la defensa de la práctica en el aula.

FECHA LÍMITE DE ENTREGA.
Semana del 14 al 18 de Enero de 2008. Cada alumno ha de defenderla dentro del grupo de prácticas al que pertenece (Lunes, Martes, Miércoles o Viernes).

Problema de productores/consumidores (búffer limitado).

Var

Semáforo mutex = 1 // exclusión mutua en acceso a búffer
Semáforo vacio = n; // número de celdas vacías en el búffer
Semáforo lleno = 0; //número de celtas llenas en el búffer.

Productor	Consumidor

Var it: t_item
Begin
  Repeat
  Producir(it);
  P(vacio);
  P(mutex);
  Añadir it al búffer;
  V(mutex);
  V(lleno);
  Until false
End;
	Var it: t_item
	Begin
	  Repeat
	  P(lleno);
	  P(mutex);
	  It ← elemento del búffer;
	  V(mutex);
	  V(vacio);
	  Consumir (it);
	  Until false;
	End