Sistemas Operativos I. Enxeñería Informática. Curso 2008/09.

Práctica 2: Concurrencia de procesos: Problema de los filósofos cenando.

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

> Cada filósofo imprimirá por pantalla información relativa a los instantes siguientes: ***a)***el instante en el que comienza a comer, ***b)***el momento en el que deja los tenedores, así como ***c)*** el instante en el que se pone a pensar.

Nótese que para la sincronización de los distintos filósofos que puedan estar sentados a la mesa, es necesario usar una serie de estructuras compartidas (número de filósofos y estado de los mismos) y varios semáforos.

El número **N** de filósofos, así como el estado de los filósofos, se guardará en un segmento de memoria compartida, por lo que será necesario utilizar las funciones para el manejo de memoria compartida (shmget, shmat, shmdt, shmctl,…). Nótese que un si hay N filósofos deberemos compartir N+1 enteros: Los N primeros permiten almacenar el estado de los filósofos (i=0..N-1) y el último entero será el propio valor "N".

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 3 programas:

inicio.c: Crea los semáforos necesarios para la sincronización de los filósofos. Además **crea la zona de memoria compartida**, y la inicializa convenientemente.

$inicio –nf 4

filosofo.c: Crea los ***N*** procesos filósofos. Cada proceso creado simula el funcionamiento de un filósofo. Un parámetro "times" indica el número de veces que el filósofo desea comer. Una vez que haya conseguido comer "times" veces, el proceso terminará.

$filosofo –times 30

final.c: Una vez que todos los filósofos hayan finalizado de comer, se encarga de liberar todos los recursos utilizados (creados por "inicia.c"). Esto es, los semáforos creados, y la/s zona/s de memoria compartida.
$final

Filósofos

El programa filosofos.c se encargará de crear *N* procesos filósofos. Para la creación de un número concreto de filósofos, puede usarse la función *fork()*, como se muestra a continuación:

```
main() {
   pid_t pid;
   int i;
    int np= N;   //número de procesos a crear ( ##obtener N!!##
   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 filósofo en lugar de un bucle infinito, se ejecuta un número fijo de veces (en función del parámetro *–times*) ejecutando su código asociado.

Siempre que un filósofo se ponga a pensar, o comience a comer, se quedará en espera un tiempo aleatorio entre 1 y 4 segundos (ver funciones *sleep, srand, rand, urand)*. Además, se imprime la hora a la que trata de tomar tenedores, la hora a la que comienza a comer, y la hora a la que deja los tenedores. En cada uno de esos casos, también mostrará por pantalla, el estado actual de todos los comensales: [H]ambriento, [C]omiendo, [P]ensando.

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

```
void come(int id_proc) {
   char mensaje [MAXCADENA];
   time_t t;
   pid_t p; p=getpid();
   t=time(NULL);
   sprintf (mensaje,"filósofo %2d (%6u): comienza a comer
                        a las %s",id_proc,p ,ctime(&t));
   write (STDOUT_FILENO,mensaje,strlen(mensaje));
   mostrarEstados();

   /*esperar tiempo aleatorio entre 1 y 5 segundos*/
}
```

Zona compartida: Estado de filósofos y número de filósofos (N).
El estado de los filósofos, y el almacenamiento la variable N, se implementará siguiendo el siguiente tipo de datos.

```
Struct tBuff {
   int estados[MAXN];   //vector de estados [pensando=0, hambriento = 1, comiendo = 2]
   int N;               //valor indicado en "inicializa.c"
}
Typedef struct tBuff tMemCompartida
```

Donde MAXN será una constante tal que MAXN > N (p.ej MAXN=20)

Al ser necesario que la zona de memoria compartida sea común a todos los procesos filósofos, no es posible declararlo simplemente como:

tMemCompartida comun;

pues esto daría lugar a que cada proceso tuviese su propia variable de tipo tMemCompartida llamada *comun*, que en definitiva, no sería compartida entre todos ellos, y por lo tanto un proceso no conocería el estado de los demás!!!

Para poder compartir el estado en una estructura común, 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 sincronizar los N procesos "filósofos", 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().

   Ej:

   ```
   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 (la zona común), y a través de dicho puntero acceder a la zona compartida para comprobar o modificar los estados.

   ```
   tMemCompartida *pcomun;
   pcomun =(tMemCompartida *) shmat(shmid,0,0);
   ```

3. A partir de ahí, cada proceso podrá usar *pcomun* como si fuese un puntero a tMemCompartida "de los de siempre", pero que se comporta ahora como una estructura que comparte memoria.
4. Al finalizar cada proceso filósofo, se "desasigna" el puntero *pcomun* usando *shmdt*.

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

Utilización de semáforos:

La implementación propuesta requiere el uso de N+1 semáforos.
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.

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.

Las prácticas se entregarán por email (antonio.fari.so@gmail.com o jcasanova@udc.es) antes de proceder a su defensa. Debe enviarse un fichero comprimido tar.gz conteniendo: a) código fuente y b) fichero Makefile (se compilará tecleando $make). El asunto del mensaje deberá ser:

[SO1P2]::[Nombre1erIntegrante][Nombre2ºIntegrante] //de ser sólo 1 alumno à 1 nombre.
El nombre del fichero comprimido estará formado por "p2" y los códigos de usuario concatenados: **p2_infxxx00-infyyy00.tar.gz**

FECHA LÍMITE DE ENTREGA.
Semana del 12 al 16 de Enero de 2008. Cada alumno ha de defenderla dentro del grupo de prácticas al que pertenece (Lunes, Martes, Miércoles o Viernes).

Anexo. FILÓSOFOS CENANDO

Estructuras compartidas

```
N                                   /*  Número total de semáforos */
integer estado[N];                    /*  array de estados para cada filósofo  */
semáforo mutex=1;                   /*  semáforo para acceso exclusivo a las secciones críticas  *
semáforo S[N];                       /*  un semáforo para el posible bloqueo de cada semáforo  *
```

Constantes

```
IZQUIERDA(i)      (i-1)%N        /*  vecino de la izquierda  */
DERECHA(i)        (i+1)%N        /*  vecino de la derecha  */
PENSANDO           0
HAMBRIENTO        1
COMIENDO           2
```

filósofo(i) /* código para cada filósofo i; i=0...N-1 */

```
begin
   repeat
      piensa(i);
      toma_cubiertos(i);
      come(i)                              /*  toma ambos cubiertos o se bloquea  */
      deja_cubiertos(i);
   until false;
end;
```

toma_cubiertos(i)

```
begin
   P(mutex);
   estado[i]= HAMBRIENTO;
   test(i);                                   /*  intenta tomar los dos cubiertos  */
   V(mutex);
   P(S[i]);                                 /*  se bloquea si no tomó los dos
cubiertos  */
end;
```

deja_cubiertos(i)

```
begin
   P(mutex);
   Estado[i]=PENSANDO;                    /*  Ha terminado de comer  */
   test(IZQUIERDA);                        /*  Despierta al vecino de la izquierda
o de la
   test(DERECHA);                           derecha, si estaba bloqueado, y
puede comer  */
   V(mutex);
end;
```

test(i)

```
begin
   if (estado[i]=HAMBRIENTO and
      estado[IZQUIERDA] != COMIENDO and
      estado[DERECHA] != COMIENDO) then
         begin
            estado[i]=COMIENDO;
            V(S[i]);
         end;
end;
```