SISTEMAS OPERATIVOS 1. Enxeñería Informática. Curso 2006-2007

Práctica 2: Concurrencia de procesos. Lectores/escritores.

Implementar semáforos generales a partir de semáforos binarios a partir de la implementación de *Kearns* y utilizarlos para solucionar el problema de los lectores/escritores en su 1ª variante. Cada proceso indicará en pantalla el *item* que acaba de escribir o leer y el instante en que lo ha leído o escrito en la sección crítica. Comprobar el funcionamiento ejecutando varios lectores o escritores desde distintas ventanas.

Comentarios:

Se deben codificar los siguientes programas:

- *inicializa.c:* Recibe como parámetros el número de lectores y escritores. Crea los semáforos (ficheros asociados) y los inicializa a los valores correspondientes.
- *lectores.c:* Recibe el número de lectores como parámetro y los crea. Cada proceso simula el comportamiento de un lector indicando en pantalla cuándo comienza y termina de leer.
- *escritores.c:* Idem para los escritores.

Ejemplo:

$ *inicializa n 5 e 3* – crea los ficheros necesarios para los semáforos requeridos en la concurrencia de 5 lectores y 3 escritores.

$ *lectores 5* – crea 5 procesos lectores desde un intérprete de comandos.

$ *escritores 3* – crea 3 procesos escritores.

Cada lector y cada escritor, en lugar de un bucle infinito, se ejecuta n veces (n predeterminado a 50). Para la creación de un número concreto *np* de lectores o escritores, una solución posible es la siguiente:

```
main ()
{
pid_t pid;
int i;
int np=10;
for (i=0; i<np; i++) {
    pid = fork();
    if (pid==0)
        break;
}
printf("\nEl padre del proceso %d es %d", getpid(), getppid());
}
```

Una vez que cada proceso accede a la sección crítica, las funciones leer() y escribir() dejan al lector/escritor en espera un tiempo aleatorio e imprimen la hora a la que empieza y termina de leer o escribir el correspondiente proceso. Las salidas de la información a pantalla se realizarán con la llamada al sistema *write*, como en el código que se muestra a continuación:

```
void leer(int i)
{
 char mensaje [MAXCADENA];
 time_t t;
 pid_t p;

 p=getpid();
 t=time(NULL);
 sprintf (mensaje,"lector %d (%u): Comienzo a leer a las %s",i,p,ctime(&t));
 write (STDOUT_FILENO,mensaje,strlen(mensaje));
 /*esperar tiempo aleatorio entre 0 y 5 segundos*/
 sprintf (mensaje,"lector %d (%u): termino de leer a las %s",i,p,ctime(&t));
 write (STDOUT_FILENO,mensaje,strlen(mensaje));
}
```

Deben implementarse semáforos generales a partir de semáforos binarios según la implementacion de *Kearns*. Los semáforos binarios se implementarán haciendo uso de que en UNIX la llamada *open*, cuando intenta crear un fichero en modo exclusivo, lo hace de manera atómica. Así, las operaciones binarias *Pb* y *Vb* quedarían:

```
Pb (char * s)
{
 int df
  while ((df=open(s,O_CREAT | O_EXCL))==-1);
  close (df);
}

Vb(char * s)
{
 unlink (s);
}
```

La implementación de los semáforos generales debe incluir:

- La definición del tipo semáforo general.
- Las operaciones
    - InicializarSemaforo: Da al semáforo el valor inicial, e inicializa los semáforos binarios auxiliares a su valor correspondiente.
    - Operacion P
    - Operacion V

IMPORTANTE

Debe además tenerse en cuenta que:

- La llamada *open* no funciona de manera atómica en sistemas de ficheros *NFS*; por tanto, los ficheros que implementan los semáforos binarios deben crearse en un directorio LOCAL de la máquina donde se trabaja (p.e. /tmp)
- Los valores de las variables de la implementación de *Kearns* deben ser compartidos por los procesos que usan el semáforo. Estos valores compartidos se almacenarán en un fichero. Toda la entrada/salida se hará con las llamadas al sistema *open, read, write, lseek y close*; además, el fichero donde se guardan estos valores del semáforo no debe quedar abierto entre operaciones P y/o V sobre el semáforo correspondiente.

FORMA DE ENTREGA Como en la práctica anterior.

FECHA LÍMITE DE ENTREGA: Semana de 8 al 12 Enero de 2007.

Implementación de Kearns

```
type
    semáforo = record
       mutex , delay:  semáforo_binario
       valor, contador_delay: integer
   end;
 /* val_ini delay=1, mutex=1, contador_delay=0*/

procedure P(var s: semáforo)
begin
   Pb(s.mutex)
   s.valor = s.valor -1;
   if s.valor < 0 then
      begin
         Vb(s.mutex);
         Pb(s.delay);
         Pb(s.mutex);
         s.contador_delay = s.contador_delay -1;
         if s.contador_delay > 0 then Vb(s.delay)
      end;
   Vb(s.mutex)
end;
```