SISTEMAS OPERATIVOS I

Segundo curso Ingeniería Informática. Curso 2003-2004

Práctica 3: Implementación de semáforos con espera inactiva. Utilización de semáforos para snicronización de procesos concurrentes.

Implementar un semáforo con espera inactiva y utilizarlo para sincronizar el acceso de varios procesos concurrentes a su sección crítica como en la práctica anterior. Para ello utilizaremos los programas de la práctica anterior ligeramente modificados:

- Programa p3a.c. Es el programa que se encarga de crear e inicializar el semáforo. Su pseudocódigo, excluyendo el control de errores, puede verse a continuación:

```
Crear e inicializar el semaforo;
Esperar a que terminen los procesos concurrentes;
Eliminar el semaforo;
```

- Programa p3b.c. Es el programa del que ejecutaremos varias instancias en distintas terminales, y que se pretende sincronizar mediante el semáforo. Su pseudocódigo es:

```
Obtener referencia al semaforo;
loop
  seccion de entrada;
  seccion critica;
  seccion de salida;
  seccion restante;
end loop;
```

  Nótese que ahora las secciones de entrada y salida son operaciones sobre el semáforo.

Comentarios

- **Implementación del semáforo general con espera inactiva**. La implementación del semáforo general con espera inactiva consta de

1. Una variable entera para almacenar el valor del semáforo.
2. Una lista de los identificadores de los procesos que esperan en ese semáforo (no han podido completar la operación P).
3. un semáforo binario auxiliar para garantizar la atomicidad de las operaciones P y V

Para realizar la práctica es necesario llevar a cabo las siguientes tareas

- Declaración del tipo del semáforo, que incluya las entidades antes mencionadas

InicializarSemáforo Recibe como parámetro el valor inicial del semáforo y devuelve un puntero al semáforo ya creado e inicializado. Inicializa tambien el semáforo binario auxiliar que garantiza la atomicidad de las operaciones P y V.

Operaciones P y V Reciben una referencia al semáforo y puede optarse por cualquiera de las implementaciones con espera inactiva vistas en teoría (tanto la implementación que comprueba si la lista de procesos en espera está vacia como la que utiliza el valor negativo del semáforo para llevar cuenta de los procesos en espera).

EliminarSemaforo Recibe como argumento una referencia al semáforo y lo elimina del sistema.

La implementación de los semáforos binarios mediante ficheros de acceso exclusivo, así como las funciones bloquear y despertar se suministran en el fichero semaforos.h disponible en la siguiente url:

`http://www.dc.fi.udc.es/os/~afyanez/Practicas/sources/semaforos.h`

En dicho fichero se incluye:

- Declaración del tipo semaforo_binario como un array de caracteres en el que se almacena el nombre del fichero que implementa el semáforo binario.

Pb(semaforo_binario s) La operacion P binaria sobre el semáforo binario.

Vb(semaforo_binario s) La operacion V binaria sobre el semáforo binario.

InicializarSemaforoBinario Recibe 4 parametros, el semaforo binario a inicializar, el nombre del fichero que queremos que lo implemente, el directorio donde queremos que resida dicho fichero y el valor inicial del semaforo binario (0 ó 1). Devuelve -1 en caso de error y la variable er-

ror_semaforo contiene una descripción del error.

despertar (pid_t pid) Despierta (saca de la espera) el proceso cuyo pid se le pasa como argumento

bloquear (pid_t pid, s) Bloquea el proceso cuyo pid se le pasa como argumento y hace una Vb sobre el semáforo binario s. Hay que tener en cuenta, que si en la implementación del semáforo general usamos un semaforo auxiliar mutex para garantizar la atomicidad de las operaciones P y V y en la operación P lo colocásemos de la siguiente manera:

```
P
begin
  Pb(mutex);
  if S>0 then S:=S-1
  else begin
    meter identificador en lista de procesos en espera;
    bloquear proceso;
    end
  Vb(mutex);
end
```

nos encontariamos con el problema de que el proceso una vez que se bloquea no puede liberar la exclusión mutua para que otro proceso pueda hacer una operación V. Esto se soluciona haciendo la implementación como se ve a continuación, donde ahora la operación bloquear, despues de bloquear el proceso libera la exclusión mutua. Es esta última función bloquear la que se ha incluido en el fichero semaforo.h

```
P
begin
  Pb(mutex);
  if S>0 then begin
    S:=S-1;
    Vb(mutex)
  else begin
    meter identificador en lista de procesos en espera;
    bloquear proceso;
    end
end
```

Ha de tenerse tambien en cuenta que el valor del semáforo, así como la lista de procesos que esperan a completar la operación P son variables compartidas por los procesos que usan en semáforo, por lo que deben implementarse en memoria compartida.
El nombre del fichero que implementa el semáfor binario debe generarse en la función InicializarSemáforo, teniendo en cuenta:

  - El servidor NFS que exporta los directorios $HOME puede que no garantize la atomicidad de la apertura en modo exclusivo, por lo que dicho fichero ha de residir en el directorio /tmp
  - Es posible que más de una persona haga la práctica a la vez en la misma máquina, por lo cual dicho nombre no debe ser lo suficientemente común como para que otro usuario lo utilize (por ejemplo, puede incluir, como parte del nombre, el pid o el uid del proceso que lo crea)

- **Memoria compartida** Se necesita memoria compartida para implementar el semáforo. Se utilizará el fichero *sharedmem.h* y las funciones descritas en la práctica anterior.
- **Secciones crítica y restante de los procesos**. Exactamente igual que en la práctica anterior.
  Ejemplo:

```
Proceso 3, pid 34721. Comienzo seccion critica: Mon Dec 1 15:12:13 2003
....
Proceso 3, pid 34721. Finalizacion seccion critica: Mon Dec 1 15:12:17 2003
```

Debe comprobarse el funcionamiento de la práctica

- Sin el semáforo
- Con el seáforo inicializado correctamente.
- Con el semáfor inicializado a otros valores..

FORMA DE ENTREGA Como en la práctica anterior.

FECHA DE ENTREGA VIERNES 16 ENERO 2004