SISTEMAS OPERATIVOS I

Segundo curso Ingeniería Informática. Curso 2004-2005

Práctica 2: Regiones críticas condicionales. Solucionar el problema de los filósofos comensales usando regiones críticas condicionales

Comentarios

Hay que codificar un programa que recibe como parámetro el número de filósofos que queremos simular. Dicho programa crea la región, inicilaiza las variables compartidas y crea un proceso para cada filósofo. Después espera a que terminen todos los filósofos y elimina la región del sistema. Una solución al problema de los filósofos cenando sería:

```
type
   FIL=record
     estado: array [0..N-1] of (PENSANDO, COMIENDO);
   end;
var
   F:shared FIL;

......
......


Procedure filosofo (i:integer)

begin
 repeat
  region F
     when (estado[izq(i)]<> COMIENDO AND estado[der(i)]<> COMIENDO)
       do
         estado[i]:=COMIENDO;
  come(i);
  region F
       do
         estado[i]:=PENSANDO;
  piensa(i);
 until FALSE
end;
```

En lugar de hacer que cada filósofo sea un blucle infinito, lo haremos de 50 iteraciones. Para crear los distintos filósofos

```
for (i=0; i< nfilosofos; i++)
  if (fork()==0) { /*crea un proceso hijo */
     filosofo (i); /*el hijo ejecuta el filosofo i */
     exit (0); /*importantisimo, si no el hijo continua creando procesos*/
     }
```

Las funciones de *pensar y comer* tienen que indicar el momento de inicio, esperar un tiemo aleatorio e indicar el momento de finalización. Para evitar que se mezcle mucho la salida de los distintos procesos toda la salida por pantalla deberá ser con la llamada al sistema *write*

Así el **pseudocódigo** de *piensa* podría ser

```
piensa()
{
  generar cadena "filosofo i, comienza a pensar a las ...."
  imprimir cadena con write;
  esperar tiempo aleatorio
  generar cadena "filosofo i, termina de pensar a las ...."
  imprimir cadena con write;
}
```

Para usar regiones críticas debe hacerse lo siguiente

I Utilizar el include que se suministra, "region.h", disponible en

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

e incluirlo en el código fuente

```
#include "region.h"
```

II Declarar las variables que se quieren compartir en una estructura (el nombre de la estructura no es significativo)

```
struct COMPARTIR {
     /* lista de variables a compartir */
       ...
};
```

III Declarar un puntero al tipo REGION (definido en "region.h") que es el que nos va a permitir acceder a la región.

IV Usar las funciones que se suministran en el include "region.h".

- **REGION * rcreate (size_t s)**
  Crea una región y devuelve un puntero a la región creada. Recibe como parámetro el tamaño de la estructura que se comparte.

- **int rdestroy (REGION *r)**
  Destruye una región creada con rcreate y elimina los recursos que utilizaba.
- **REGION * rmalloc (size_t s)**
  Devuelve un puntero a una región ya creada.
- **int rfree (REGION *r)**
  Libera la memoria ocupada por una región sin eliminar los recursos ipc.
- **int Region (REGION * r, void (*sentencia)())**
  Ejecuta region r do sentencia. Sentencia es una función que devuelve void y recibe un puntero a la estrutura compartida declarado como (void *).
- **int RegionP (REGION *r, void (*sentencia)(), void * parm)**
  Ejecuta region r do sentencia. Sentencia es una función que devuelve void y recibe un puntero a la estrutura compartida declarado como (void *) y un parámetro adicional también declarado como (void *).
- **int RegionCond (REGION * r,int (*cond)(), void (*sentencia)())**
  Ejecuta region r when cond do sentencia. Sentencia es una función que devuelve void, y recibe un puntero a la estrutura compartida declarado como (void *). cond es una función que devuelve int y recibe como parámetro un puntero a la estructura compartida, tambien declarado como (void *).
- **int RegionCondP (REGION * r,int (*cond)(), void * parmcond, void (*sentencia)(), void * sentparm)**
  Igual que la anterior, pero cond y sentencia reciben un parámetro adicional
- **int RegionCond2 (REGION * r, int (*cond)(), void (*sentencia1)(), void (*sentencia2)())**. Forma segunda de la region crítica condicional. Ejecuta

```
region r do
  begin
    sentencia1;
    esperar (cond);
    sentencia2
  end;
```

- **int RegionCond2P (REGION *r,int(*cond)(),void *parmcond, void (*sentencia1)(), void *sent1parm, void (*sen-**

tencia2)(), void *sent2parm)
Igual que la anterior pero cond, sentencia1 y sentencia2 reciben parámetros adicionales parmcond, sent1parm y sent2parm.

En caso de error las funciones que devuelven un puntero devuelven NULL y las que devuelven entero devuelven -1. El entero regerrno indica qué error y reg_errlist[regerrno] nos da una descripción del error. Con sys_errlist[errno] o con perror podemos obtener más información del error.
Por ejemplo, si rcreate devuelve NULL y reg_errlist[regerrno] es "semget error" sabemos que rcreate falló porque no pudo obtener el semáforo. Si ahora vemos que perror indica "File exists", conocemos el motivo por el que semget falló.

Ejemplo de uso: Productor en el problema de los productores-consumidores

Declaración de la variable compartida

```
struct PC {
     TIPOITEM buff [N]; /*tipoitem estaria hecho con typedef*/
     in,out,cont: integer;
};
```

La función añadir al buffer

```
void AniadirBuffer (struct PC *p, TIPOTITEM * item)
{
   p->buff[p->in]=*item;
   p->in = (p->in +1) %N;
   (p->cont)++;
}
```

Esta construcción de AniadirBuffer es posible que produzca *warnings* dependiendo del compilador y de las opciones de compilacion ya que en "region.h" los punteros están declarados como (void *). Para evitar los *warnings* podríamos hacer

```
void AniadirBuffer (void * p1, void *p2)
{
   struct PC *p;
   TIPOITEM *item;

   p = (struct PC *) p1;
   item = (TIPOITEM *) p2;
```

```
   p->buff[p->in]=*item;
   p->in = (p->in +1) %N;
   (p->cont)++;
}
```

Como se ha usado una función que recibe un parámetro adicional (el item), la función condición también debe recibir otro parámetro aunque no lo use.

```
int PuedoAniadir (void * p1, void * nouso)
{
   struct PC * p = (struct PC *) p1;
   return (p->cont == N);
}
```

El código del productor (sin el control de errores):

```
Productor ()
{
  REGION *r;
  TIPOITEM item;
  int veces = 30; /* en lugar de un repeat until false */
  r=rmalloc (sizeof (struct PC)); /*si el proceso es hijo */
   /*del que crea la region, esto no seria necesario*/
   /*pues hereda el identificador r*/
  while (-- veces) {
     producir (&item);
     RegionCondP (r,PuedoAniadir, NULL, AniadirBuffer, &item);
     }
  rfree (r);
}
```

FORMA DE ENTREGA Como en la práctica anterior.

Fecha de entrega: 14 Enero 2005