Sistemas Operativos I. Enxeñería Informática. Curso 2009/10.

Práctica 2: Concurrencia de procesos: Problema dos lectores/escritores.

Nesta práctica trátase de resolver o problema dos **lectores/escritores con prioridade para os escritores**, utilizando semáforos como medio de sincronización e memoria compartida para os elementos comúns aos distintos procesos a sincronizar.

Neste caso, os procesos **escritor**, engadirán *nitems* caracteres a un búffer compartido no que caben size*Buff* items. Isto fará que os *nitems* caracteres que leven máis tempo no búffer serán eliminados. Ademais, cada *escritor* amosará por pantalla a seguinte información: (i) o momento no que comeza a escribir, (ii) o estado do búffer tras a escritura dos seus *items*, e (iii) o momento no que remata de escribir.

Os **lectores**, accederán ao búffer compartido e amosarán o seu contido. Adicionalmente, cada *lector* indicará por pantalla: (i) o momento no que comeza a ler, (ii) o estado do búffer compartido, e (iii) o momento no que remata de ler.

Para a resolución do problema deberanse usar unha serie de estructuras que serán compartidas por todos os procesos implicados mediante o uso de *segmentos de memoria compartida* (número de lectores que están a ler, número de escritores que están a escribir, búffer de memoria compartida) e tamén varios semáforos. Isto resolverase mediante o empleo de funcións para o manexo de memoria compartida (shmget, shmat, shmdt, shmctl,…), e de semáforos (semget, semop y semctl,...).

Descrición xeral:
Débense implementar 4 programas:

inicializa.c: Crea os **semáforos** necesarios para a sincronización de lectores/escritores, así como a **zona de memoria compartida**. Nos dous casos inicializándoos convintemente. O número de elementos (caracteres) que conterá o *búffer compartido* para una execución determinada (que en ningún caso será maior que unha constante MAX_ITEMS=1000) pódese indicar a través dun argumento opcional (o valor por defecto é **-items 10**). Este programa tamén se encarga de inicializar todos os items do buffer compartido co carácter '-';

Sintaxe: $inicializa [-items n]

exemplo de execución: $./inicializa -nitems 20

```
Conxunto de semaforos  con id=1179648, e X semáforos creado con éxito [OK]
Segmendo de memoria compartida, con id=6717488 e búffer de 20 items  [OK]
[PID:18884][NL=0][NE=0]::[num=20], Buffer[--------------------]
```

onde [PID] = id do proceso, [NL]= num lectores que están lendo, [NE] = num escritores que están actualmente escribindo, [num] = núm de items no buffer compartido, e [Buffer] é o contido do búffer compartido.

escritor.c: Este programa crea ***ne*** procesos fillos, cada un dos cales simulará o funcionamiento dun escritor. Un argumento "**ntimes**" indica cantas veces escribirá cada proceso no *búffer* compartido. Outro argumento "**nitems**" indica o número de caracteres que cada escritor engade, cada vez que lle toca escribir no *búffer compartido*.
O primeiro proceso fillo creado escribirá nitems veces a letra 'A', o segundo escribirá nitems veces a letra 'B', o terceiro escribirás letras 'C', etc. Unha vez que un proceso escritor teña conseguido escribir "ntimes", rematará a súa execución.
O proceso *pai* non rematará a súa execución ata que todos os seus fillos xa teñan rematado, e amosará o *status* co que cada proceso fillo remate.

Sintaxe: $escritor [-e ne -times ntimes -items nitems]

exemplo de execución: $./escritor -e 2 -times 3 -items 5

```
creados 2 procesos 'escritores' [1, pid 20458][2,pid 20459]
escritor 1 (20458): Comezo a escribir  A 5 veces: hora=Thu Dec  3 20:47:08 2009
        [PID:20458][NL=0][NE=2]::[num=20], Buffer[----------AAAAA]
escritor 1 (20458): Remato de escribir  A 5 veces: hora=Thu Dec  3 20:47:09 2009
escritor 2 (20459): Comezo a escribir  B 5 veces: hora=Thu Dec  3 20:47:09 2009
        [PID:20459][NL=0][NE=1]::[num=20], Buffer[----------AAAAABBBBB]
escritor 2 (20459): Remato de escribir  B 5 veces: hora=Thu Dec  3 20:47:11 2009
escritor 1 (20458): Comezo a escribir  A 5 veces: hora=Thu Dec  3 20:47:11 2009
        [PID:20458][NL=0][NE=1]::[num=20], Buffer[-----AAAAABBBBBAAAAA]
escritor 1 (20458): Remato de escribir  A 5 veces: hora=Thu Dec  3 20:47:14 2009
escritor 2 (20459): Comezo a escribir  B 5 veces: hora=Thu Dec  3 20:47:14 2009
        [PID:20459][NL=0][NE=2]::[num=20], Buffer[AAAAABBBBBAAAAABBBBB]
escritor 2 (20459): Remato de escribir  B 5 veces: hora=Thu Dec  3 20:47:17 2009
escritor 1 (20458): Comezo a escribir  A 5 veces: hora=Thu Dec  3 20:47:17 2009
        [PID:20458][NL=0][NE=1]::[num=20], Buffer[BBBBBAAAAABBBBBAAAAA]
escritor 1 (20458): Remato de escribir  A 5 veces: hora=Thu Dec  3 20:47:19 2009
PAI ESCRITOR: proceso fillo 20458 acaba con status 256
escritor 2 (20459): Comezo a escribir  B 5 veces: hora=Thu Dec  3 20:47:19 2009
        [PID:20459][NL=0][NE=1]::[num=20], Buffer[AAAAABBBBBAAAAABBBBB]
escritor 2 (20459): Remato de escribir  B 5 veces: hora=Thu Dec  3 20:47:22 2009
PAI ESCRITOR: proceso fillo 20459 acaba con status 256.
```

lector.c: Este programa crea ***nl*** procesos fillos, cada un dos cales simulará o funcionamiento dun lector. Un argumento "**ntimes**" indica cantas veces amosará cada proceso o estado do *búffer compartido*. Unha vez que un proceso lector teña conseguido amosar "ntimes" veces o contido do *búffer compartido*, rematará a súa execución.
De novo, o proceso *pai* non rematará a súa execución ata que todos os seus fillos teñan rematado, e amosará o *status* co que cada proceso fillo remate.

Sintaxe: $lector [-l nl -times ntimes]

exemplo de execución: $./lector -l 2 -times 1

```
creados 2 procesos 'lectores' [1, pid 20991][2,pid 20992]
lector 1 (20991): Comezo a leer: hora=Thu Dec  3 20:59:49 2009
lector 2 (20992): Comezo a leer: hora=Thu Dec  3 20:59:49 2009
        [PID:20992][NL=2][NE=0]::[num=20], Buffer[AAAAABBBBBAAAAABBBBB]
lector 2 (20992): Remato de leer: hora=Thu Dec  3 20:59:51 2009
PAI lector: proceso fillo 20992 acaba con status 256
        [PID:20991][NL=1][NE=0]::[num=20], Buffer[AAAAABBBBBAAAAABBBBB]
lector 1 (20991): Remato de leer: hora=Thu Dec  3 20:59:52 2009
PAI lector: proceso fillo 20991 acaba con  status 256.
```

NOTA: Nesta execución asumiuse que non había lectores concurrentemente, senón que se executaron os lectores cando os escritores xa tiñan rematado a súa execución. Nunha condición normal, lectores e escritores tratarán de acceder concurrentemente ao búffer, e polo tanto o contido do búffer amosado por un lector nun momento dado dependerá do estado no que quedase o búffer tras a modificación producida polo último escritor.

finaliza.c: Unha vez que todos los productores e consumidores teñan rematado as súas execucións, este programa encargarase de liberar todos los recursos utilizados (creados por "inicializa.c"). Isto es, os semáforos creados, e a/s zona/s de memoria compartida.

Sintaxe: $ finaliza /*non ten parámetros adicionais.*/

exemplo de execución: $./finaliza

Conxunto de semaforos con id = 1179648, liberados con éxito [OK]
Segmento de memoria compartida, con id=6717488 liberado con éxito [OK]

Creación de procesos lectores/escritores:

Para a creación dun número concreto de procesos fillos (lectores ou escritores), pode usarse a función *fork()*, como se amosa a continuación:

```
CreaProcesos(int np) {
   pid_t pid;
   int i;
   for (i=0;i<np;i++) {  //np = número de procesos a crear
      pid=fork();
      if (pid == 0) {
         break;
      }
   }
   printf("\n o pai do proceso %d é o %d",getpid(),getppid());
}
```

Cada proceso lector/escritor en lugar executarse dentro dun bucle infinito, executarase un número fixo de veces (en función do parámetro *–times*).

Sempre que un *lector* acabe de amosar o contido do búffer compartido, ou que un *escritor* o modifique, quedarán en espera un tempo aleatorio entre 1 e 4 segundos (ver funcións *sleep, srand, rand, urand)*.

A información que amosarán por pantalla os procesos lector/escritor cando (i) consigan acceder ao búffer compartido, (ii) modifiquen/lean os datos do búffer, e (iii) eesperten tras ter durmido ese tempo aleatorio entre 1-4 segundos, é a amosada nos exemplos de execución previos.

Todas as salidas de información a pantalla serán realizadas mediante a chamada ao sistema *write*.

Exemplo de uso de **write()**:

```
void ler (tBuffer *buffer, uint id_proc, pid_t p){

   char mensaxe [MAXCADENA];

   time_t t=time(NULL);
   sprintf (mensaxe,"\nlector %d (%u): Comenzo a ler: hora=%s"
                                          ,id_proc,p,ctime(&t));
   write (STDOUT_FILENO,mensaxe,strlen(mensaxe));

   amosaEstadoBuffer(buffer);

   /**esperar tempo aleatorio entre 1 e 4 segundos**/

   sprintf (mensaxe,"\nlector %d (%u): Remato de ler: hora=%s"

                                          ,id_proc,p,ctime(&t));
   write (STDOUT_FILENO,mensaxe,strlen(mensaxe));
}
```

NOTA: Esta función *__ler__* pode conter erros ou funcionalidade que non se axuste ás necesidades da práctica.

Memoria compartida:

A información compartida, y el almacenamiento la variable N, se implementará siguiendo el siguiente tipo de datos.

```
struct abuff {
   char datos [MAX_ITEMS];  //vector de longitud indicada en 'inicializa.c'
   int nitems;
   /*poden engadirse aquí máis campos que poidan ser de interese*/
   /*para a xestión do  array datos */
};
typedef struct abuff tBuffer;


struct tcompartido {
   tBuffer buffer;
   int nlectores;
   int nescritores;

};
typedef struct tcompartido tMemCompartida;
```

Recordemos que MAX_ITEMS será una constante tal que MAX_ITEMS >= nitems.

Ao ser necesario que a zona de memoria compartida sexa común a todos os procesos lectores/escritores, non é posible declarala simplemente como:

tMemCompartida comun;

pois isto daría lugar a que cada proceso tivese a súa propia variable de tipo tMemCompartida chamada *comun*, que en definitiva, non sería compartida entre todos eles!!!

Para poder compartir unha estructura común, usaremos las funciones de la librería ipc para manexo de memoria compartida ***shmget, shmat, shmdt, shmctl*** do seguinte modo:

1. O proceso pai crea/accede a unha área de memoria compartida usando *shmget*(). Shmget devolve un número (shmid) que será usado como identificador da zona de memoria compartida que se teña creado. Á función shmget() debe pasárselle unha clave (key), que pode obterse mediante a función ftok().

   Ex:

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

2. Usando o identificador que devolveu shmget, cada proceso poderá obter un punteiro á zona de memoria compartida (a zona común), e a través de dito punteiro acceder á zona compartida cando sexa preciso (saber cantos lectores ou escritores están accedendo ao búffer de datos, ver/modificar o seu contido, etc.)

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

3. A partires deste momento, cada proceso poderá usar *pcomun* como se fose un punteiro a tMemCompartida "dos de sempre", pero que se comporta agora como unha estrutura que comparte memoria.
4. Ao finalizar cada proceso lector/escritor, debe "desasignarse" o punteiro *pcomun* usando *shmdt*.

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

Utilización de semáforos:
A implementación proposta require do uso de 4 semáforos.

A creación de semáforos farase mediante a chamada ao sistema *semget()*. Pódense implementar as operaciones básicas para la manipulación de cada semáforo facendo uso das chamadas *semop() y semctl()*. Finalmente la llamada *semctl()* tamén permitirá liberar dito recurso. Más concretamente:

- semget: Permite obtener un grupo de k semáforos que será identificado por un identificador semid.
- semop: Opera (alomenos) sobre un dos semáforos (i) do grupo obtenido con semget (grupo identificado por *semid*), e permite entre outras cousas, facer as operacións P e V sobre algún dos semáforos del grupo.
- semctl: encargase de liberar un grupo de semáforos identificado por *semid* cando executa o comando IPC_RMID. Tamén permite establecer un valor a un semáforo determinado, cando a través de semctl se executa o comando SETVAL.

NOTAS:
Ao igual que shmget(), semget() recibe como parámetro un valor key_t key, que pode obterse previamente coa función ftok(). Deberemos asegurarnos de que a clave que devolva ftok sexa distinta que a obtenida para crear o búffer compartido, e a mesma para todos os procesos que teñen que "compartir" o mesmo semáforo.

Máis información e comandos de interese.
Ver manual en liña para as funciones comentadas.
Ipcs e *ipcrm* para ver e eliminar recursos compartidos (semáforos ou segmentos de memoria compartida) desde a liña de comandos.

MODO DE ENTREGA.

As prácticas entregaranse por email antonio.fari.so@gmail.com ou jcasanova@udc.es antes de proceder a súa defensa. Deberase enviar un ficheiro comprimido tar.gz contendo: a) código fonte e b) ficheiro Makefile (o profesor compilará tecleando $make). O asunto da mensaxe deberá ser o seguinte:

[SO1P2]::[Nome 1º Integrante Grupo][Nome 2º Integrante Grupo] // de ser só un alumno → 1 só nome.
O nome do ficheiro comprimido debe ter os códigos de usuario concatenados: loginAlumno1-loginAlumno2.tar.gz

A práctica será entregada e defendida ante o profesor na aula de prácticas. Todos os membros dun grupo deberán estar presentes para a entrega, de xeito que o profesor poida revisar o seu correcto funcionamento, así como realizar comentarios/cuestións aos integrantes do grupo. É habitual que, durante a defensa, o profesor lle pida a algún dos integrantes a realización de pequenos cambios no código que se poidan considerar pertinentes (para solucionar algún problema/ engadir algunha pequena variante). Nese caso, o alumno deberá amosar a súa solvencia á hora de abordar o cambio solicitado.

As prácticas entregadas e que posteriormente non se saiban defender correctamente, "non serán ben vistas" e (dependendo das circunstancias) poderán implicar un "**non apto**" para todos os membros do grupo.

O programa **debe**:
- Compilar correctamente: usarase gcc coa opción -Wall, e non debe conter erros.
- Executar correctamente:
  - Seguirá as especificacións marcadas.
  - Funcionará correctamente. O alumno debe ter en conta os valores que devolven todas as funcións que utilice (OLLO aos valores devoltos polas funcións utilizadas!!).
  - Non debe conter memory-leaks (o profesor usará **valgrind** para chequear este aspecto).

- A data límite de entrega será o: Domingo **17 de xaneiro de 2009 (ata as 23:59:59h)** a partires dese intre calquera práctica recibida será considerada fóra de prazo (o que implica que a súa valoración máxima será dividida por 2). Prácticas enviadas despois do 29 de xaneiro serán consideradas "non-aptas".

DEFENSA.

A defensa realizarase durante a semana do 18 ao 22 de xañeiro. Cada alumno ten que defendela dentro do grupo de prácticas ao que pertenza (Luns, Martes, Mércoles ou Venres).

DETALLES A RECORDAR.

- O valor máximo das prácticas é de 1.5 puntos. Cómpre que todas as prácticas reciban a calificación de "apta", para poder aprobar a asignatura.
- Non todas as prácticas teñen o mesmo valor. En principio esta primeira práctica ten un valor de 0.75 Sen embargo, os profesores resérvanse a posibilidade de modificar esta valoración (+-0.15).
- Para acadar a máxima nota a práctica debe funcionar correctamente, e a defensa da mesma deberá ser adecuada.
- A detección de prácticas copiadas (salvo raras excepcións) implicará a cualificación da práctica como "non-apta" na convocatoria actual.

Anexo 1: Problema dos lectores/escritores (prioridade escritores)

http://www.dc.fi.udc.es/ai/~soto/sist_oper/Sol_prob_clas_sem.htm

Estructuras compartidas

```
semáforo mutex1, mutex2;          /* inicializados a 1.  */
semáforo lectores, escritores;    /* inicializados a 1  */
integer n_lectores, n_escritores; /* reconto do número de lectores na Sec. Crí. */
```

*** código para cada lector/escritor i; i=1...nl */**

```
lector(i)
begin
   repeat
      P(lectores);
      P(mutex1);
         n_lectores= n_lectores +1;
         if (n_lectores = 1) then
            P(escritores)
      V(mutex1)
      V(lectores);

   ----- Ler -----

      P(mutex1);
         n_lectores= n_lectores -1;
         if (n_lectores = 0) then
            V(escritores);
      V(mutex1)
   until false
end;
```

```
escritor(i)
begin
   repeat
      P(mutex2);
         n_escritores= n_escritores +1;
         if (n_escritores = 1) then
            P(lectores)
      V(mutex2)
      P(escritores);

   ----- Escribir -----

      V(escritores);
      P(mutex2);
         n_escritores= n_escritores -1;
         if (n_escritores = 0) then
            V(lectores);
      V(mutex2)
   until false
end;
```