

# Práctica 6: Semáforos

75.59 - Técnicas de Programación Concurrente I

## Ejercicios

1. Estudiar el comando `ipcs` y, dentro de la cabecera `/usr/include/sys/ipc.h`, la siguiente estructura de datos:

```
1 struct ipc_perm {
2     ushort uid;
3     ushort gid;
4     ushort cuid;
5     ushort mode;
6     ushort seq;
7     key_t key;
8 };
```

2. Mediante la llamada al sistema `semget()` se puede crear o acceder a un conjunto de semáforos unidos bajo un mismo identificador. Escribir un código que indique cómo crear un nuevo identificador con cuatro semáforos, asociado a una clave creada a partir del archivo "auxiliar" y la clave 'k'. Debe ser creado con permisos de lectura y modificación para el usuario.
3. Escribir un código que cree un identificador con cuatro semáforos asociados tales que los dos primeros se inicializan en cinco y el resto en ocho. Luego se pregunta por el valor del semáforo número tres.

4. Explicar el siguiente código:

```
1 struct sembuf operaciones[4];
2 ...
3 operaciones[0].sem_num = 1;
4 operaciones[0].sem_op = -1;
5 operaciones[0].sem_flg = 0;
6 operaciones[1].sem_num = 3;
7 operaciones[1].sem_op = 1;
8 operaciones[1].sem_flg = 0;
9 semop ( semid,operaciones,2 );
10 ...
```

5. Escribir un programa que muestre la sincronización entre procesos padre e hijo, utilizando semáforos: se necesitan al menos dos semáforos. Uno de los semáforos es utilizado por el padre para darle paso a su hijo. El otro es para que lo utilice el hijo en forma análoga.  
Proceso padre: toma su semáforo, realiza operaciones. Cuando termina, abre el semáforo del hijo. El padre espera hasta que el hijo haga un signal sobre su semáforo.  
Proceso hijo: similar al padre. Primero toma su semáforo y luego levanta el de su padre.

6. Explicar el siguiente código:

```
1 #include <malloc.h>
2 #include <stdio.h>
3 #include <sys/types.h>
4 // #include <sys/ipc.h>
5 // #include <sys/sem.h>
6
7 int initsem ( int semid,int nbsem ) {
8     unsigned short *op;
9     int i, n;
```

```

10     op = (unsigned short *) malloc ( sizeof(unsigned short)*nbsem );
11
12     for ( i = 0; i < nbsem; i++ ) {
13         printf ( "sem[%d] = \007", i );
14         scanf ( "%d", &n );
15         op[i] = n;
16     }
17     return semctl ( semid, nbsem, SETALL, op );
18 }
19
20 int mostrar ( int semid, int nbsem ) {
21     int i, val;
22     struct semid_ds buf;
23     for ( i = 0; i < nbsem; i++ ) {
24         if ( (val = semctl(semid, i, GETVAL)) == -1 ) {
25             fprintf ( stderr, "semaforo %d-> ", i );
26             perror("semctl ");
27         } else
28             printf("semaforo %d = %d\n", i, val);
29     }
30     return 0;
}

```

7. Escribir un código que utilice las funciones del ejercicio anterior, de modo tal que el programa cree un conjunto de semáforos, los inicialice y muestre la manipulación que hace sobre los mismos. El programa debe también, mostrar toda la información sobre los semáforos.
8. Implementar el problema de lectores y escritores en ambiente Unix-Linux utilizando semáforos.
9. Implementar el problema de los filósofos comensales, tal como se presentó en clase.
10. Implementar el problema de consumidores - productores con buffer acotado.

## Ejercicios adicionales

1. En la solución vista en clase al problema de lectores - escritores con prioridad de escritores, ¿cómo se consigue dar prioridad a cualquier escritor que espera escribir frente a los lectores que esperan leer?
2. En la solución vista en clase al problema de lectores - escritores con prioridad de lectura, ¿por qué se pueden perder cualidades de tipo "liveness"?
3. Hacer un informe (corto) acerca de las instrucciones especiales para implementar las operaciones "lock" y "unlock" necesarias para asegurar la atomicidad de *wait()* y *signal()* con que cuentan los procesadores Intel actuales.
4. Escribir una implementación de una solución al problema de los filósofos comensales, en la que un filósofo impar toma primero su palito de la izquierda y luego el de la derecha, mientras que un filósofo par toma primero el palito de su derecha y luego el de su izquierda. Analizar las propiedades de corrección de la implementación.
5. Definir algún invariante en el modelo de solución del problema de productores - consumidores con buffer acotado y demostrar que lo es.
6. En la solución al problema de productores - consumidores demostrar que no puede ser que el consumidor sobrepase al productor. Ayuda: utilizar los invariantes de semáforos.
7. Sean las siguientes proposiciones:
  - a)  $P \leftarrow W$  significa el proceso P invocó la operación *wait()* de semáforos y está esperando que sea completada
  - b)  $s=0$  significa que el valor del semáforo s es cero
8. Explicar la semántica de las siguientes proposiciones de la lógica temporal:
  - a)  $(P \leftarrow W) \wedge (s = 0) \rightarrow (P \leftarrow W)$

$$b) (P \leftarrow W) \rightarrow (NOP \leftarrow W)y(s = 1))$$

9. Ver con un ejemplo cuál es el resultado de operar con *signal()* a un semáforo System V, binario y que está valiendo 1.
10. Mostrar, utilizando el modelo de problema de los filósofos comensales, situaciones en las que falla la corrección de manera que se produzca:
  - a) Un estado de livelock
  - b) Un estado de deadlock
  - c) Un estado de espera indefinida

## Apuntes

Introducción  
Creación de semáforos  
Inicialización de semáforos  
Operaciones con semáforos  
Destrucción de semáforos  
Bibliografía

Semáforos

75.59 - Técnicas de Programación Concurrente I

Facultad de Ingeniería - Universidad de Buenos Aires

75.59 - Técnicas de Programación Concurrente I

Semáforos

Introducción  
Creación de semáforos  
Inicialización de semáforos  
Operaciones con semáforos  
Destrucción de semáforos  
Bibliografía

Resumen

- 1 Introducción
- 2 Creación de semáforos
- 3 Inicialización de semáforos
- 4 Operaciones con semáforos
- 5 Destrucción de semáforos
- 6 Bibliografía

75.59 - Técnicas de Programación Concurrente I

Semáforos

**Introducción**

Creación de semáforos

Inicialización de semáforos

Operaciones con semáforos

Dstrucción de semáforos

Bibliografía

## Introducción (I)

Mecanismos de sincronismo de acceso a un recurso

Un semáforo es un contador:

- Si contador  $> 0 \Rightarrow$  recurso disponible
- Si contador  $\leq 0 \Rightarrow$  recurso no disponible
- El valor del semáforo representa la cantidad de recursos disponibles
- Si el valor es cero o uno, se llaman *semáforos binarios* y se comportan igual que los locks de escritura

Operaciones

- p (*wait*): resta 1 al contador
- v (*signal*): suma 1 al contador

75.59 - Técnicas de Programación Concurrente I
Semáforos

**Introducción**

Creación de semáforos

Inicialización de semáforos

Operaciones con semáforos

Dstrucción de semáforos

Bibliografía

## Introducción (II)

Tipos de semáforos

- System V
- POSIX

Un semáforo System V está compuesto por:

- El valor del semáforo
- El *process id* del último proceso que utilizó el semáforo
- La cantidad de procesos esperando por el semáforo
- La cantidad de procesos que está esperando que el semáforo sea cero

75.59 - Técnicas de Programación Concurrente I
Semáforos

## Creación de un conjunto de semáforos

### Función *semget()*

```
int semget ( key_t key,int nsems,int semflg );
```

- Parámetros:

- key: identificador del conjunto generado con *ftok()*
- nsems: cantidad de semáforos del conjunto
- semflg: permisos OR flags
  - IPC\_CREAT: crea el semáforo si no existe
  - IPC\_EXCL: si el semáforo existe, la función falla

- Retorna:

- identificador del conjunto (entero positivo) en caso de éxito
- -1 en caso de error, seteando la variable externa *errno*

## Inicialización de semáforos (I)

### Función *semctl()*

```
int semctl ( int semid,int semnum,int cmd, ... );
```

- Parámetros:

- semid: identificador del conjunto de semáforos a inicializar
- semnum: número del semáforo dentro del conjunto
- cmd: operación SETVAL
- resto: parámetros de la operación; es de tipo *union\_semnum*: se debe definir explícitamente en el código (no existe en ningún *header*)

- Retorna:

- 0 en caso de éxito para la operación SETVAL
- -1 en caso de error, seteando la variable externa *errno*

## Inicialización de semáforos (II)

Ejemplo:

```
// se define explícitamente la union
union semnum {
    int val;
    struct semid_ds* buf;
    ushort* array;
};

semnum init;
init.val = valorInicial;
semctl ( id,0,SETVAL,init );
```

## Inicialización de semáforos (III)

La creación y la inicialización se realizan en dos pasos  $\Rightarrow$  *race condition*

- Puede ocurrir que un proceso trate de utilizar un semáforo que aún no ha sido inicializado
- Varias soluciones posibles:
  - Utilizar un único proceso para crear e inicializar todos los semáforos que se utilizarán
  - Utilizar el campo *sem\_otime* para saber si el semáforo fue inicializado (consultar en la bibliografía)
  - Utilizar un lockfile
  - ...

## Otras operaciones que se pueden realizar con `semctl`

Comando	Resultado
SETVAL	Establece el valor del semáforo indicado en <code>semctl()</code>
GETVAL	Obtiene el valor del semáforo indicado en <code>semctl()</code>
SETALL	Establece el valor para todos los semáforos del conjunto (semnum se ignora)
GETALL	Obtiene el valor de todos los semáforos del conjunto y los almacena en el array de la unión (semnum se ignora)
IPC_RMID	Elimina el conjunto de semáforos (semnum se ignora)
IPC_STAT	Obtiene información del estado del conjunto

## Operaciones con semáforos (I)

### Función `semop()`

```
int semop ( int semid, struct sembuf* sops, unsigned nsops );
```

- Realiza una o más operaciones sobre los semáforos elegidos dentro del conjunto
- Las operaciones se realizan en forma atómica
- Parámetros:
  - `semid`: identificador del conjunto
  - `sops`: puntero al vector con operaciones
  - `nsops`: cantidad de elementos del vector de operaciones
- Retorna:
  - 0 en caso de éxito
  - -1 en caso de error, seteando la variable externa `errno`



## Operaciones con semáforos (II)

Estructura de operaciones:

```
struct sembuf {
    ushort sem_num; /* ID del semaforo en el set */
    short sem_op; /* positivo, negativo o cero */
    short sem_flg; /* IPC_NOWAIT, SEM_UNDO */
};
```

Miembros:

- sem\_op:
  - Positivo: incrementa el contador en sem\_op
  - Negativo: decrementa el contador en sem\_op
  - Cero: el proceso espera a que el valor sea cero
- sem\_flg:
  - IPC\_NOWAIT: no se bloquea si el semáforo es cero
  - SEM\_UNDO: se deshacen las operaciones cuando el programa termina

## Operaciones con semáforos (III)

- Operación p:

```
struct sembuf operacion;
operacion.sem_num = 0; // numero de semaforo
operacion.sem_op = -1; // restar 1 al semaforo
operacion.sem_flg = SEM_UNDO;

semop ( id,&operacion,1 );
```

- Operación v:

```
struct sembuf operacion;
operacion.sem_num = 0; // numero de semaforo
operacion.sem_op = 1; // sumar 1 al semaforo
operacion.sem_flg = SEM_UNDO;

semop ( id,&operacion,1 );
```

Introducción  
Creación de semáforos  
Inicialización de semáforos  
Operaciones con semáforos  
**Destrucción de semáforos**  
Bibliografía

## Destrucción de semáforos

Destrucción del conjunto

- Función `semctl()`
- Comando `IPC_RMID` sin argumentos:  

```
semctl ( id,0,IPC_RMID );
```

Comandos útiles

- `ipcs`: lista los conjuntos de semáforos que están creados
- `ipcrm`: permite eliminar un conjunto de semáforos

75.59 - Técnicas de Programación Concurrente I

Semáforos

Introducción  
Creación de semáforos  
Inicialización de semáforos  
Operaciones con semáforos  
Destrucción de semáforos  
**Bibliografía**

## Bibliografía

- *The Design of the Unix Operating System*, Maurice Bach
- *Unix Network Programming, Interprocess Communications*, W. Richard Stevens, segunda edición
- Manuales del sistema operativo

75.59 - Técnicas de Programación Concurrente I

Semáforos

## Fuentes de los ejemplos

Listado 1: Ejemplo 1

```

1  #ifndef EJEMPLO_1
2
3  #include <iostream>
4  #include <unistd.h>
5  #include <stdlib.h>
6  #include <sys/wait.h>
7
8  #include "Semaforo.h"
9  #include "MemoriaCompartida.h"
10
11 using namespace std;
12
13 int calcularRandom ();
14
15 int main () {
16
17     static const string NOMBRE = "main1.cc";
18     static const string SHM = "Semaforo.h";
19     static const char LETRA = 'a';
20
21     MemoriaCompartida<int> buffer;
22     Semaforo semaforo ( NOMBRE,0 );
23
24     int pid = fork ();
25
26     if ( pid == 0 ) {
27
28         // lector
29         buffer.crear ( SHM,LETRA );
30         cout << "Lector: esperando por el semaforo . . ." << endl;
31         semaforo.p ();
32         int resultado = buffer.leer ();
33         buffer.liberar ();
34         cout << "Lector: tome el semaforo - valor leído = " << resultado << endl;
35         exit ( 0 );
36
37     } else {
38
39         // escritor
40         buffer.crear ( SHM,LETRA );
41         int aDormir = calcularRandom ();
42         cout << "Escritor: durmiendo " << aDormir << " segundos hasta liberar el
43             semaforo" << endl;
44         sleep ( aDormir );
45         buffer.escribir ( aDormir );
46         semaforo.v ();
47         cout << "Escritor: libere el semaforo" << endl;
48
49         // espero a que finalice el hijo antes de liberar los recursos
50         wait ( NULL );
51
52         buffer.liberar ();
53         semaforo.eliminar ();
54         exit ( 0 );
55     }
56
57     int calcularRandom () {
58         srand ( time(NULL) );
59         int resultado = rand() % 10 + 1;
60         return resultado;
61     }
62
63 #endif

```

Listado 2: Ejemplo 2

```

1  #ifndef EJEMPLO_2
2
3  #include <iostream>
4  #include <unistd.h>
5  #include <stdlib.h>
6  #include <sys/wait.h>
7
8  #include "Semaforo.h"
9  #include "MemoriaCompartida.h"

```

```

10
11
12 using namespace std;
13
14 int calcularRandom ();
15
16 int main () {
17
18     static const string NOMBRE1 = "main1.cc";
19     static const string NOMBRE2 = "main2.cc";
20     static const string NOMBRE3 = "main3.cc";
21     static const string SHM = "Semaforo.cpp";
22     static const char LETRA = 'a';
23
24     MemoriaCompartida<int> buffer;
25     Semaforo semaforo1 ( NOMBRE1,0 );
26     Semaforo semaforo2 ( NOMBRE2,0 );
27     Semaforo semaforo3 ( NOMBRE3,0 );
28
29     pid_t pid = fork ();
30
31     if ( pid == 0 ) {
32
33         int pid2 = fork ();
34
35         if ( pid2 == 0 ) {
36
37             int pid3 = fork ();
38
39             if ( pid3 == 0 ) {
40
41                 // lector 1
42                 buffer.crear ( SHM,LETRA );
43
44                 cout << "Lector (pid " << getpid () << "): esperando por el
45                     semaforo" << endl;
46                 semaforo1.p ();
47
48                 int resultado = buffer.leer ();
49                 cout << "Lector (pid " << getpid() << "): tome el semaforo -
50                     valor leído = " << resultado << endl;
51
52                 buffer.liberar ();
53                 exit ( 0 );
54             } else {
55
56                 // lector 2
57                 buffer.crear ( SHM,LETRA );
58
59                 cout << "Lector (pid " << getpid () << "): esperando por el
60                     semaforo" << endl;
61                 semaforo2.p ();
62
63                 int resultado = buffer.leer ();
64                 cout << "Lector (pid " << getpid() << "): tome el semaforo -
65                     valor leído = " << resultado << endl;
66
67                 buffer.liberar ();
68                 wait(NULL);
69                 exit ( 0 );
70             }
71         } else {
72
73             // lector 3
74             buffer.crear ( SHM,LETRA );
75
76             cout << "Lector (pid " << getpid () << "): esperando por el semaforo"
77                 << endl;
78             semaforo3.p ();
79
80             int resultado = buffer.leer ();
81             cout << "Lector (pid " << getpid() << "): tome el semaforo - valor
82                 leído = " << resultado << endl;
83
84             buffer.liberar ();
85             wait(NULL);
86             exit ( 0 );
87         }
88     } else {
89
90     }
91 }

```

```

86         // escritor
87         buffer.crear ( SHM,LETRA );
88
89         int aDormir = calcularRandom ();
90
91         cout << "Escritor (pid " << getpid () << "): duerme " << aDormir << " segundos
           hasta liberar el semaforo" << endl;
92         sleep ( aDormir );
93
94         buffer.escribir ( aDormir );
95         semaforo1.v ();
96         semaforo2.v ();
97         semaforo3.v ();
98         cout << "Escritor: libere los semaforos" << endl;
99
100        // espero a que terminen los hijos antes de liberar los recursos
101        wait ( NULL );
102
103        buffer.liberar ();
104        semaforo1.eliminar ();
105        semaforo2.eliminar ();
106        semaforo3.eliminar ();
107        exit ( 0 );
108    }
109 }
110
111 int calcularRandom () {
112     srand ( time(NULL) );
113     int resultado = rand() % 10 + 1;
114     return resultado;
115 }
116
117 #endif

```

### Listado 3: Ejemplo 3

```

1  #ifndef EJEMPLO_3
2
3  #include <iostream>
4  #include <unistd.h>
5  #include <stdlib.h>
6  #include <sys/wait.h>
7
8  #include "Semaforo.h"
9  #include "MemoriaCompartida.h"
10
11
12 using namespace std;
13
14 int calcularRandom ();
15
16 int main () {
17
18     static const string NOMBRE = "main3.cc";
19     static const string SHM = "Semaforo.cpp";
20     static const char LETRA = 'a';
21     static const int HIJOS = 3;
22
23     MemoriaCompartida<int> buffer;
24     Semaforo semaforo ( NOMBRE,0 );
25
26     for ( int i=0;i<HIJOS;i++ ) {
27         pid_t pid = fork();
28
29         if ( pid == 0 ) {
30             // lector n
31             buffer.crear ( SHM,LETRA );
32
33             cout << "Lector (pid " << getpid () << "): esperando por el semaforo"
34                 << endl;
35             semaforo.p ();
36
37             int resultado = buffer.leer ();
38             buffer.liberar ();
39             cout << "Lector (pid " << getpid() << "): tome el semaforo - valor
40                 leído = " << resultado << endl;
41             exit ( 0 );
42         }
43     }
44
45     // escritor

```

```

44     buffer.crear ( SHM,LETRA );
45     int aDormir = calcularRandom ();
46
47     cout << "Escritor (pid " << getpid () << "): duerme " << aDormir << " segundos hasta
        liberar el semaforo" << endl;
48     sleep ( aDormir );
49
50     buffer.escribir ( aDormir );
51     for ( int i=0;i<HIJOS;i++ ) {
52         semaforo.v();
53     }
54     cout << "Escritor: libere el semaforo" << endl;
55
56     // espero a que terminen los hijos antes de liberar los recursos
57     for ( int i=0;i<HIJOS;i++ ) {
58         wait(NULL);
59     }
60
61     buffer.liberar ();
62     cout << "Escritor: Buffer liberado" << endl;
63     semaforo.eliminar ();
64
65     exit ( 0 );
66 }
67
68 int calcularRandom () {
69     srand ( time(NULL) );
70     int resultado = rand() % 10 + 1;
71     return resultado;
72 }
73
74 #endif

```

#### Listado 4: Clase Semaforo

```

1  #ifndef SEMAFORO_H_
2  #define SEMAFORO_H_
3
4  #include <sys/ipc.h>
5  #include <sys/sem.h>
6  #include <sys/types.h>
7  #include <string>
8
9  class Semaforo {
10
11 private:
12     int id;
13     int valorInicial;
14
15     int inicializar () const;
16
17 public:
18     Semaforo ( const std::string& nombre,const int valorInicial );
19     ~Semaforo();
20
21     int p () const; // decrementa
22     int v () const; // incrementa
23     void eliminar () const;
24 };
25
26 #endif /* SEMAFORO_H_ */

```

#### Listado 5: Clase Semaforo

```

1  #include "Semaforo.h"
2
3  Semaforo :: Semaforo ( const std::string& nombre,const int valorInicial ):valorInicial(
    valorInicial) {
4      key_t clave = ftok ( nombre.c_str(),'a' );
5      this->id = semget ( clave,1,0666 | IPC_CREAT );
6
7      this->inicializar ();
8  }
9
10 Semaforo::~Semaforo() {
11 }
12
13 int Semaforo :: inicializar () const {
14
15     union semnum {

```

```

16         int val;
17         struct semid_ds* buf;
18         ushort* array;
19     };
20
21     semnum init;
22     init.val = this->valorInicial;
23     int resultado = semctl ( this->id,0,SETVAL,init );
24     return resultado;
25 }
26
27 int Semaforo :: p () const {
28
29     struct sembuf operacion;
30
31     operacion.sem_num = 0; // numero de semaforo
32     operacion.sem_op = -1; // restar 1 al semaforo
33     operacion.sem_flg = SEM_UNDO;
34
35     int resultado = semop ( this->id,&operacion,1 );
36     return resultado;
37 }
38
39 int Semaforo :: v () const {
40
41     struct sembuf operacion;
42
43     operacion.sem_num = 0; // numero de semaforo
44     operacion.sem_op = 1; // sumar 1 al semaforo
45     operacion.sem_flg = SEM_UNDO;
46
47     int resultado = semop ( this->id,&operacion,1 );
48     return resultado;
49 }
50
51 void Semaforo :: eliminar () const {
52     semctl ( this->id,0,IPC_RMID );
53 }

```