

Práctica 2: Memoria Compartida

75.59 - Técnicas de Programación Concurrente I

Ejercicios

1. Investigar los límites existentes en el sistema operativo para los tamaños máximo y mínimo de un segmento de memoria compartida.
2. Escribir un programa en C o C++ que permita detectar los límites anteriores por prueba y error.
3. Las principales llamadas al sistema para gestión de memoria compartida son: obtener memoria compartida `shmget()`; control de memoria compartida `shmctl()`; adosar memoria compartida `shmat()`; y liberar memoria compartida `shmdt()`. Sus prototipos se encuentran definidos en la cabecera `<sys/types.h>`.
Escribir un programa con dos procesos que se comunican mediante una memoria compartida. Uno escribe una palabra en la memoria compartida y el otro la lee e imprime.

Apuntes

Introducción
Creación de una memoria compartida
Adosado (attach) de la memoria compartida
Lectura y escritura de los datos
Destrucción de la memoria compartida
Bibliografía

Memoria Compartida

75.59 - Técnicas de Programación Concurrente I

Facultad de Ingeniería - Universidad de Buenos Aires

75.59 - Técnicas de Programación Concurrente I

Memoria Compartida

Introducción
Creación de una memoria compartida
Adosado (attach) de la memoria compartida
Lectura y escritura de los datos
Destrucción de la memoria compartida
Bibliografía

Resumen

- 1 Introducción
- 2 Creación de una memoria compartida
- 3 Adosado (attach) de la memoria compartida
- 4 Lectura y escritura de los datos
- 5 Destrucción de la memoria compartida
- 6 Bibliografía

75.59 - Técnicas de Programación Concurrente I

Memoria Compartida

Introducción
 Creación de una memoria compartida
 Adosado (attach) de la memoria compartida
 Lectura y escritura de los datos
 Destrucción de la memoria compartida
 Bibliografía

Introducción

- Es una facilidad IPC que ofrece el sistema operativo, la cual posibilita intercambiar datos entre dos o más procesos
- Se implementa como una región de memoria accesible para dichos procesos
- Dos tipos:
 - System V
 - POSIX
- En la clase veremos la versión System V

75.59 - Técnicas de Programación Concurrente I
Memoria Compartida

Introducción
 Creación de una memoria compartida
 Adosado (attach) de la memoria compartida
 Lectura y escritura de los datos
 Destrucción de la memoria compartida
 Bibliografía

Creación de una memoria compartida (I)

Función *shmget()*

```
int shmget ( key_t key, size_t size, int shmflg );
```

- Parámetros:
 - key: clave que se crea con la función *ftok()*
 - size: tamaño en bytes del segmento de memoria que se quiere crear
 - shmflg: flags:
 - Permisos de acceso al segmento de memoria, por ejemplo 0x666
 - IPC_CREAT: crea el segmento de memoria si no existe
 - IPC_EXCL: usado junto con IPC_CREAT, la operación falla si el segmento existe
- Retorna:
 - Un identificador válido del segmento en caso de éxito
 - -1 en caso de error, seteando la variable externa *errno*

75.59 - Técnicas de Programación Concurrente I
Memoria Compartida

Creación de una memoria compartida (II)

Creación de una clave para identificar unívocamente al segmento de memoria compartida

- Función *ftok()*

```
key_t ftok ( const char* pathname,int proj_id );
```

- Parámetros:

- *pathname*: path a un archivo que existe y al cual el proceso tiene permiso de lectura
- *proj_id*: número mayor a cero - se utilizan los 8 bits menos significativos (por lo cual se puede usar un *char*)

- Retorna:

- Una clave válida en caso de éxito
- -1 en caso de error, seteando la variable externa *errno*

Adosado (attach) de la memoria compartida

Mapeo del segmento de memoria al espacio de direcciones de proceso

- Función *shmat()*

```
void* shmat ( int shmid,const void* shmaddr,int shmflg );
```

- Parámetros:

- *shmid*: identificador de segmento de memoria (obtenido con *shmget()*)
- *shmaddr*: dirección de memoria a la cual se mapea el segmento; si es NULL, la elige el sistema operativo
- *shmflg*: flags:
 - SHM_RDONLY: el segmento se adosa sólo para lectura
 - 0 (cero)

- Retorna:

- Puntero al segmento de memoria compartida en caso de éxito
- (*void **) -1 en caso de error, seteando la variable externa *errno*

Introducción
 Creación de una memoria compartida
 Adosado (attach) de la memoria compartida
Lectura y escritura de los datos
 Destrucción de la memoria compartida
 Bibliografía

Lectura y escritura de los datos

- Lectura: leer el dato almacenado en el puntero que retorna *shmat()*
- Escritura: escribir el dato en la dirección de memoria donde apunta el puntero que retorna *shmat()*

El kernel no participa (mediante *system calls*) en la lectura y escritura del bloque de memoria

75.59 - Técnicas de Programación Concurrente I
Memoria Compartida

Introducción
 Creación de una memoria compartida
 Adosado (attach) de la memoria compartida
 Lectura y escritura de los datos
Destrucción de la memoria compartida
 Bibliografía

Destrucción de la memoria compartida (I)

Primer paso: desadosado (detach) del segmento de memoria

- Todos los procesos que hicieron *attach* deben hacer *detach*
- Función *shmdt()*:

```
int shmdt ( const void* shmaddr );
```
- Parámetros:
 - *shmaddr*: dirección del segmento de memoria obtenido con *shmat()*
- Retorna:
 - 0 en caso de éxito
 - -1 en caso de error, seteando la variable externa *errno*

75.59 - Técnicas de Programación Concurrente I
Memoria Compartida

Introducción
 Creación de una memoria compartida
 Adosado (attach) de la memoria compartida
 Lectura y escritura de los datos
Destrucción de la memoria compartida
 Bibliografía

Destrucción de la memoria compartida (II)

Segundo paso: destrucción del objeto IPC

- Función *shmctl()*:

```
int shmctl ( int shmid,int cmd,struct shmid_ds* buf );
```
- Parámetros:
 - shmid: identificador de segmento de memoria (obtenido con *shmget()*)
 - cmd: IPC_RMID
 - buf: NULL
- Retorna:
 - 0 en caso de éxito (para esta operación)
 - -1 en caso de error, seteando la variable externa *errno*

75.59 - Técnicas de Programación Concurrente I
Memoria Compartida

Introducción
 Creación de una memoria compartida
 Adosado (attach) de la memoria compartida
 Lectura y escritura de los datos
Destrucción de la memoria compartida
 Bibliografía

Comandos útiles

Los siguientes son comandos útiles para probar y debuggear:

- *ipcs*: lista información de IPC, entre ella está el listado de objetos de memoria compartida que fueron creados
- *ipcrm*: permite eliminar un recurso de IPC que fue creado previamente

75.59 - Técnicas de Programación Concurrente I
Memoria Compartida

Bibliografía

- *The Design of the Unix Operating System*, Maurice Bach
- Manuales del sistema operativo

Fuentes de los ejemplos

Listado 1: Programa principal

```

1  #ifndef EJEMPLO_1
2
3  #include <iostream>
4  #include <stdlib.h>
5  #include <unistd.h>
6  #include <wait.h>
7  #include "MemoriaCompartida.h"
8
9  using namespace std;
10
11 int calcularRandom ();
12
13 int main () {
14
15     pid_t procId = fork ();
16
17     if ( procId == 0 ) {
18
19         string archivo ( "main1.cc" );
20         MemoriaCompartida<int> memoria;
21         int estadoMemoria = memoria.crear ( archivo, 'R' );
22
23         cout << "Hijo: duermo 5 segundos..." << endl;
24         sleep ( 5 );
25
26         if ( estadoMemoria == SHM_OK ) {
27             int resultado = memoria.leer ();
28             cout << "Hijo: leo el numero " << resultado << " de la memoria
29                 compartida" << endl;
30             memoria.liberar ();
31         } else {
32             cout << "Hijo: error en memoria compartida: " << estadoMemoria << endl;
33         }
34         cout << "Hijo: fin del proceso" << endl;
35         exit ( 0 );
36     } else {
37
38         string archivo ( "main1.cc" );
39         MemoriaCompartida<int> memoria;
40         int estadoMemoria = memoria.crear ( archivo, 'R' );
41         if ( estadoMemoria == SHM_OK ) {
42
43             // escribe un dato para el hijo
44             int random = calcularRandom ();
45             cout << "Padre: escribo el numero " << random << " en la memoria
46                 compartida" << endl;
47             memoria.escribir ( random );
48
49             // espera a que termine el hijo
50             wait ( NULL );
51             // libera la memoria
52             memoria.liberar ();
53         } else {
54             cout << "Padre: error en memoria compartida: " << estadoMemoria << endl;
55         }
56
57         cout << "Padre: fin del proceso" << endl;
58         exit ( 0 );
59     }
60 }
61
62 int calcularRandom () {
63     srand ( time(NULL) );
64     int resultado = rand() % 100 + 1;
65     return resultado;
66 }
67
68 #endif
69

```

Listado 2: Clase MemoriaCompartida

```

1  #ifndef MEMORIACOMPARTIDA_H_

```



```

2  #define MEMORIACOMPARTIDA_H_
3
4  #define SHM_OK                0
5  #define ERROR_FTOK            -1
6  #define ERROR_SHMGET          -2
7  #define ERROR_SHMAT           -3
8
9  #include <sys/types.h>
10 #include <sys/ipc.h>
11 #include <sys/shm.h>
12 #include <string>
13
14
15 template <class T> class MemoriaCompartida {
16
17 private:
18     int shmId;
19     T*   ptrDatos;
20
21     int cantidadProcesosAdosados () const;
22
23 public:
24     MemoriaCompartida ();
25     ~MemoriaCompartida ();
26     int crear ( const std::string& archivo, const char letra );
27     void liberar ();
28     void escribir ( const T& dato );
29     T leer () const;
30
31 };
32
33 template <class T> MemoriaCompartida<T> :: MemoriaCompartida() : shmId(0), ptrDatos(NULL) {
34 }
35
36 template <class T> MemoriaCompartida<T> :: ~MemoriaCompartida() {
37 }
38
39 template <class T> int MemoriaCompartida<T> :: crear ( const std::string& archivo, const char
    letra ) {
40
41     // generacion de la clave
42     key_t clave = ftok ( archivo.c_str(), letra );
43     if ( clave == -1 )
44         return ERROR_FTOK;
45     else {
46         // creacion de la memoria compartida
47         this->shmId = shmget ( clave, sizeof(T), 0644|IPC_CREAT );
48
49         if ( this->shmId == -1 )
50             return ERROR_SHMGET;
51         else {
52             // attach del bloque de memoria al espacio de direcciones del proceso
53             void* ptrTemporal = shmat ( this->shmId, NULL, 0 );
54
55             if ( ptrTemporal == (void *) -1 ) {
56                 return ERROR_SHMAT;
57             } else {
58                 this->ptrDatos = static_cast<T*> (ptrTemporal);
59                 return SHM_OK;
60             }
61         }
62     }
63 }
64
65
66 template <class T> void MemoriaCompartida<T> :: liberar () {
67     // detach del bloque de memoria
68     shmdt ( static_cast<void*> (this->ptrDatos) );
69
70     int procAdosados = this->cantidadProcesosAdosados ();
71
72     if ( procAdosados == 0 ) {
73         shmctl ( this->shmId, IPC_RMID, NULL );
74     }
75 }
76
77 template <class T> void MemoriaCompartida<T> :: escribir ( const T& dato ) {
78     * (this->ptrDatos) = dato;
79 }
80
81 template <class T> T MemoriaCompartida<T> :: leer () const {
82     return ( *(this->ptrDatos) );

```

```

83 }
84
85 template <class T> int MemoriaCompartida<T> :: cantidadProcesosAdosados () const {
86     shmids estado;
87     shmctl ( this->shmId,IPC_STAT,&estado );
88     return estado.shm_nattch;
89 }
90
91
92 #endif /* MEMORIACOMPARTIDA_H_ */

```

Listado 3: Programa principal

```

1  #ifndef EJEMPLO_2
2
3  #include <unistd.h>
4  #include <stdlib.h>
5  #include <wait.h>
6  #include "MemoriaCompartida2.h"
7
8  int calcularRandom ();
9
10 int main () {
11
12     std::string archivo = "main2.cc";
13     pid_t procId = fork ();
14
15     if ( procId == 0 ) {
16         // codigo del hijo
17         try {
18             MemoriaCompartida2<int> buffer ( archivo,'A' );
19
20             std::cout << "Hijo: duermo 5 segundos..." << std::endl;
21             sleep ( 5 );
22
23             int resultado = buffer.leer ();
24             std::cout << "Hijo: leo el numero " << resultado << " de la memoria
                compartida" << std::endl;
25             std::cout << "Hijo: fin del proceso" << std::endl;
26         } catch ( std::string& mensaje ) {
27             std::cerr << mensaje << std::endl;
28         }
29
30         exit(0);
31     } else {
32         // codigo del padre
33         try {
34             MemoriaCompartida2<int> buffer ( archivo,'A' );
35
36             int random = calcularRandom ();
37             std::cout << "Padre: escribo el numero " << random << " en la memoria
                compartida" << std::endl;
38             buffer.escribir(random);
39
40             // espera a que termine el hijo
41             std::cout << "Padre: esperando a que termine el hijo" << std::endl;
42             wait(NULL);
43
44         } catch ( std::string& mensaje ) {
45             std::cerr << mensaje << std::endl;
46         }
47
48         exit(0);
49     }
50
51 }
52
53 int calcularRandom () {
54     srand ( time(NULL) );
55     int resultado = rand() % 100 + 1;
56     return resultado;
57 }
58
59 #endif

```

Listado 4: Clase MemoriaCompartida2

```

1  #ifndef MEMORIACOMPARTIDA2_H_
2  #define MEMORIACOMPARTIDA2_H_
3

```

```

4  #include <sys/types.h>
5  #include <sys/ipc.h>
6  #include <sys/shm.h>
7  #include <string>
8  #include <string.h>
9  #include <iostream>
10 #include <errno.h>
11
12 template <class T> class MemoriaCompartida2 {
13
14 private:
15     int      shmId;
16     T*       ptrDatos;
17
18     int      cantidadProcesosAdosados() const;
19
20 public:
21     MemoriaCompartida2 ();
22     void crear ( const std::string& archivo, const char letra );
23     void liberar ();
24
25     MemoriaCompartida2 ( const std::string& archivo, const char letra );
26     MemoriaCompartida2 ( const MemoriaCompartida2& origen );
27     ~MemoriaCompartida2 ();
28     MemoriaCompartida2<T>& operator= ( const MemoriaCompartida2& origen );
29     void escribir ( const T& dato );
30     T leer () const;
31 };
32
33 template <class T> MemoriaCompartida2<T>::MemoriaCompartida2 () : shmId(0), ptrDatos(NULL) {
34 }
35
36 template <class T> void MemoriaCompartida2<T>::crear ( const std::string& archivo, const char
37     letra ) {
38     key_t clave = ftok ( archivo.c_str(), letra );
39
40     if ( clave > 0 ) {
41         this->shmId = shmget ( clave, sizeof(T), 0644|IPC_CREAT );
42
43         if ( this->shmId > 0 ) {
44             void* tmpPtr = shmat ( this->shmId, NULL, 0 );
45             if ( tmpPtr != (void*) -1 ) {
46                 this->ptrDatos = static_cast<T*> (tmpPtr);
47             } else {
48                 std::string mensaje = std::string("Error en shmat(): ") + std::
49                     string(strerror(errno));
50                 throw mensaje;
51             }
52         } else {
53             std::string mensaje = std::string("Error en shmget(): ") + std::string(
54                 strerror(errno));
55             throw mensaje;
56         }
57     } else {
58         std::string mensaje = std::string("Error en ftok(): ") + std::string(strerror(
59             errno));
60         throw mensaje;
61     }
62 }
63
64 template <class T> void MemoriaCompartida2<T>::liberar() {
65     int errorDt = shmdt ( (void *) this->ptrDatos );
66
67     if ( errorDt != -1 ) {
68         int procAdosados = this->cantidadProcesosAdosados ();
69         if ( procAdosados == 0 ) {
70             shmctl ( this->shmId, IPC_RMID, NULL );
71         }
72     } else {
73         std::string mensaje = std::string("Error en shmdt(): ") + std::string(strerror(
74             errno));
75         throw mensaje;
76     }
77 }
78
79 template <class T> MemoriaCompartida2<T>::MemoriaCompartida2 ( const std::string& archivo, const
80     char letra ) : shmId(0), ptrDatos(NULL) {
81     key_t clave = ftok ( archivo.c_str(), letra );
82
83     if ( clave > 0 ) {
84         this->shmId = shmget ( clave, sizeof(T), 0644|IPC_CREAT );

```

```

80         if ( this->shmId > 0 ) {
81             void* tmpPtr = shmat ( this->shmId, NULL, 0 );
82             if ( tmpPtr != (void*) -1 ) {
83                 this->ptrDatos = static_cast<T*> (tmpPtr);
84             } else {
85                 std::string mensaje = std::string("Error en shmat(): ") + std::string(
86                     strerror(errno));
87                 throw mensaje;
88             }
89         } else {
90             std::string mensaje = std::string("Error en shmget(): ") + std::string(
91                 strerror(errno));
92             throw mensaje;
93         }
94     } else {
95         std::string mensaje = std::string("Error en ftok(): ") + std::string(strerror(
96             errno));
97         throw mensaje;
98     }
99 }
100
101 template <class T> MemoriaCompartida2<T>::MemoriaCompartida2 ( const MemoriaCompartida2& origen
102     ):shmId(origen.shmId) {
103     void* tmpPtr = shmat ( origen.shmId, NULL, 0 );
104
105     if ( tmpPtr != (void*) -1 ) {
106         this->ptrDatos = static_cast<T*> (tmpPtr);
107     } else {
108         std::string mensaje = std::string("Error en shmat(): ") + std::string(strerror(
109             errno));
110         throw mensaje;
111     }
112 }
113
114 template <class T> MemoriaCompartida2<T>::~MemoriaCompartida2 () {
115     int errorDt = shmdt ( static_cast<void*> (this->ptrDatos) );
116
117     if ( errorDt != -1 ) {
118         int procAdosados = this->cantidadProcesosAdosados ();
119         if ( procAdosados == 0 ) {
120             shmctl ( this->shmId, IPC_RMID, NULL );
121         }
122     } else {
123         std::cerr << "Error en shmdt(): " << strerror(errno) << std::endl;
124     }
125 }
126
127 template <class T> MemoriaCompartida2<T>& MemoriaCompartida2<T>::operator= ( const
128     MemoriaCompartida2& origen ) {
129     this->shmId = origen.shmId;
130     void* tmpPtr = shmat ( this->shmId, NULL, 0 );
131
132     if ( tmpPtr != (void*) -1 ) {
133         this->ptrDatos = static_cast<T*> (tmpPtr);
134     } else {
135         std::string mensaje = std::string("Error en shmat(): ") + std::string(strerror(
136             errno));
137         throw mensaje;
138     }
139
140     return *this;
141 }
142
143 template <class T> void MemoriaCompartida2<T>::escribir ( const T& dato ) {
144     *(this->ptrDatos) = dato;
145 }
146
147 template <class T> T MemoriaCompartida2<T>::leer() const {
148     return *(this->ptrDatos);
149 }
150
151 template <class T> int MemoriaCompartida2<T> :: cantidadProcesosAdosados () const {
152     shmids estado;
153     shmctl ( this->shmId, IPC_STAT, &estado );
154     return estado.shm_nattch;
155 }
156
157 #endif

```