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

Memoria Compartida

75.59 - Técnicas de Programación Concurrente I

Facultad de Ingeniería - Universidad de Buenos Aires

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

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

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*

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*

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

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*

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*

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

Bibliografía

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

Fuentes de los ejemplos

Listado 1: Programa principal

```
#ifdef EJEMPLO_1

#include <iostream>
#include <stdlib.h>
#include <unistd.h>
#include <wait.h>
#include "MemoriaCompartida.h"

using namespace std;

int calcularRandom ();

int main () {

        pid_t procId = fork ();

        if ( procId == 0 ) {

                string archivo ( "main1.cc" );
                MemoriaCompartida<int> memoria;
                int estadoMemoria = memoria.crear ( archivo,'R' );

                cout << "Hijo: duermo 5 segundos..." << endl;
                sleep ( 5 );

                if ( estadoMemoria == SHM_OK ) {
                        int resultado = memoria.leer ();
                        cout << "Hijo: leo el numero " << resultado << " de la memoria
                             compartida" << endl;
                        memoria.liberar ();
                } else {
                        cout << "Hijo: error en memoria compartida: " << estadoMemoria << endl;
                }
                cout << "Hijo: fin del proceso" << endl;
                exit ( 0 );

        } else {

                string archivo ( "main1.cc" );
                MemoriaCompartida<int> memoria;
                int estadoMemoria = memoria.crear ( archivo,'R' );
                if ( estadoMemoria == SHM_OK ) {

                        // escribe un dato para el hijo
                        int random = calcularRandom ();
                        cout << "Padre: escribo el numero " << random << " en la memoria
                             compartida" << endl;
                        memoria.escribir ( random );

                        // espera a que termine el hijo
                        wait ( NULL );
                        // libera la memoria
                        memoria.liberar ();
                } else {
                        cout << "Padre: error en memoria compartida: " << estadoMemoria << endl
                             ;
                }

                cout << "Padre: fin del proceso" << endl;
                exit ( 0 );

        }

}

int calcularRandom () {
        srand ( time(NULL) );
        int resultado = rand() % 100 + 1;
        return resultado;
}

#endif
```

Listado 2: Clase MemoriaCompartida

```
#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          shmid_ds 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   #ifdef 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
       letra ) {
37         key_t clave = ftok ( archivo.c_str(),letra );
38
39         if ( clave > 0 ) {
40                 this->shmId = shmget ( clave,sizeof(T),0644|IPC_CREAT );
41
42                 if ( this->shmId > 0 ) {
43                         void* tmpPtr = shmat ( this->shmId,NULL,0 );
44                         if ( tmpPtr != (void*) -1 ) {
45                                 this->ptrDatos = static_cast<T*> (tmpPtr);
46                         } else {
47                                 std::string mensaje = std::string("Error en shmat(): ") + std::
                                        string(strerror(errno));
48                                 throw mensaje;
49                         }
50                 } else {
51                         std::string mensaje = std::string("Error en shmget(): ") + std::string(
                                strerror(errno));
52                         throw mensaje;
53                 }
54         } else {
55                 std::string mensaje = std::string("Error en ftok(): ") + std::string(strerror(
                        errno));
56                 throw mensaje;
57         }
58 }
59
60 template <class T> void MemoriaCompartida2<T>::liberar() {
61         int errorDt = shmdt ( (void *) this->ptrDatos );
62
63         if ( errorDt != -1 ) {
64                 int procAdosados = this->cantidadProcesosAdosados ();
65                 if ( procAdosados == 0 ) {
66                         shmctl ( this->shmId,IPC_RMID,NULL );
67                 }
68         } else {
69                 std::string mensaje = std::string("Error en shmdt(): ") + std::string(strerror(
                        errno));
70                 throw mensaje;
71         }
72 }
73
74 template <class T> MemoriaCompartida2<T>::MemoriaCompartida2 ( const std::string& archivo,const
       char letra ):shmId(0),ptrDatos(NULL) {
75         key_t clave = ftok ( archivo.c_str(),letra );
76
77         if ( clave > 0 ) {
78                 this->shmId = shmget ( clave,sizeof(T),0644|IPC_CREAT );
79
```

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