

Práctica 7: Mensajes

75.59 - Técnicas de Programación Concurrente I

Ejercicios

Para los ejercicios 1 a 4 serán necesarias las siguientes primitivas:

```
int msget ( key_t clave,int opcion );  
int msgsnd ( int msqid,struct msgbuf* pmes,int lg,int opcion );  
int msgrcv ( int msqid,struct msgbuf* pmes,int lgmax,int opcion,long tipo );  
int msgctl ( int msqid,int cmd,struct msqid_ds* pbuf );
```

1. Estudiar el archivo `/usr/include/sys/msg.h`. Describir el contenido del mismo.
2. Escribir el código que muestra cómo crear una cola de mensajes.
3. Escribir el código necesario para enviar y recibir un mensaje de tipo 1, compuesto por una cadena de 20 caracteres.
4. Implementar las operaciones `send()` y `receive()` que posibilitan la comunicación entre procesos, mediante una cola de hasta 20 mensajes de capacidad, utilizando las llamadas al sistema antes indicadas.
5. Hacer lo mismo que en el ejercicio anterior utilizando FIFOs o tuberías con nombre ("named pipes").
6. Escribir dos programas tales que uno se comportará como un gestor de una base de datos compuesta por una tabla de personas (`char(61) nombre, char(120) direccion, char(13) telefono`);
El otro programa es un cliente de la base de datos. Este cliente puede leer el contenido de la base de datos y también puede agregar nuevos registros. Se deberá utilizar cola de mensajes para comunicar el programa cliente con el gestor de la base.
7. ¿Qué comparación se puede establecer entre las señales y los mensajes como facilidades ipc en Unix/Linux?

Apuntes

Introducción
Creación de una cola de mensajes
Transmisión de mensajes
Recepción de mensajes
Destrucción de una cola de mensajes
Bibliografía

Colas de Mensajes

75.59 - Técnicas de Programación Concurrente I

Facultad de Ingeniería - Universidad de Buenos Aires

75.59 - Técnicas de Programación Concurrente I

Colas de Mensajes

Introducción
Creación de una cola de mensajes
Transmisión de mensajes
Recepción de mensajes
Destrucción de una cola de mensajes
Bibliografía

Resumen

1

 Introducción

2

 Creación de una cola de mensajes

3

 Transmisión de mensajes

4

 Recepción de mensajes

5

 Destrucción de una cola de mensajes

6

 Bibliografía

75.59 - Técnicas de Programación Concurrente I

Colas de Mensajes

Introducción
 Creación de una cola de mensajes
 Transmisión de mensajes
 Recepción de mensajes
 Destrucción de una cola de mensajes
 Bibliografía

Introducción

- Mecanismo de sincronismo y pasaje de mensajes entre procesos
- Funcionan de forma similar a los fifos, pero tienen opciones adicionales
- Se pueden usar para comunicación bidireccional entre dos procesos
- Los procesos no deben tener relación entre ellos para compartir una cola de mensajes

75.59 - Técnicas de Programación Concurrente I
Colas de Mensajes

Introducción
 Creación de una cola de mensajes
 Transmisión de mensajes
 Recepción de mensajes
 Destrucción de una cola de mensajes
 Bibliografía

Creación de una cola de mensajes

Función *msgget()*

```
int msgget ( key_t key, int msgflg );
```

- Parámetros:
 - key: clave que identifica la cola generada con *ftok()*
 - msgflg: flags + permisos
 - IPC_CREAT: crea la cola si no existe
 - IPC_EXCL: la función falla si la cola existe
- Retorna:
 - identificador de la cola (entero positivo) creada en caso de éxito
 - -1 en caso de error, seteando la variable externa *errno*

75.59 - Técnicas de Programación Concurrente I
Colas de Mensajes

Introducción
 Creación de una cola de mensajes
Transmisión de mensajes
 Recepción de mensajes
 Destrucción de una cola de mensajes
 Bibliografía

Mensajes que se pueden transmitir

- ¿Qué mensajes se pueden transmitir?
 - Se transmiten estructuras cuyo primer miembro es un entero positivo tipo *long*
- Ejemplo:


```

      struct barco_pirata {
          long mtype;
          char nombre[30];
          char tipo_barco;
          int mala_fama;
          int crueldad;
          int valor_botin;
      };
      
```

75.59 - Técnicas de Programación Concurrente I
Colas de Mensajes

Introducción
 Creación de una cola de mensajes
Transmisión de mensajes
 Recepción de mensajes
 Destrucción de una cola de mensajes
 Bibliografía

Transmisión de mensajes

Función *msgsnd()*

```
int msgsnd ( int msqid,const void* msgp,size_t msgsz,int msgflg );
```

- Parámetros:
 - msqid: identificador de la cola en la cual se quiere transmitir
 - msgp: puntero a la estructura con el mensaje a transmitir
 - msgsz: tamaño del mensaje (sin contar el tamaño del miembro mtype)
 - msgflg: flags: IPC_NOWAIT: no esperar si la cola está llena
- Retorna:
 - 0 en caso de éxito
 - -1 en caso de error, seteando la variable externa *errno*

75.59 - Técnicas de Programación Concurrente I
Colas de Mensajes

Introducción
 Creación de una cola de mensajes
 Transmisión de mensajes
Recepción de mensajes
 Destrucción de una cola de mensajes
 Bibliografía

Recepción de mensajes (I)

Función *msgrcv()*

```
ssize_t msgrcv ( int msqid,void* msgp,size_t msgsz,long msgtyp,int
msgflg );
```

- Parámetros:
 - msqid: identificador de la cola de la cual se quiere recibir mensajes
 - msgp: puntero al buffer donde se almacenará el mensaje
 - msgsz: tamaño del buffer del mensaje (sin contar el tamaño del miembro mtype)
 - msgtyp: tipo del mensaje que se quiere recibir
 - msgflg: flags: qué hacer en caso en que no haya mensajes del tipo deseado en la cola
 - IPC_NOWAIT: la llamada retorna inmediatamente con error (no bloqueante)
 - MSG_NOERROR: retorna la porción de datos sin error si no hay espacio en el buffer

75.59 - Técnicas de Programación Concurrente I
Colas de Mensajes

Introducción
 Creación de una cola de mensajes
 Transmisión de mensajes
Recepción de mensajes
 Destrucción de una cola de mensajes
 Bibliografía

Recepción de mensajes (II)

Función *msgrcv()*

- Retorna:
 - cantidad de bytes copiados en el buffer sin contar el primer miembro de la estructura (tipo del mensaje) en caso de éxito
 - -1 en caso de error, seteando la variable externa *errno*

75.59 - Técnicas de Programación Concurrente I
Colas de Mensajes

Introducción
 Creación de una cola de mensajes
 Transmisión de mensajes
Recepción de mensajes
 Destrucción de una cola de mensajes
 Bibliografía

Recepción de mensajes (III)

El comportamiento de `msgrcv()` depende del valor del parámetro `msgtyp`:

msgtyp	msgrcv()
0	Toma el siguiente mensaje de la cola, sin importar el valor de <i>mtype</i> del mensaje
>0	Toma el siguiente mensaje de la cola con <i>mtype</i> igual a <i>msgtyp</i>
<0	Toma el siguiente mensaje de la cola con menor <i>mtype</i> , cuyo valor de <i>mtype</i> es menor o igual al valor absoluto de <i>msgtyp</i>

75.59 - Técnicas de Programación Concurrente I
Colas de Mensajes

Introducción
 Creación de una cola de mensajes
 Transmisión de mensajes
 Recepción de mensajes
Destrucción de una cola de mensajes
 Bibliografía

Destrucción de una cola de mensajes

Función `msgctl()`

```
int msgctl ( int msqid,int cmd,struct msqid_ds* buf );
```

- **Parámetros:**
 - `msqid`: identificador de la cola que se quiere eliminar
 - `cmd`: comando, en este caso es `IPC_RMID`
 - `buf`: parámetros del comando, en este caso es `NULL`
- **Retorna:**
 - 0 en caso de éxito
 - -1 en caso de error, seteando la variable externa `errno`

Comandos útiles:

- `ipcs`: muestra el listado de colas de mensajes creadas
- `ipcrm`: permite eliminar una cola de mensajes

75.59 - Técnicas de Programación Concurrente I
Colas de Mensajes

Introducción
Creación de una cola de mensajes
Transmisión de mensajes
Recepción de mensajes
Destrucción de una cola de mensajes
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

Colas de Mensajes

Fuentes de los ejemplos

Listado 1: Ejemplo 1

```

1  #ifndef EJEMPLO_1
2
3  #include <unistd.h>
4  #include <errno.h>
5  #include <sstream>
6  #include <string>
7  #include <vector>
8
9  #include "Cola.h"
10 #include "Productor.h"
11 #include "Consumidor.h"
12 #include "Producto.h"
13
14
15 int main () {
16
17     static const int CANTIDAD_CONSUMIDORES = 3;
18     static const int VUELTAS = 3;
19     static const std::string ARCHIVO = "main1.cc";
20
21     for ( int i=0;i<CANTIDAD_CONSUMIDORES;i++ ) {
22
23         int pid = fork ();
24
25         if ( pid == 0 ) {
26             std::stringstream nombre;
27             nombre << "Consumidor " << (i+1);
28
29             Cola<producto> cola ( ARCHIVO,i );
30
31             Consumidor consumidor ( nombre.str() );
32             consumidor.consumir ( cola,VUELTAS );
33             return 0;
34         }
35     }
36
37     std::vector<Cola<producto>*> colas;
38
39     for ( int i=0;i<CANTIDAD_CONSUMIDORES;i++ ) {
40         colas.push_back ( new Cola<producto>(ARCHIVO,i) );
41     }
42
43     Productor productor ( "Productor" );
44     productor.producir ( colas,VUELTAS );
45
46     for ( int i=0;i<CANTIDAD_CONSUMIDORES;i++ ) {
47         int resultado = colas[i]->destruir();
48
49         if ( resultado < 0 )
50             perror ( strerror(errno) );
51
52         delete ( colas[i] );
53     }
54
55     return 0;
56 }
57
58 #endif

```

Listado 2: Clase Cola

```

1  #ifndef COLA_H_
2  #define COLA_H_
3
4  #include <sys/types.h>
5  #include <sys/msg.h>
6  #include <sys/ipc.h>
7  #include <stdio.h>
8  #include <string>
9
10 template <class T> class Cola {
11     private:
12         key_t     clave;
13         int       id;
14
15     public:

```



```

16         Cola ( const std::string& archivo,const char letra );
17         ~Cola();
18         int escribir ( const T& dato ) const;
19         int leer ( const int tipo,T* buffer ) const;
20         int destruir () const;
21     };
22
23     template <class T> Cola<T> :: Cola ( const std::string& archivo,const char letra ) {
24         this->clave = ftok ( archivo.c_str(),letra );
25         if ( this->clave == -1 )
26             perror ( "Error en ftok" );
27
28         this->id = msgget ( this->clave,0777|IPC_CREAT );
29         if ( this->id == -1 )
30             perror ( "Error en msgget" );
31     }
32
33     template <class T> Cola<T> :: ~Cola () {
34     }
35
36     template <class T> int Cola<T> :: destruir () const {
37         int resultado = msgctl ( this->id,IPC_RMID,NULL );
38         return resultado;
39     }
40
41     template <class T> int Cola<T> :: escribir ( const T& dato ) const {
42         int resultado = msgsnd ( this->id,static_cast<const void*>(&dato),sizeof(T)-sizeof(long),0 );
43         return resultado;
44     }
45
46     template <class T> int Cola<T> :: leer ( const int tipo,T* buffer ) const {
47         int resultado = msgrcv ( this->id,static_cast<void *>(buffer),sizeof(T)-sizeof(long),tipo,0 );
48         return resultado;
49     }
50
51 #endif /* COLA_H_ */

```

Listado 3: Producto

```

1 #ifndef PRODUCTO_H_
2 #define PRODUCTO_H_
3
4 #define TIPO_PRODUCTO 1
5
6 typedef struct producto {
7     long mtype;
8     int numeroRandom;
9 } producto;
10
11 #endif /* PRODUCTO_H_ */

```

Listado 4: Clase Productor

```

1 #ifndef PRODUCTOR_H_
2 #define PRODUCTOR_H_
3
4 #include <string>
5 #include <string.h>
6 #include <time.h>
7 #include <stdlib.h>
8 #include <iostream>
9 #include <errno.h>
10 #include <unistd.h>
11 #include <vector>
12
13 #include "Cola.h"
14 #include "Producto.h"
15
16
17 class Productor {
18     private:
19         std::string nombre;
20         int calcularRandom () const;
21
22     public:
23         Productor ( const std::string& nombre );
24         ~Productor ();
25         void producir ( const std::vector<Cola<producto>*>& colas,const int vueltas )
                const;

```

```

26 };
27
28 #endif /* PRODUTOR_H_ */

```

Listado 5: Clase Productor

```

1  #include "Productor.h"
2
3  Productor :: Productor ( const std::string& nombre ) {
4      this->nombre = nombre;
5  }
6
7  Productor :: ~Productor () {
8  }
9
10 void Productor :: producir ( const std::vector<Cola<producto>*>& colas, const int vueltas )
    const {
11
12     producto prod;
13     prod.mtype = TIPO_PRODUCTO;
14
15     for ( int j=0; j<vueltas; j++ ) {
16         sleep ( 5 );
17         prod.numeroRandom = this->calcularRandom ();
18
19         std::vector<Cola<producto>*>::const_iterator it;
20
21         for ( it=colas.begin(); it!=colas.end(); it++ ) {
22             int resultado = (*it)->escribir ( prod );
23
24             if ( resultado < 0 )
25                 std::cout << "Error de escritura en la cola: " << strerror(
26                     errno) << std::endl;
27
28             else
29                 std::cout << "El productor [" << this->nombre << "] produce "
30                     << prod.numeroRandom << std::endl;
31
32             }
33         }
34     }
35
36     int Productor :: calcularRandom () const {
37         srand ( time(NULL) );
38         int resultado = rand() % 100;
39         return resultado;
40     }
41 }

```

Listado 6: Clase Consumidor

```

1  #ifndef CONSUMIDOR_H_
2  #define CONSUMIDOR_H_
3
4  #include <string>
5  #include <string.h>
6  #include <iostream>
7  #include <errno.h>
8  #include <sys/time.h>
9  #include <stdlib.h>
10 #include <unistd.h>
11
12 #include "Cola.h"
13 #include "Producto.h"
14
15
16 class Consumidor {
17     private:
18         std::string nombre;
19
20     public:
21         Consumidor ( const std::string& nombre );
22         ~Consumidor ();
23         void consumir ( const Cola<producto>& cola, const int vueltas ) const;
24 };
25
26 #endif /* CONSUMIDOR_H_ */

```

Listado 7: Clase Consumidor

```

1  #include "Consumidor.h"
2

```

```

3 Consumidor :: Consumidor ( const std::string& nombre ) {
4     this->nombre = nombre;
5     std::cout << "Consumidor [" << this->nombre << "] creado con process id = " << getpid
6     () << std::endl;
7 }
8 Consumidor :: ~Consumidor () {
9 }
10
11 void Consumidor :: consumir ( const Cola<producto>& cola, const int vueltas ) const {
12
13     for ( int i=0; i<vueltas; i++ ) {
14         sleep ( 1 );
15
16         // se lee el producto de la cola
17         producto prod;
18         int resultado = cola.leer ( TIPO_PRODUCTO, &prod );
19
20         if ( resultado < 0 )
21             std::cout << "Error de lectura en la cola: " << strerror(errno) << std
22             ::endl;
23         else
24             std::cout << "El consumidor [" << this->nombre << "] lee el producto "
25             << prod.numeroRandom << std::endl;
26     }
27 }

```

Listado 8: Ejemplo 2

```

1 #ifndef EJEMPLO_2
2
3 #include <iostream>
4 #include <sstream>
5 #include <unistd.h>
6 #include <stdlib.h>
7 #include <sys/wait.h>
8
9 #include "Cliente.h"
10 #include "Servidor.h"
11 #include "Mensajes.h"
12
13 using namespace std;
14
15 int main () {
16
17     static const int CANTIDAD_INTERCAMBIOS = 3;
18
19     int processId = fork ();
20
21     if ( processId == 0 ) {
22
23         // servidor
24         Servidor servidor ( "main2.cc", 'a' );
25
26         for ( int i=0; i<CANTIDAD_INTERCAMBIOS; i++ ) {
27             cout << "Servidor: esperando peticiones" << endl;
28             servidor.recibirPeticion ();
29             cout << "Servidor: peticion recibida: " << servidor.getPeticionRecibida
30             ().texto << endl;
31             servidor.procesarPeticion ();
32             cout << "Servidor: peticion procesada - enviando respuesta: " <<
33             servidor.getRespuesta().texto << endl;
34             servidor.responderPeticion ();
35             cout << "Servidor: respuesta enviada" << endl << endl;
36
37             return 0;
38         } else {
39
40             // cliente
41             Cliente cliente ( "main2.cc", 'a' );
42
43             for ( int i=0; i<CANTIDAD_INTERCAMBIOS; i++ ) {
44                 cin.get ();
45
46                 // se arma el texto del mensaje
47                 std::stringstream peticion;
48                 peticion << "Peticion " << (i+1) << " del cliente";
49

```

```

50 // se envia el mensaje al servidor
51 mensaje rta = cliente.enviarPeticion ( i+1,peticion.str() );
52 cout << "Cliente: respuesta recibida = (ID = " << rta.id << ") - " <<
    rta.texto << endl;
53     }
54
55     wait ( NULL );
56
57     return 0;
58 }
59 }
60
61 #endif

```

Listado 9: Mensajes

```

1 #ifndef MENSAJES_H_
2 #define MENSAJES_H_
3
4 #define PETICION      1
5 #define RESPUESTA    2
6 #define TEXTO_SIZE   255
7
8
9 typedef struct mensaje {
10     long mtype;
11     int id;
12     char texto[TEXTO_SIZE];
13 } mensaje;
14
15 #endif /* MENSAJES_H_ */

```

Listado 10: Clase Cliente

```

1 #ifndef CLIENTE_H_
2 #define CLIENTE_H_
3
4 #include <string>
5 #include <string.h>
6
7 #include "Mensajes.h"
8 #include "Cola.h"
9
10 class Cliente {
11
12     private:
13         Cola<mensaje>* cola;
14
15     public:
16         Cliente ( const std::string& archivo,const char letra );
17         ~Cliente();
18         mensaje enviarPeticion ( const int id,const std::string& texto ) const;
19 };
20
21 #endif /* CLIENTE_H_ */

```

Listado 11: Clase Cliente

```

1 #include "Cliente.h"
2
3 Cliente :: Cliente ( const std::string& archivo,const char letra ) {
4     this->cola = new Cola<mensaje> ( archivo,letra );
5 }
6
7 Cliente :: ~Cliente() {
8     this->cola->destruir ();
9     delete this->cola;
10 }
11
12 mensaje Cliente :: enviarPeticion ( const int id,const std::string& texto ) const {
13     mensaje peticion;
14     mensaje respuesta;
15
16     peticion.mtype = PETICION;
17     peticion.id = id;
18     strcpy ( peticion.texto,texto.c_str() );
19
20     this->cola->escribir ( peticion );
21     this->cola->leer ( RESPUESTA,&respuesta );

```

```

22         return respuesta;
23     }
24 }

```

Listado 12: Clase Servidor

```

1  #ifndef SERVIDOR_H_
2  #define SERVIDOR_H_
3
4  #include <string>
5  #include <string.h>
6  #include <sstream>
7
8  #include "Mensajes.h"
9  #include "Cola.h"
10
11 class Servidor {
12
13     private:
14         Cola<mensaje>* cola;
15         mensaje peticionRecibida;
16         mensaje respuesta;
17
18     public:
19         Servidor ( const std::string& archivo, const char letra );
20         ~Servidor ();
21
22         int recibirPeticion ();
23         int procesarPeticion ();
24         int responderPeticion () const;
25
26         mensaje getPeticionRecibida ();
27         mensaje getRespuesta ();
28 };
29
30 #endif /* SERVIDOR_H_ */

```

Listado 13: Clase Servidor

```

1  #include "Servidor.h"
2
3  Servidor :: Servidor ( const std::string& archivo, const char letra ) {
4      this->cola = new Cola<mensaje> ( archivo, letra );
5  }
6
7  Servidor :: ~Servidor () {
8      delete this->cola;
9  }
10
11 int Servidor :: recibirPeticion () {
12     this->peticionRecibida.id = 0;
13     this->cola->leer ( PETICION, &(this->peticionRecibida) );
14     return 0;
15 }
16
17 int Servidor :: procesarPeticion () {
18     std::stringstream textoRta;
19     textoRta << "[Respuesta a " << this->peticionRecibida.texto << "]\n";
20
21     this->respuesta.mtype = RESPUESTA;
22     this->respuesta.id = this->peticionRecibida.id;
23     strcpy ( this->respuesta.texto, textoRta.str().c_str() );
24
25     return 0;
26 }
27
28 int Servidor :: responderPeticion () const {
29     this->cola->escribir ( this->respuesta );
30     return 0;
31 }
32
33 mensaje Servidor :: getPeticionRecibida () {
34     return this->peticionRecibida;
35 }
36
37 mensaje Servidor :: getRespuesta () {
38     return this->respuesta;
39 }

```