Práctica 5: Locks

75.59 - Técnicas de Programación Concurrente I

Ejercicios

1. Estudiar la llamada al sistema *fcntl()*. Describir su funcionalidad y sus posibles argumentos.

2. Estudiar y describir la estructura *flock*, en relación con *fcntl()*.

3. Hacer un programa que lee un número desde un archivo, lo aumenta en un valor constante, lo presenta en pantalla y luego lo guarda en el disco. Esto lo hace repetidas veces.
   Usar *fcntl()* para administrar los accesos de dos procesos que quieren realizar la misma operación sobre un mismo archivo.
   Probar los diferentes tipos de bloqueos y describir los resultados.

4. Hacer un programa que muestre el bloqueo de archivos, con comprobación previa del estado de bloqueo del archivo que se quiere acceder y sin ella.

5. Un DBMS es un software que controla los accesos a y mantiene el modelo de información de una empresa. Se distinguen dos partes principales: la base de datos permanente dispuesta en archivos y las primitivas de acceso que posibilitan a las aplicaciones, almacenar, recuperar, modificar y eliminar datos.
   Un DBMS tiene un "lock manager", es decir, un mecanismo que mantiene un cerrojo para cada registro en uso por un proceso.
   Un cliente es una aplicación que utiliza el DBMS. Las peticiones de los clientes se realizan como transacciones sobre la base de datos.
   Sean T1, T2 y T3 las transacciones definidas sobre una base de datos tales que dado A un ítem en la base de datos:
   T1: suma 1 a A
   T2: duplica A
   T3: muestra A en una pantalla y luego dispone A en 1

   a) Suponer que T1, T2 y T3 se ejecutan concurrentemente. Si el valor inicial de A es cero, mostrar seis secuencias de ejecución correctas

   b) Suponer que la estructura interna de T1, T2 y T3 es la siguiente:

Transacción T1	**Transacción T2**	**Transacción T3**
1. Recuperar A en la variable v1	1. Recuperar A en v2	1. Recuperar A en v3
2. Sumar 1 a v1	2. Multiplicar v2 por 2	2. Mostrar v3
3. Actualizar A con v1	3. Actualizar A con v2	3. Modificar A de manera que sea 1

   Si las transacciones se ejecutan sin cerrojos, mostrar 10 posibles secuencias correctas.

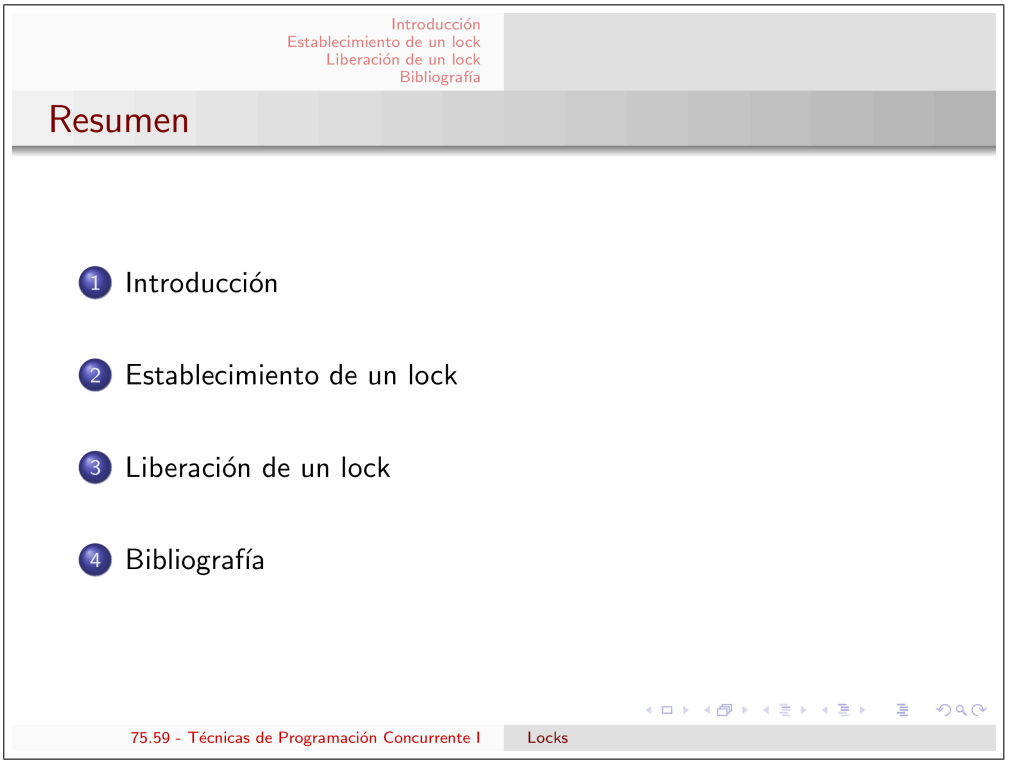
6. SQL es un lenguaje de cuarta generación especialmente diseñando para operar sobre bases de datos relacionales. No provee capacidades explícitas de manejo de cerrojos, sin embargo, incluye una cláusula SET TRANSACTION que se utiliza para definir ciertas características de la transacción siguiente a iniciarse. Una de esas características es el nivel de aislamiento (de la transacción). Los posibles niveles son:

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE (default)

El grado de aislamiento es creciente en el orden en que fueron enumerados.
El estándar de SQL también define en qué formas podría ser violada la serializabilidad. Investigarlas.
Ayuda: dirty read, non repeatable read, phantom.

Apuntes


Introducción
Establecimiento de un lock
Liberación de un lock
Bibliografía
Locks
75.59 - Técnicas de Programación Concurrente I
Facultad de Ingeniería - Universidad de Buenos Aires
75.59 - Técnicas de Programación Concurrente I
Locks


Introducción
Establecimiento de un lock
Liberación de un lock
Bibliografía
Resumen
1 Introducción
2 Establecimiento de un lock
3 Liberación de un lock
4 Bibliografía
75.59 - Técnicas de Programación Concurrente I
Locks

Introducción (I)

- Mecanismo de sincronismo de acceso a un archivo
- Se pueden utilizar también para sincronizar el acceso a cualquier otro recurso
- En Unix son *advisory*, es decir, los procesos pueden ignorarlos
- Dos tipos de locks:
  - Locks de lectura o *shared locks*: más de un proceso a la vez puede tener el lock
  - Locks de escritura o *exclusive locks*: sólo un proceso a la vez puede tener cualquier tipo de lock

Introducción (II)

Condiciones para poder tomar un lock

- Para poder tomar un *shared (read) lock*, el proceso debe esperar hasta que sean liberados todos los *exclusive locks*
- Para poder tomar un *exclusive (write) lock*, el proceso debe esperar hasta que sean liberados todos los locks (de ambos tipos)

Establecimiento de un lock (I)

Pasos para establecer un lock:

1. Abrir el archivo a lockear
2. Aplicar el lock
   1. Mediante *fcntl()*
      - Completar los campos de la estructura `struct flock`
      - Utilizar *fcntl()*
   2. Mediante *flock()*
   3. Función *lockf()*: interface construida sobre *fcntl()*

Establecimiento de un lock (II)

- Apertura del archivo: función *open()*

```
int open ( const char* pathname,int flags );
```

- Modo de apertura del archivo: depende del tipo de lock

Tipo de lock	Modo de apertura
F_RDLCK	O_RDONLY u O_RDWR
F_WRLCK	O_WRONLY u 0_RDWR

Establecimiento de un lock (III)

Método 1: Estructura `struct flock`

```
struct flock fl;
fl.l_type = F_WRLCK; // F_RDLCK, F_WRLCK o F_UNLCK
fl.l_whence = SEEK_SET; // SEEK_SET, SEEK_CUR o SEEK_END
fl.l_start = 0; // offset desde l_whence
fl.l_len = 0; // longitud, 0 = EOF
```

Miembros:

- l_type: tipo de lock, lectura o escritura
- l_whence: dentro del archivo a lockear, byte a partir del cual se quiere lockear
- l_start: byte de inicio del lock relativo a l_whence
- l_len: tamaño en bytes de la región del archivo que se quiere lockear

Establecimiento de un lock (IV)

Método 1: Aplicación del lock: función *fcntl()*

```
int fcntl ( int fd,int cmd, ... );
```

Comandos de *fcntl()*

- F_SETLKW: intentará obtener el lock. Si no se puede, se bloquea hasta que se libere el lock
- F_SETLK: intentará obtener el lock. Si no se puede, no se bloquea y retorna -1. Se usa para liberar el lock
- F_GETLK: se puede usar para verificar si hay un lock

Establecimiento de un lock (V)

Método 2: Función `int flock ( int fd,int operation );`

- Preserva los locks establecidos a través de `fork()` y `execve()`
- Parámetros:
  - fd: file descriptor obtenido con *open()*
  - operation: LOCK_SH, LOCK_EX o LOCK_UN
- Retorna:
  - 0 en caso de éxito
  - -1 en caso de error, seteando la variable externa *errno*

Liberación de un lock

- Primer paso: liberar el lock

```
fl.l_type = F_UNLCK;
fcntl ( fd,F_SETLK,&fl );
```

- Segundo paso: cerrar el archivo

```
int close ( int fd );
```

Bibliografía

- Manuales del sistema operativo
- *Unix Network Programming, Interprocess Communications*, W. Richard Stevens, segunda edición

Fuentes de los ejemplos

Listado 1: Ejemplo 1

```
#ifdef EJEMPLO_1

#include <iostream>
#include <stdlib.h>

#include "LockFile.h"

using namespace std;

int main () {

        pid_t processId = fork ();

        if ( processId == 0 ) {
                // hijo
                cout << "Empieza el hijo" << endl;
                LockFile lock ( "main1.cc" );

                sleep ( 1 );
                cout << "Hijo: esperando para tomar lock..." << endl;
                lock.tomarLock();
                cout << "Hijo: tome el lock" << endl;
                sleep ( 10 );
                lock.liberarLock();
                cout << "Hijo: libere el lock y termino" << endl;

                return ( 0 );

        } else {
                // padre
                cout << "Empieza el padre" << endl;
                LockFile lock ( "main1.cc" );

                lock.tomarLock();
                cout << "Padre: tome el lock" << endl;
                sleep ( 5 );
                lock.liberarLock();
                cout << "Padre: libere el lock y termino" << endl;

                return ( 0 );
        }
}

#endif
```

Listado 2: Ejemplo 2

```
#ifdef EJEMPLO_2

#include <iostream>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

using namespace std;

int main () {

        static const std::string NOMBRE_ARCHIVO = "salida.txt";
        static const std::string TEXTO_HIJO = "Mensaje del hijo\n";
        static const std::string TEXTO_PADRE = "Mensaje del padre\n";

        pid_t pid = fork ();

        if ( pid == 0 ) {

                int descriptor = open ( NOMBRE_ARCHIVO.c_str(),O_CREAT | O_WRONLY,0777 );

                if ( descriptor > 0 ) {
                        lseek ( descriptor,0,SEEK_END );
                        write ( descriptor,static_cast<const void*>(TEXTO_HIJO.c_str()),
                            TEXTO_HIJO.size() );
                        sleep ( 1 );

                        lseek ( descriptor,0,SEEK_END );
```

```
28                          write ( descriptor,static_cast<const void*>(TEXTO_HIJO.c_str()),
                                TEXTO_HIJO.size() );
29
30                          lseek ( descriptor,0,SEEK_END );
31                          write ( descriptor,static_cast<const void*>(TEXTO_HIJO.c_str()),
                                TEXTO_HIJO.size() );
32
33                          close ( descriptor );
34                  } else {
35                          perror ( "Hijo: error al abrir archivo" );
36                  }
37
38                  cout << "Hijo: fin del programa" << endl;
39                  return 0;
40
41          } else {
42
43                  int descriptor = open ( NOMBRE_ARCHIVO.c_str(),O_CREAT | O_WRONLY,0777 );
44
45                  if ( descriptor > 0 ) {
46                          lseek ( descriptor,0,SEEK_END );
47                          write ( descriptor,static_cast<const void*>(TEXTO_PADRE.c_str()),
                                TEXTO_PADRE.size() );
48
49                          lseek ( descriptor,0,SEEK_END );
50                          write ( descriptor,static_cast<const void*>(TEXTO_PADRE.c_str()),
                                TEXTO_PADRE.size() );
51                          sleep ( 1 );
52
53                          lseek ( descriptor,0,SEEK_END );
54                          write ( descriptor,static_cast<const void*>(TEXTO_PADRE.c_str()),
                                TEXTO_PADRE.size() );
55
56                          close ( descriptor );
57                  } else {
58                          perror ( "Padre: error al abrir archivo" );
59                  }
60
61                  cout << "Padre: fin del programa" << endl;
62                  return 0;
63
64          }
65
66  }
67
68  #endif
```

Listado 3: Ejemplo 3

```
1   #ifdef EJEMPLO_3
2
3   #include <iostream>
4   #include "LockFile.h"
5
6   using namespace std;
7
8   int main () {
9
10          static const std::string NOMBRE_ARCHIVO = "salida.txt";
11          static const std::string TEXTO_HIJO = "Mensaje del hijo\n";
12          static const std::string TEXTO_PADRE = "Mensaje del padre\n";
13
14          pid_t pid = fork ();
15
16          if ( pid == 0 ) {
17
18                  LockFile lock ( NOMBRE_ARCHIVO );
19                  lock.tomarLock ();
20
21                  lock.escribir ( static_cast<const void*>(TEXTO_HIJO.c_str()),TEXTO_HIJO.size()
                        );
22                  sleep ( 1 );
23                  lock.escribir ( static_cast<const void*>(TEXTO_HIJO.c_str()),TEXTO_HIJO.size()
                        );
24                  lock.escribir ( static_cast<const void*>(TEXTO_HIJO.c_str()),TEXTO_HIJO.size()
                        );
25
26                  lock.liberarLock ();
27                  cout << "Hijo: fin del programa" << endl;
28                  return 0;
29
```

```
30          } else {
31
32                  LockFile lock ( NOMBRE_ARCHIVO );
33                  lock.tomarLock ();
34
35                  lock.escribir ( static_cast<const void*>(TEXTO_PADRE.c_str()),TEXTO_PADRE.size
                        () );
36                  lock.escribir ( static_cast<const void*>(TEXTO_PADRE.c_str()),TEXTO_PADRE.size
                        () );
37                  sleep ( 1 );
38                  lock.escribir ( static_cast<const void*>(TEXTO_PADRE.c_str()),TEXTO_PADRE.size
                        () );
39
40                  lock.liberarLock ();
41                  cout << "Padre: fin del programa" << endl;
42                  return 0;
43
44          }
45
46  }
47
48  #endif
```

Listado 4: Clase LockFile

```
1   #ifndef LOCKFILE_H_
2   #define LOCKFILE_H_
3
4   #include <unistd.h>
5   #include <fcntl.h>
6   #include <string>
7
8   class LockFile {
9
10  private:
11          struct flock fl;
12          int fd;
13          std::string nombre;
14
15  public:
16          LockFile ( const std::string nombre );
17          ~LockFile();
18
19          int tomarLock ();
20          int liberarLock ();
21          ssize_t escribir ( const void* buffer,const ssize_t buffsize ) const;
22  };
23
24  #endif /* LOCKFILE_H_ */
```

Listado 5: Clase LockFile

```
1   #include "LockFile.h"
2
3   LockFile :: LockFile ( const std::string nombre ) {
4
5           this->nombre = nombre;
6           this->fl.l_type = F_WRLCK;
7           this->fl.l_whence = SEEK_SET;
8           this->fl.l_start = 0;
9           this->fl.l_len = 0;
10          this->fd = open ( this->nombre.c_str(),O_CREAT|O_WRONLY,0777 );
11  }
12
13  int LockFile :: tomarLock () {
14          this->fl.l_type = F_WRLCK;
15          return fcntl ( this->fd,F_SETLKW,&(this->fl) );
16  }
17
18  int LockFile :: liberarLock () {
19          this->fl.l_type = F_UNLCK;
20          return fcntl ( this->fd,F_SETLK,&(this->fl) );
21  }
22
23  ssize_t LockFile :: escribir ( const void* buffer,const ssize_t buffsize ) const {
24          lseek ( this->fd,0,SEEK_END );
25          return write ( this->fd,buffer,buffsize );
26  }
27
28  LockFile :: ~LockFile () {
```

```
29          close ( this->fd );
30  }
```