

Clase 5

En la clase anterior

Problemas con Pipes/Fifos:

- Lectura no atómica -> No pueden leer muchos
 - Si quiero hacer esto, necesito evitar que todos lean a la vez

Problemas con Archivos:

- No proveen un mecanismo de sincronización

Clase 5

Locks

Locks

Mecanismo de sincronismo de acceso a un archivo

- Están pensados para archivos estructurados
- En general se utilizan para sincronizar el acceso a cualquier otro recurso

⚠ En unix, los locks son **advisor** (pueden ser ignorados) ⚠

Tipos

- Exclusivos (*Exclusive Locks*)
 - También conocido como *Lock de Escritura*
 - Solo un proceso a la vez puede tomar el lock
- Compartidos (*Shared Locks*)
 - También conocido como *Lock de Lectura*
 - Más de un proceso puede tomar el mismo lock
 - Se usan en conjunto con los exclusivos

Condiciones

- Exclusivos
 - Para poder tomar el lock, el proceso debe esperar a que no haya locks de ningun tipo tomados
- Compartidos
 - Para poder tomar el lock, el proceso debe esperar a que no haya locks exclusivos tomados

Condiciones (cheatsheet)

		Lock pedido	
		Compartido (Lectura)	Exclusivo (Escritura)
El archivo tiene	Ningún lock	OK	OK
	Uno o más locks de lectura	OK	Denegado
	Un lock de escritura	Denegado	Denegado

Operaciones

Operaciones

Operación	Locks
Obtener	?
Liberar	?

💡 Funcionan sobre archivos 💡

💡 no hay operaciones de creación/destrucción **propias de locks** 💡

Flujo general

1. Abrir el archivo a lockear

2. Aplicar el lock:

- Opciones

- `fcntl`

- Completando los campos de la estructura `struct flock`

- `flock`

- `lockf` (Es una interfaz construida sobre `fcntl`)

Apertura del archivo

open

Apertura del archivo `open`

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

El modo de apertura depende del tipo de lock

- Lock Exclusivo
 - `O_WRONLY` / `O_RDWR`
- Lock Compartido
 - `O_RDONLY` / `O_RDWR`

Establecimiento del lock

`fcntl`

(Método 1)

Estructura `flock`

- `l_type` : Tipo de lock
 - `F_WRLCK` : Exclusivo
 - `F_RDLCK` : Compartido
 - `F_UNLCK` : Liberar el lock
- `l_whence` : Dentro del archivo, byte desde el cual se quiere lockear
 - `SEEK_SET` : Inicio del archivo
 - `SEEK_CUR` : Posición actual del cursor
 - `SEEK_END` : Desde el final

Estructura `flock`

- `l_start` : Byte de inicio del lock relativo a `l_whence`
 - Con `SEEK_CUR` y `SEEK_END`, puede ser un valor negativo
- `l_len` : Tamaño de la región a lockear
 - Con 0 se bloquea hasta el `EOF`

Estructura flock

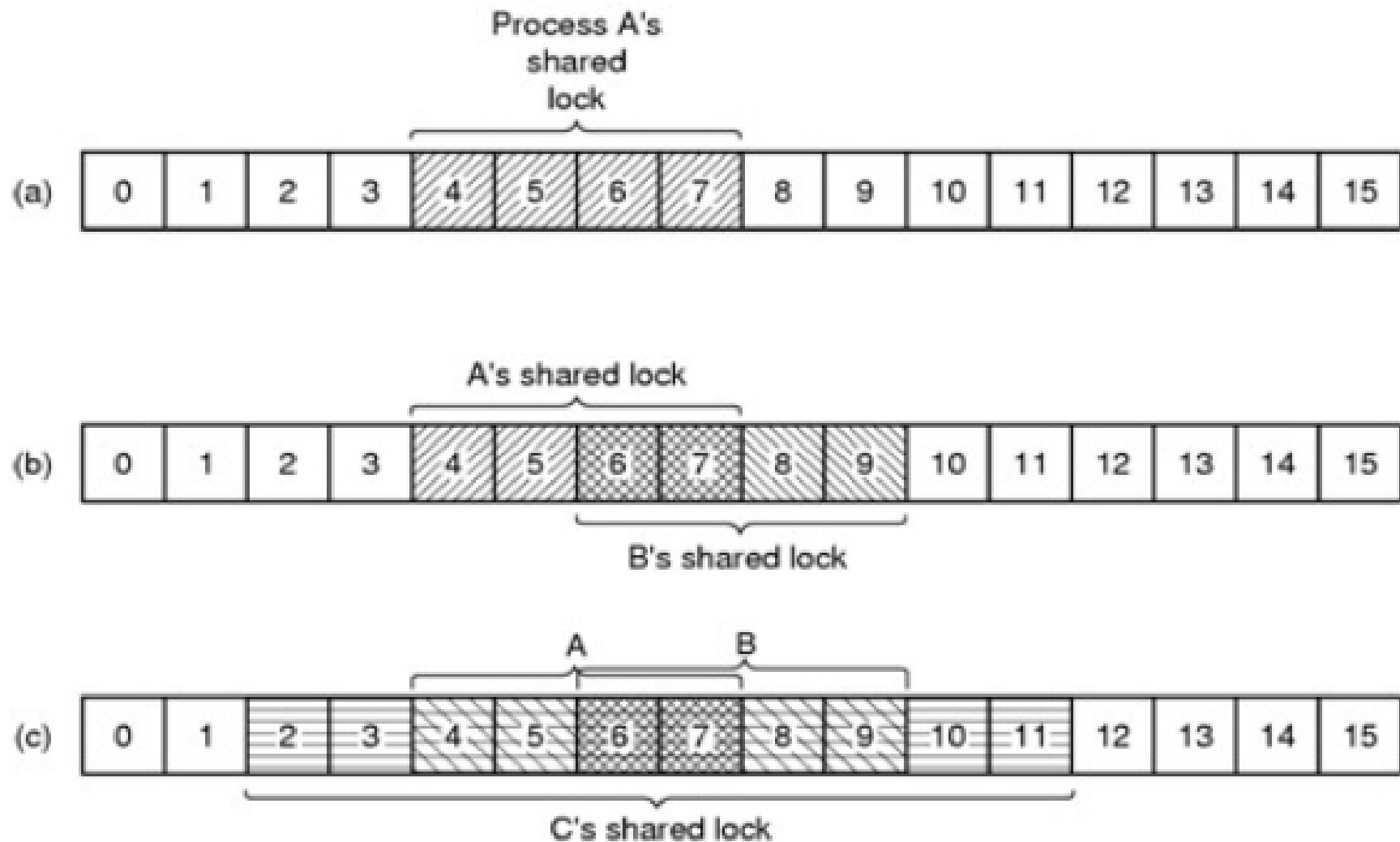
Ejemplo de definición de la estructura

```
#include <fcntl.h>
#include <unistd.h>

struct flock fl;

fl.l_type = F_WRLCK; //Exclusive lock
fl.l_whence = SEEK_SET; //From start
fl.l_start = 0; // No offset from start
fl.l_len = 0; // End of file
```


Múltiples locks



Establecimiento con `fcntl`

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

Comando a utilizar:

- `F_SETLK`: Intenta tomar el lock
 - Si no puede, no se bloquea y retorna -1
 - Se usa para liberar el lock
- `F_SETLKW`: Intenta tomar el lock
 - Si no puede, se bloquea hasta que se libere el lock
 - 💡 File **set lock wait**

Establecimiento con `fcntl`

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

- `F_GETLK`: Se usa para verificar si hay un lock
 - El campo `l_type` es `F_UNLCK`
 - Si hay un lock, retorna el pid del proceso que lo tiene en el campo `l_pid` del `struct flock`

Establecimiento con `fcntl`

```
int fd = open (LOCKFILE_PATH, O_CREAT | O_WRONLY, 0777);

struct flock fl;

fl.l_type = F_WRLCK; //Exclusive lock
fl.l_whence = SEEK_SET; //From start
fl.l_start = 0; // No offset from start
fl.l_len = 0; // End of file

fcntl (fd, F_SETLKW, fl);
```

Liberación del lock

`fcntl`

(Método 1)

Liberación del lock `fcntl`

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

- Para liberar el lock se opera de forma similar que para obtenerlo
 - El campo `l_type` del `struct flock` en `F_UNLCK`

⚠ No hay que olvidarse de cerrar el archivo si no se va a volver a utilizar el lock ⚠

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

Liberación del lock `fcntl`

```
int fd = open (LOCKFILE_PATH, O_CREAT | O_WRONLY, 0777);

struct flock fl;

fl.l_type = F_WRLCK; //Exclusive lock
fl.l_whence = SEEK_SET; //From start
fl.l_start = 0; // No offset from start
fl.l_len = 0; // End of file

fcntl (fd, F_SETLKW, fl);

/*DO SOMETHING IMPORTANT*/

fl.l_type = F_UNLCK; //Release lock

fcntl (fd, F_SETLK, fl);
close(fd);
```

Manejo de locks con

`flock`

(Método 2)

Manejo de locks con `flock`

```
int flock (int fd, int operation);
```




- `fd`: File descriptor del archivo de lock
- `operation`: Operación a realizar `LOCK_SH` / `LOCK_EX` / `LOCK_UN`
- Retorna 0 en caso de éxito
 - -1 en caso de error, y setea la variable externa `errno`
- **!** Los locks obtenidos se preservan a través de `fork` y `execve`
 - **!** Dos procesos pueden tener el mismo lock exclusivo **!**

Recap

Recap Operaciones

Operación	Archivo	Locks	flock l_type
Abrir	<code>open</code>	-	
Obtener	-	<code>fcntl</code> + <code>F_SETLKW</code>	<code>F_WRLCK</code> / <code>F_RDLCK</code>
Liberar	-	<code>fcntl</code> + <code>F_SETLK</code>	<code>F_UNLCK</code>
Cerrar	<code>close</code>	-	

Bibliografía

-  *Unix Network Programming, Interprocess Communications*
 -  W. Richard Stevens
-  [Manuales del sistema operativo](#)