Clase de Locks

Tomás Arjovsky

11/4/2018

fcntl

```
fnctl(int fd, int cmd, struct flock * fl);
```

`fcntl` es una función general que permite hacer muchas cosas sobre file descriptors. Aplica el comando `cmd` sobre el file descriptor `fd`.

Los comandos para lock son:

- `GETLK`:
- `SETLKW`: intentará obtener lock o bloquea.
- `SETLK`: intenta obtener lock o retorna -1.
- `GETLK`: verifica si hay lock o no.

flock es más fácil, pero es sobre todo el archivo. flock(int fd, int operation); LOCK_SH, LOCK_EX, LOCK_UN (shared, exclusivo, unlock).

El tercer argumento varía según `cmd`, y para el caso de locks es un puntero a `struct flock`.

```
fl.l_type = F_WRLCK; // F_RDLCK,
fl.l_whence = SEEK_SET;
fl.l_start = 0; //
fl.l_len = 0; // longitud. 0 es eof.
```

Se puede bloquear de a porciones, no necesariamente todo el archivo.

- type: se utiliza para aclarar si es de lectura (`FD_RDLCK`) o de escritura (`FD_WRLCK`). El de lectura no es exclusivo, pero el de escritura sí.
- whence: donde comienza la sección de bloqueo.
  - `SEEK_SET` es para mover al ppio de un archivo.

  - `SEEK_CUR` es a donde está el puntero actual.
  - `SEEK_END` es el final del archivo.
- start: offset desde whence. Puede ser negativo para `SEEK_CUR` y `SEEK_END`.
- len: tamaño del área bloqueada en bytes.

El lock calcula las superposiciones. Si alguien pidió todo el archivo y otro pide una sola sección, este segundo deberá esperar a que el que pidió todo legue el suyo.

fcntl no se preserva con el fork (????).

El recurso lock se libera finalmente como cualquier archivo, con:

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

flock

Típicamente un wrapper de fcntl para locks, con pocas opciones.

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

Aplica la operación sobre el arhcivo abierto `fd`. Hay 3 operaciones:

- `LOCK_SH`: toma el lock de forma compartida.
- `LOCK_EX`: toma el lock de forma exclusiva.
- `LOCK_UN`: libera el lock.

lockf

Un poco más general que flock. Permite aclarar `len` desde la posición actual y permite testear. (Completar.)

Ejemplo (algunos comentarios)

Se crea un lock con el nombre del archivo. Se configura el struct. Se lo abre con:

```
open(nombre, O_CREAT | O_WRONLY, 0777);
```

El nombre es un `const char *`, si tenemos una string hay que aplicarle el método `c_str()`. El struct de configuración se le pasa al tomar el lock o al liberarlo con

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

La struct para tomar un lock para escritura tiene los siguientes campos:

```
fl.l_type = F_WRLCK;
fl.l_whence = SEEK_SET;
fl.l_start = 0;
fl.l_len = 0;
```

Para mover el puntero al final del archivo utilizamos:

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

Podemos usar fopen en lugar de open, ¿no?

Erlang

¿Qué es lo que pasa si decimos "los locks no sirven"?. ¿Qué hacemos? Queremos sincronizar y que ambos escriban sobre un archivo. Uno sumando 1000 y otro restando.

Patrón de programación defensiva: capturar todos los errores todo el tiempo. En C es necesario. En Python, Go y demás se puede.

Erlang básicamente su forma de manejar errores es tirar el proceso. Que muera. Hay otro que lo reemplazará.

Arity, funcional, error. No sabemos quién nos llama, en qué contexto, etc. Eso se va a encargar. El try y el catch de Erlang funcionan de otro modo.

En Erlang típicamente se usa `exit:_-> 0`. Pero esto ya lo hace Erlang por default.

Hacer un catch de todo no tiene sentido. Ya se hace y su respuesta es precisamente hacer un exit.