

CS810D - Advanced Programming in the UNIX Environment

—

Advanced I/O: Nonblocking I/O, Polling, and Record Locking

Department of Computer Science
Stevens Institute of Technology
Jan Schaumann

`jschauma@cs.stevens.edu`

<http://www.cs.stevens.edu/~jschauma/810-APUE/>

Nonblocking I/O

Certain system calls can block forever:

- `read(2)` from a particular file, if data isn't present (pipes, terminals, network devices)
- `write(2)` to the same kind of file
- `open(2)` of a particular file until a specific condition occurs
- `read(2)` and `write(2)` of files that have mandatory locking enabled
- certain `ioctl(2)`
- some IPC functions (such as `sendto(2)` or `recv(2)`)

Nonblocking I/O lets us issue an I/O operation and not have it block forever. If the operation cannot be completed, return is made immediately with an error noting that the operating would have blocked.

Advisory Locking

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

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

Returns: 0 if OK, -1 otherwise

- applies or removes an advisory lock on the file associated with the file descriptor *fd*
- *operation* can be
 - LOCK_SH
 - LOCK_EX
 - LOCK_NB
 - LOCK_UN
- locks entire file

Advisory “Record” Locking

Record locking is done using `fcntl(2)`, using one of `F_GETLK`, `F_SETLK` or `F_SETLKW` and passing a

```
struct flock {
    short l_type;    /* F_RDLCK, F_WRLCK, or F_UNLCK */
    off_t l_start;  /* offset in bytes from l_whence */
    short l_whence; /* SEEK_SET, SEEK_CUR, or SEEK_END */
    off_t l_len;    /* length, in bytes; 0 means "lock to EOF" */
    pid_t l_pid;    /* returned by F_GETLK */
}
```

Lock types are:

- `F_RDLCK` – Non-exclusive (read) lock; fails if write lock exists.
- `F_WRLCK` – Exclusive (write) lock; fails if any lock exists.
- `F_UNLCK` – Releases our lock on specified range.

Advisory “Record” locking

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

```
int lockf(int fd, int value, off_t size);
```

Returns: 0 on success, -1 on error

Convenience library function.

value can be:

- F_ULOCK – unlock locked sections
- F_LOCK – lock a section for exclusive use
- F_TLOCK – test and lock a section for exclusive use
- F_TEST – test a section for locks by other processes

Memory Mapped I/O

```
#include <sys/types.h>
#include <sys/mman.h>

void *mmap(void *addr, size_t len, int prot, int flags, int fd, off_t offset);
```

Returns: pointer to mapped region if OK

Protection specified for a region:

- PROT_READ – region can be read
- PROT_WRITE – region can be written
- PROT_EXEC – region can be executed
- PROT_NONE – region can not be accessed

flag needs to be one of

- MAP_SHARED
- MAP_PRIVATE
- MAP_COPY

which may be OR'd with other flags (see `mmap(2)` for details).

I/O Multiplexing

```
#include <sys/types.h>
#include <sys/time.h>
#include <unistd.h>

int select(int maxfdp1, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *tvptr);
```

Returns: count of ready descriptors, 0 on timeout, -1 otherwise

Arguments passed:

- which descriptors we're interested in
- what conditions we're interested in
- how long we want to wait

`select(2)` tells us

- total count of descriptors that are ready
- which descriptors are ready

Homework

Work on your final.
Seriously.