# Linux System Programming #10
# Lecture Notes

**Spring 2020**

**School of Computer Science and Engineering,**

**Soongsil University, Seoul, Korea**

**Jiman Hong**

**jiman@acm.org**

# Blocking IO vs Nonblocking IO

- blocking과 non-blocking
  - 연산이나 시스템 콜을 포함한 함수 호출과 관련된 연산 방법
- blocking I/O
  - 응용 프로그램에서 I/O 시스템 호출 함수가 호출될 때 호출된 시스템 호출 함수와 관련한 커널 함수가 수행이 완료된 후 결과를 리턴
  - 다소 시간이 많이 걸리는 작업으로 예를 들어 데이터를 읽을(read()) 때 함수 호출의 리턴이 올 때까지 기다림
  - 이 경우 CPU는 다른 프로세스가 선택하여 처리하거나 CPU는 idle 상태
- nonblocking I/O
  - 시스템 호출 함수와 관련한 커널 함수를 호출하기 전에 바로 리턴
  - blocking I/O의 비효율성을 극복하고자 만들어진 방법으로 I/O가 진행되는 동안 응용 프로그램을 대기시키지 않고 I/O 요청에 대해 바로 결과를 리턴하고 응용 프로그램을 계속 진행시킴
  - I/O의 완료와 상관없이 함수 호출의 바로 리턴
  - 연산이 완료될 수 없는 경우에도 호출이 차단되지 않고 에러 코드를 설정 후 즉시 리턴
- IO => schedule ()
  - I/O를 실행한 응용 프로그램은 I/O 컨트롤러에 의해 인터럽트가 오기 전까지 대기(block, sleep)
  - blocking I/O 방식은 특수 파일들(파이프, 터미널 장치, 네트워크 장치)에서 몇 가지 문제점이 나타날 수 있음
    - 특정 조건이 발생할 때까지 오픈 연산이 대기되거나, 파일이 존재하지 하지 않을 경우 read 연산이 계속 대기
    - 파이프가 꽉 차 있거나 네트워크 연결에 문제가 있을 경우 쓰기 연산이 대기
    - 필수 레코드 잠금이 활성화된 파일을 읽거나 쓰는 연산들도 생각 이상으로 대기
- non-blocking I/O 지정
  - open()로 파일 디스크립터를 오픈할 때 O_NONBLOCK 플래그를 지정
  - 이미 오픈한 파일 디스크립터 대해, fcntl()를 이용해서 파일 상태 플래그 O_NONBLOCK 플래그 지정

# fcntl(2)

```
#include <unistd.h>
#include <fcntl.h>
int fcntl(int fd, int cmd, ... /*arg*/);
int fcntl(int fd, int cmd);
int fcntl(int fd, int cmd, long arg);
int fcntl(int fd, int cmd, struct flock *lock);
```

리턴 값: 성공 시 cmd에 따라 다름(아래 설명 참고), 에러 시 -1

- fcntl()에서 제공하는 다양한 기능들은 두 번째 인자 cmd 값에 의해 결정
- (1) 오픈한 파일의 속성을 가져오거나 변경
  - open()이나 fopen()을 사용할 때 mode 인자를 이용하여 파일의 속성을 설정하고 open된 파일을 다른 프로세스가 읽거나 쓰는 것을 막을 수 있음 => 오픈한 파일을 닫고, 다시 mode 인자를 변경하여 파일의 속성을 변경
  - fcntl()을 사용하면 오픈한 파일을 닫지 않고 파일의 속성을 바로 변경하여 다른 프로세스가 읽거나 쓰는 것을 막을 수 있음
  - fcntl()은 오픈한 파일에 대해서 여러 번 속성 변경 가능

# cmd 값에 대한 fcntl()의 기능

| 플래그 | 설명 |
|---|---|
| F_DUPFD | fd로 지정된 파일 디스크립터를 복사. dup2()와 유사하나 dup2()는 파일 디스크립터를 프로그래머가 지정해 주지만 fcntl()는 이미 해당 파일 디스크립터가 사용되고 있으면 arg 인자 보다 큰 할당 가능한 파일 디스크립터 번호 중 가장 작은 번호를 리턴. 복사된 파일 디스크립터가 이미 사용되어지고 있다면, arg 인자 보다 큰 할당 가능한 파일 디스크립터 번호 중 가장 작은 번호를 라턴. 복사된 파일 디스크립터는 close-on-exec()는 공유되지 않지만 락, 파일 위치 포인터, 파일 플래그 등은 공유 |
| F_GETFD | fd로 지정된 파일 디스크립터의 플래그를 리턴하는데 arg 인자는 무시 (FD_CLOEXEC 값을 리턴) |
| F_SETFD | fd로 지정된 파일 디스크립터의 플래그를 arg 인자에서 지정한 플래그 값으로 재설정 (FD_CLOEXEC 값을 지정된 플래그 값으로 설정) |
| F_GETFL | fd로 지정된 파일 디스크립터가 open할 때 지정한 파일의 접근 권한과 상태 플래그(file status flag)를 리턴하는데 arg 인자는 무시 |
| F_SETFL | fd로 지정된 파일 디스크립터의 파일 상태 플래그를 arg 인자에 지정한 플래그 값(O_APPEND, O_NONBLOCK, O_ASYNC, O_DIRECT)으로 재설정. arg 인자에서 접근 권한 플래그(O_RDONLY, O_WRONLY, O_RDWR) 및 파일 생성 플래그(O_CREAT, O_EXCL, O_NOCTTY, O_TRUNC)를 지정하면 무시 |
| F_GETLEASE | 파일 디스크립터에 담긴 lease 타입 |
| F_GETOWN | 파일 디스크립터의 소유자 프로세스 ID |
| F_GETSIG | 읽기나 쓰기가 가능해 질 때 보내지는 시그널 값 |
| F_GETPIPE_SZ, F_GETPIPE_SZ | 파이프 용량 |
| F_GET_SEALS | 파일 디스크립터에 의해서 참조되는 inode에 대한 seal을 인식하는 비트 마스크 |
| 그 외 | 0 |

# fcntl(2)

```
#include <unistd.h>
#include <fcntl.h>
int fcntl(int fd, int cmd, ... /*arg*/);
int fcntl(int fd, int cmd);
int fcntl(int fd, int cmd, long arg);
int fcntl(int fd, int cmd, struct flock *lock);
```

리턴 값: 성공 시 cmd에 따라 다름(아래 설명 참고), 에러 시 -1

- fcntl()에서 제공하는 다양한 기능들은 두 번째 인자 cmd 값에 의해 결정
- (2) 파일 전체 뿐만 아니라 일부를 락 설정
  - cmd 인자에 F_GETLK나 F_SETLK, F_SETLKW 지정
  - flockptr 인자
  - flock 구조체를 가리키는 포인터를 지정
  - 레코드 락 함수라고 불리기도 함
    - 파일의 일부를 락을 설정 가능
    - 프로세스가 파일의 일정 영역을 읽거나 수정하는 동안 다른 프로세스들이 그 영역을 수정하지 못하도록 락을 설정
    - 레코드 단위의 락은 파일 안의 한 범위, 즉 일부 또는 전체를 락을 설정하는 기능을 지원

# 락을 걸기 위해 사용되는 fcntl()의 cmd 인자

| cmd | 설명 |
|---|---|
| F_GETLK | 락 상태 정보를 세 번째 struct flock *flockpt 인자를 통해 확인할 때 사용.<br>lockptr로 지정된 락을 설정할 수 없게 만드는 다른 어떤 락이 존재하는지 파악하는데 쓰이며, 기존 락 때문에 새 락을 설정할 수 없는 상황이면 기존 락에 대한 정보가 flockptr가 가리키는 구조체에 write. 기존 락이 없어서 새 락을 설정하는 것이 가능한 상황이면 flockptr 구조체의 l_type 필드만 F_UNLCK으로 설정되고 나머지 필드들은 변하지 않음 |
| F_SETLK | 락을 설정할 때 사용하며 다른 프로세스가 이미 락을 설정 했을 경우 –1을 리턴.<br>flockptr로 지정된 락을 걸수 있음. read 락이나 write 락을 설정하려고 하는데 호환성 규칙 때문에 시스템이 잠금을 거부한 경우, fcntl는 ermo를 EACCES나 EAGAIN으로 설정하고 즉시 리턴 |
| F_SETLKW | 다른 프로세스가 이미 락을 설정 했을 경우 락을 해제할 때까지 기다림.<br>F_SETLK의 차단 버전(W는 wait를 뜻함). 다른 프로세스가 요청된 영역의 일부를 잠그고 있어서 read 락이나 write 락을 설정할 수 없으면 fcntl을 호출한 프로세스는 수면에 들어갔다가 이후 락이 가능해지거나 시그널에 의해 가로채기가 일어날 때 깨어남 |

# flock 구조체의 멤버 변수

```
struct flock {
short l_type; // F_RDLCK, F_WRLCK, 또는 F_UNLCK
short l_whence; // SEEK_SET, SEEK_CUR, 또는 SEEK_END
__off_t l_start; // 바이트 단위 오프셋, l_whence와 관련 있음
__off_t l_len; // length, in bytes; 0 means lock to EOF */
__pid_t l_pid; // returned with F_GETLK
};
```

| l_type | 의미 |
|---|---|
| F_RDLCK | 다른 프로세스가 읽기 락만 가능, 쓰기 락은 불가능 |
| F_WRLCK | 다른 프로세스는 읽기 락과 쓰기 락 모두 불가능 |
| F_UNLCK | 락 해제 |

# 락이 설정될 때 read(), write() 호출 규칙

| 다른 프로세스가 영역에 건 락의 종류 | 차단 파일 디스크립터 | | 비차단 파일 디스크립터 | |
|---|---|---|---|---|
| | read | write | read | write |
| read 락 | OK | 차단됨 | OK | EAGAIN |
| write 락 | 차단됨 | 차단됨 | EAGAIN | EAGAIN |

# 파일의 부분/전체 락을 설정하는 함수 코드

```
#include <fcntl.h>
int lock_reg(int fd, int cmd, int type, off_t offset, int whence, off_t len)
{
    struct flock lock;
    lock.l_type = type;             /* F_RDLCK, F_WRLCK, F_UNLCK */
    lock.l_start = offset;          /* 바이트 오프셋, l_whence와 관련 */
    lock.l_whence = whence;         /* SEEK_SET, SEEK_CUR, SEEK_END */
    lock.l_len = len;               /* #bytes (0 means to EOF) */
    return(fcntl(fd, cmd, &lock));
}
```

```
#include <unistd.h>
#include <fcntl.h>
int lockfile(int fd)
{
    struct flock fl;
    fl.l_type = F_WRLCK;
    fl.l_start = 0;
    fl.l_whence = SEEK_SET;
    fl.l_len = 0;
    return(fcntl(fd, F_SETLK, &fl));
}
```

# 파일 전체에 락을 설정하는 함수 코드

```
#include <unistd.h>
#include <fcntl.h>
int lockfile(int fd)
{
    struct flock fl;
    fl.l_type = F_WRLCK;
    fl.l_start = 0;
    fl.l_whence = SEEK_SET;
    fl.l_len = 0;
    return(fcntl(fd, F_SETLK, &fl));
}
```

# fcntl() 예제 1

**<ssu_fddup.c>**

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>

int main(void)
{
        int testfd;
        int fd;

        fd = open("test.txt", O_CREAT);
        testfd = fcntl(fd, F_DUPFD, 5);
        printf("testfd :%d\n", testfd);
        testfd = fcntl(fd, F_DUPFD, 5);
        printf("testfd :%d\n", testfd);
        getchar();
}
```

실행 결과

```
root@localhost:/home/oslab/lsp# ./ssu_fddup
testfd :5
testfd :6
root@localhost:/home/oslab/lsp#
```

# fcntl() 예제 2

**<ssu_fdcopy.c>**

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>
#include <fcntl.h>

#define MSG "message will be written to Terminal\n"

int main(void)
{
        int new_fd;
        if ((new_fd=fcntl(STDOUT_FILENO, F_DUPFD, 3))== -1){
                fprintf(stderr, "Error : Copying File Descriptor\n");
                exit(1);
        }

        close(STDOUT_FILENO);
        write(3, MSG, strlen(MSG));
        exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab/lsp# ./ssu_fdcopy
message will be written to Terminal
```

# fcntl() 예제 3

**<ssu_closeonexec_3.c>**

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <fcntl.h>

int main(void)
{
	int flag;

	if((flag=fcntl(STDOUT_FILENO, F_DUPFD))== -1){
		fprintf(stderr, "Error : Checking CLOSE_ON_EXEC\n");
		exit(1);
	}

	printf("CLOSE ON EXEC flag is = %d\n", flag);
	exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab/lsp# ./ssu_closeonexec_1
Error : Checking CLOSE_ON_EXEC
```

# fcntl() 예제 4

- P.465

# 비트 연산자를 이용한 플래그 처리

| 비트 연산자 | 설명 |
| --- | --- |
| flag \|= mask | 플래그의 특정 비트를 켬 |
| flag &= ~mask | 플래그의 특정 비트를 끔 |
| flag ^= mask | 플래그의 특정 비트를 토글시킴 |
| flag &mask | 플래그의 특정 비트가 켜져 있는지 검사 |

# fcntl() 예제 6

**<ssu_open_on_exec.c>**

```
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>

int main(void)
{
        int fd;
        int val;

        fd = open("exec_copy.txt", O_CREAT);
        execl("/home/oslab/loop", "./loop", NULL);
        exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./a.out | ps
...
15612 pts/18 00:00:00 a.out
...
root@localhost:/home/oslab# cd /proc/15612/fd
root@localhost:/proc/15612/fd# ls -al
합계 0
dr-x------ 2 root root 0 10월 25 13:59 .
dr-xr-xr-x 3 root root 0 10월 25 13:59 ..
lrwx------ 1 root root 64 10월 25 13:59 0 -> /dev/ttyp0
lrwx------ 1 root root 64 10월 25 13:59 1 -> /dev/ttyp0
lrwx------ 1 root root 64 10월 25 13:59 2 -> /dev/ttyp0
lr-x------ 1 root root 64 10월 25 13:59 3 ->
/home/oslab/exec_copy.txt
```

# <ssu_close_on_exec.c>

**<ssu_close_on_exec.c>**

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>

int main(void)
{
        int fd;
        int val;

                if ((fd = open("exec_copy.txt", O_CREAT)) < 0) {
                fprintf(stderr, "open error for %s\n", "exec_copy.txt");
                exit(1);
                }
                val = fcntl(fd, F_GETFD, 0);
```

```
        if (val & FD_CLOEXEC)
                printf("close-on-exec bit on\n");
        else
                printf("close-on-exec bit off\n");
        val |= FD_CLOEXEC;

        if (val & FD_CLOEXEC)
                printf("close-on-exec bit on\n");
        else
                printf("close-on-exec bit off\n");

        fcntl(fd, F_SETFD, val);
        execl("/home/oslab/loop", "./loop", NULL);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_close_on_exec
close-on-exec bit off
close-on-exec bit on
```

# fcntl() 예제 7 8 9

- P. 473. / 477 / 478

# fcntl() 예제 10

**<ssu_fcntl_lock2.c>**

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>

int main(int argc, char *argv[])
{
        struct flock lock;
        int fd;
        char command[100];

        if((fd = open(argv[1], O_RDWR)) == -1) {
                perror(argv[1]);
                exit(1);
        }

        lock.l_type = F_WRLCK;
        lock.l_whence = 0;
        lock.l_start = 0l;
        lock.l_len = 0l;
        if(fcntl(fd, F_SETLK, &lock) == -1) {
                if (errno == EACCES) {
                        printf("%s busy -- try later\n", argv[1]);
                        exit(2);
                }
                perror(argv[1]);
                exit(3);
        }

        sprintf(command, "vim %s\n", argv[1]);
        system(command);
        lock.l_type = F_UNLCK;
        fcntl(fd, F_SETLK, &lock);
        close(fd);

        return 0;
}
```

실행 결과

```
root@localhost:/home/oslab/lsp# ./ssu_fcntl_lock2
[vi 실행됨]
```

# fcntl() 예제 11 / 12

- P.481. 483

# fcntl() 예제 12

- P.483