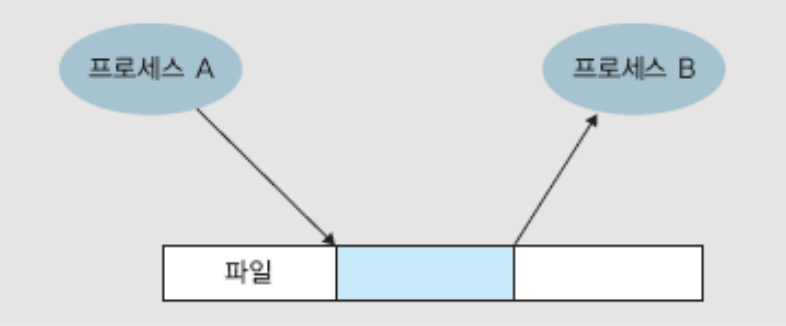
6장 파일 및 레코드 잠금

6.1 파일 및 레코드 잠금

파일 및 레코드 잠금의 원리

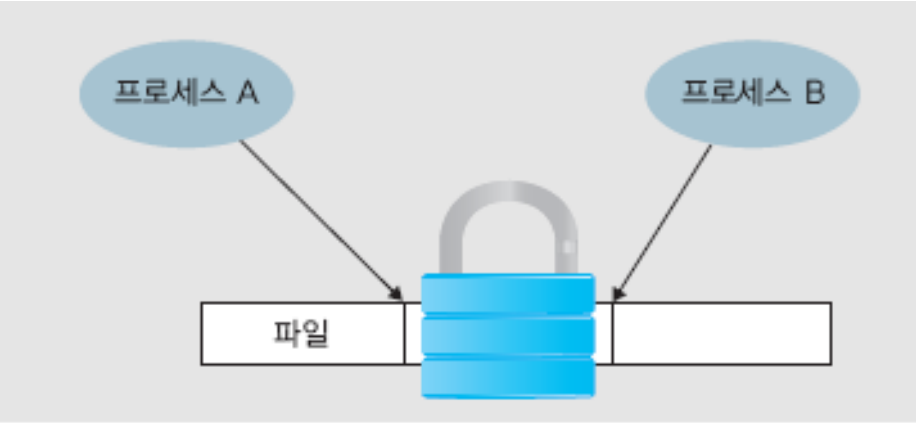
- 어떻게 프로세스 사이에 데이터를 주고받을 수 있을까?
  - 한 프로세스가 파일에 쓴 내용을 다른 프로세스가 읽음



- 문제점
  - 한 프로세스가 파일 내용을 수정하는 동안에 다른 프로세스가 그 파일을 읽는 경우
  - 두 개의 프로세스가 하나의 파일에 동시에 접근하여 데이터를 쓰는 경우

잠금(lock)

- 파일 혹은 레코드(파일의 일부 영역) 잠금
  - 한 프로세스가 그 영역을 읽거나 수정할 때 다른 프로세스의 접근을 제한
  - 잠금된 영역에 한 번에 하나의 프로세스만 접근



  - 특히 레코드에 쓰기(혹은 수정)를 할 경우 대상 레코드에 대해 잠금을 해서 다른 프로세스가 접근하지 못하게 해야 한다.

잠금이 필요한 예

- 잠금 없음

(1) 프로세스 A가 잔액을 읽는다:
잔액 100만원

(2) 프로세스 B가 잔액을 읽는다:
잔액 100만원

(3) 프로세스 B가 잔액에 입금액 20만원을 더하여 레코드를 수정한다: 잔액 120만원

(4) 프로세스 A가 잔액에 입금액 10만원 을 더하여 레코드를 수정한다: 잔액 110만원

- 잠금 사용

(1) 프로세스 A가 레코드에 잠금을 하고 잔액을 읽는다:
잔액 100만원

(2) 프로세스 A가 잔액에 입금액을 더하여 레코드를 수정하고 잠금을 푼다:
잔액 110만원

(3) 프로세스 B가 레코드에 잠금을 하고 잔액을 읽는다:
잔액 110만원

(4) 프로세스 B가 잔액에 입금액을 더하여 레코드를 수정하고 잠금을 푼다:
잔액 130만원

잠금 구현

- fcntl( ) 함수
  - 파일 및 레코드 잠금을 구현할 수 있다.

- 잠금의 종류
  - F_RDLCK : 여러 프로세스가 공유 가능한 읽기 잠금
  - F_WRLCK : 한 프로세스만 가질 수 있는 배타적인 쓰기 잠금

대상 영역의 현재 잠금 상태	읽기 잠금 요청	쓰기 잠금 요청
잠금 없음	승인	승인
하나 이상의 읽기 잠금	승인	거절
하나의 쓰기 잠금	거절	거절

잠금 함수: fcntl()

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

cmd에 따라 잠금 검사 혹은 잠금 설정을 한다. 성공하면 0 실패하면 -1을 리턴

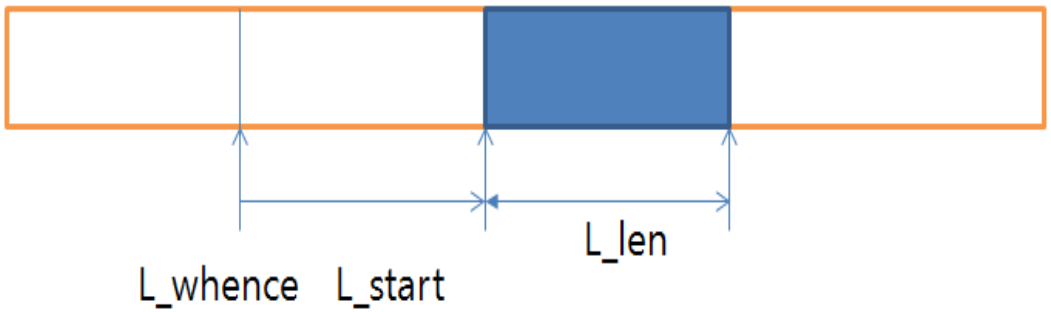
- fd는 대상이 되는 파일 디스크립터
- cmd
  - F_GETLK : 잠금 검사
  - F_SETLK : 잠금 설정 혹은 해제
  - F_SETLKW: 잠금 설정(블로킹 버전) 혹은 해제
- flock 구조체
  - 잠금 종류, 프로세스 ID, 잠금 위치 등

cmd 인자 설명

- cmd
  - F_GETLK : 잠금 검사
    - 잠금이 없다면 flock.l_type를 F_UNLCK로 설정한다.
    - 만약 잠금이 있다면, 현재의 flock 정보를 flock 구조체로 돌려준다.
  - F_SETLK : 잠금 설정 혹은 해제
    - 잠금을 얻을수 없으면 -1을 반환
  - F_SETLKW: 잠금 설정(블로킹 버전) 혹은 해제
    - 잠금이 풀릴때까지 해당영역에서 기다린다.

flock 구조체

```
struct flock {
    short l_type;       // 잠금 종류: F_RDLCK, F_WRLCK, F_UNLCK
    off_t l_start;      // 잠금 시작 위치: 바이트 오프셋
    short l_whence;     // 기준 위치: SEEK_SET, SEEK_CUR, SEEK_END
    off_t l_len;        // 잠금 길이: 바이트 수 (0이면 파일끝까지)
    pid_t l_pid;        // 프로세스 번호. 검사에만 필요
};
```



6.2 잠금 예제 및 잠금 함수

Lock 예제

https://beej.us/guide/bgipc/html/multi/flocking.html

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
int main(int argc, char *argv[]) {
                    /* l_type    l_whence  l_start  l_len    l_pid */
   struct flock fl = {F_WRLCK,  SEEK_SET, 0,       0,       0 };

   int fd;

   fl.l_pid = getpid();

   if (argc > 1)
         fl.l_type = F_RDLCK;

   if ((fd = open("lockdemo.c", O_RDWR)) == -1) {
         perror("open");
         exit(1);
   }
   printf("Press <RETURN> to try to get lock: ");
   getchar();

   printf("Trying to get lock...");
   if (fcntl(fd, F_SETLKW, &fl) == -1) {
       perror("fcntl");
       exit(1);
   }
```

Lock 예제

```
    printf("got lock\n");

    printf("Press <RETURN> to release lock: ");
    getchar();

    fl.l_type = F_UNLCK; /* set to unlock same region */

    if (fcntl(fd, F_SETLK, &fl) == -1) {
         perror("fcntl");
         exit(1);
    }

    printf("Unlocked.\n");

    close(fd);

    return 0;
}
```

고려사항:
- wait or not?
- read lock or write lock?
- lock error 처리

F_SETLK 활용 예

```
#include <error.h>
#define MAX 5

while(fcntl(fd, F_SETLK, &lock) == -1) {
        if(errno == EAGAIN || errno == EACCES) {
            if(try++ <MAX) {     //이미 잠겨있는 경우
                sleep(1);
                continue;
             }
             printf("%s 잠금 다시 시도₩n");
             exit(2);
        }
        perror(argv[2]);    //다른 이유로 잠금 실패
        exit(3);
}
```

간편한 잠금 함수

```
#include <unistd.h>
int lockf(int fd, int cmd, off_t len);
```

cmd에 따라 잠금 설정, 잠금 검사 혹은 잠금 해제 한다.
잠금 영역은 현재 파일 위치부터 len 길이 만큼이다. (0이면 파일끝까지)
성공하면 0 실패하면 –1을 반환한다.
- 보다 쉬운 방법으로 잠금 설정(배타적 잠금만 지원)

- cmd
  - F_LOCK : 지정된 영역에 대해 잠금을 설정한다.
    이미 잠금이 설정되어 있으면 잠금이 해제될 때까지 기다린다.
  - F_TLOCK : 지정된 영역에 대해 잠금을 설정한다.
    이미 잠금이 설정되어 있으면 기다리지 않고 오류(-1)를 반환한다.
  - F_TEST : 지정된 영역이 잠금이 되어 있는지 검사한다. 잠금이 설정되어 있지 않으면 0을 반환하고 잠금이 설정되어 있으면 –1을 반환한다.
  - F_ULOCK : 지정된 영역의 잠금을 해제한다.

실습

- 1. 앞의 Lock 예제를 간편한 잠근 함수로 재작성하라
- 2. 지난 학생 레코드 관리 예제에 대해서 lock을 추가하는 코드를 삽입하라.

6.3 권고 잠금과 강제 잠금

권고 잠금과 강제 잠금

- 권고 잠금(advisory locking)
  - 지금까지 살펴본 잠금: 잠금을 할 수 있지만 강제되지는 않음.
  - 즉 이미 잠금된 파일의 영역에 대해서도 잠금 규칙을 무시하고 읽거나 쓰는 것이 가능
  - 모든 관련 프로세스들이 자발적으로 잠금 규칙을 준수해야 한다.

- 강제 잠금(mandatory locking)
  - 이미 잠금된 파일 영역에 대해 잠금 규칙을 무시하고 읽거나 쓰는 것이 불가능
  - 커널이 잠금 규칙을 강제하므로 시스템의 부하가 증가
  - 파일시스템을 마운트할 때 –o mand 옵션을 설정해야만 가능

강제 잠금

우선 mount할 때 “-o mand” Option 사용

- 강제 잠금을 하는 방법
  - 해당 파일에 대해 set-group-ID 비트를 설정하고
  - group-execute 비트를 끄면 된다

```
$ chmod 2644 mandatory.txt
$ ls -l mandatory.txt
-rw-r-Sr-- 1 chang faculty 160 1월 31일 11:48 stdb1
```

file_lock.c

```
#include <stdio.h>
#include <fcntl.h>
int main(int argc, char **argv) {
   static struct flock lock;
   int fd, ret, c;
   if (argc < 2) {
      fprintf(stderr, "사용법: %s 파일\n",
    argv[0]);
      exit(1);
   }
   fd = open(argv[1], O_WRONLY);
   if (fd == -1) {
      printf("파일 열기 실패 \n");
      exit(1);
   }
   lock.l_type = F_WRLCK;
   lock.l_start = 0;
   lock.l_whence = SEEK_SET;
   lock.l_len = 0;
   lock.l_pid = getpid();
   ret = fcntl(fd, F_SETLKW, &lock);
   if(ret == 0) {  // 파일 잠금 성공하면
      pause();
   }
}
```

강제잠금 실행예

Mount the partition and create a file with the mandatory locking enabled:

```
$ mkdir dir

$ su -
$ mount -t tmpfs -o mand,size=1m tmpfs ./dir
$ exit

$ echo hello > dir/lockfile
$ chmod g+s,g-x dir/lockfile
```

Acquire a lock in the first terminal:

```
$ ./fcntl_lock dir/lockfile
(wait for a while)
^C
```

Try to read the file in the second terminal:

```
$ cat dir/lockfile
(hangs until ^C is pressed in the first terminal)
hello
```