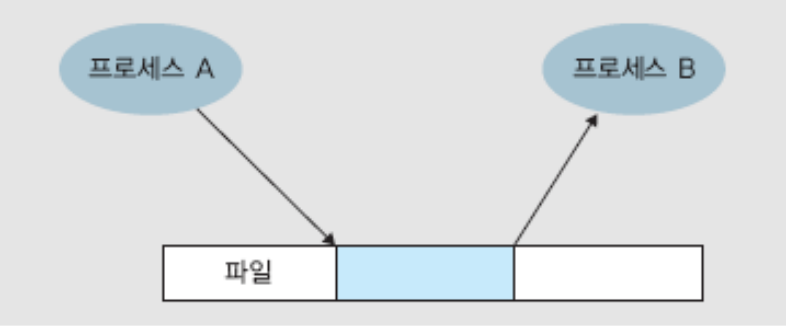
7장 파일 및 레코드 잠금

7.1 파일 잠금

파일 및 레코드 잠금의 원리

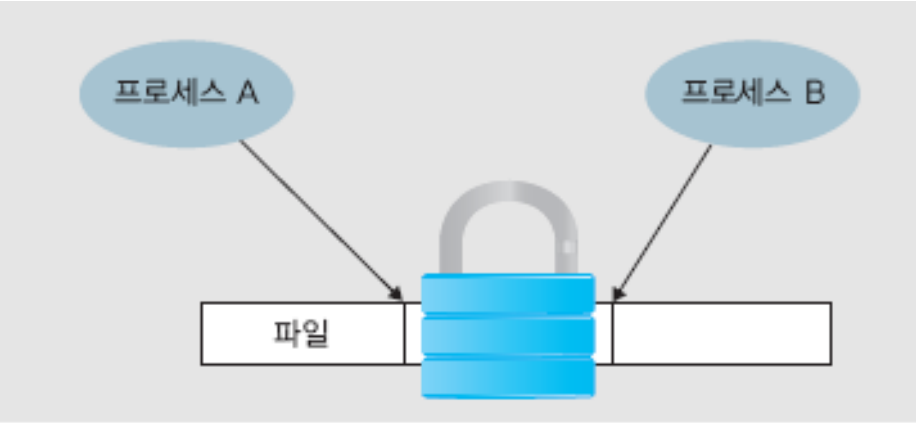
- 어떻게 프로세스 사이에 데이터를 주고받을 수 있을까?
  - 한 프로세스가 파일에 쓴 내용을 다른 프로세스가 읽음



- 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(20만원)을 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

파일 잠금 flock()

- 파일 전체에 잠금
  - LOCK_SH : 공유 잠금(Shared Lock)
  - LOCK_EX : 배타 잠금(Exclusive Lock)

파일의 현재 잠금 상태	공유 잠금 요청	배타 잠금 요청
잠금 없음	승인	승인
하나 이상의 공유 잠금	승인	거절
하나의 배타 잠금	거절	거절

파일 잠금 flock()

```
#include <sys/file.h>
int flock(int fd, int operation)
```

이 함수는 열려진 파일에 대해서 잠금을 적용하거나 해제하는 일을 수행한다.

잠금 명령어	설명
LOCK_SH	공유 잠금으로 한개 이상의 프로세스들이 파일에 대한 공유 잠금이 가능하다.
LOCK_EX	배타 잠금으로 한번에 하나의 프로세스만이 파일에 대한 배타 잠금을 할 수 있다. 파일에 이미 배타 잠금이 걸려 있으면 기다린다.
LOCK_UN	파일에 걸려 있는 잠금을 해제한다.
LOCK_NB	파일에 잠금이 걸려 있으면 기다리지 않고 반환하며 이런 경우에 errno에 EWOULBLOCK가 설정된다. 잠금을 확인하고 다른 일을 할 때 유용하다.

파일 잠금 예제

```
 8  /* 파일 잠금 예제 프로그램 */
 9  int main(int argc, char **argv )
10 {
11      int fd;
12
13      fd = open(argv[1], O_WRONLY | O_CREAT, 0600);
14      if (flock(fd, LOCK_EX) != 0) {
15          printf("flock error₩n");
16          exit(0);
17      }
18
19      for (int i = 0; i < 5; i++) {
20          printf("file lock %d : %d₩n", getpid(), i);
21          sleep(1);
22      }
23
24      if (flock(fd, LOCK_UN) != 0) {
25          printf("unlock error₩n");
26      }
27      close(fd);
28 }
```

파일 잠금 예제

```
$ flock lock.f & flock lock.f &
[1] 96313
[2] 96314
file lock 96313 : 0
file lock 96313 : 1
file lock 96313 : 2
file lock 96313 : 3
file lock 96313 : 4
file lock 96314 : 0
file lock 96314 : 1
file lock 96314 : 2
file lock 96314 : 3
file lock 96314 : 4
[1]- 완료  flock 1 lock.f
[2]+ 완료  flock 2 lock.f
```

7.2 레코드 잠금

레코드 잠금

- 파일 잠금 vs 레코드 잠금
  - 파일 잠금은 파일 전체에 대해 잠금을 하고
  - 레코드 잠금은 파일의 일부 영역(레코드)에 대해서 잠금을 한다.
- fcntl( ) 함수
  - 파일 및 레코드 잠금을 할 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

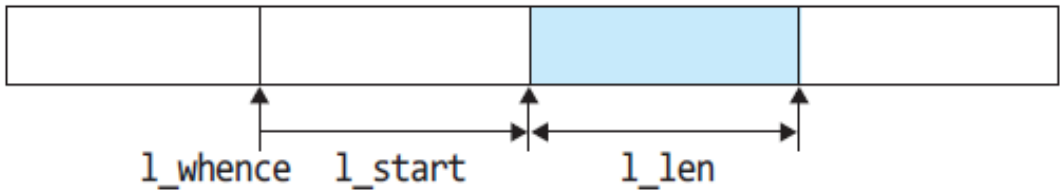
잠금 함수: fcntl()

```
#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 반환

- 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

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
};
```



잠금 예제

- rdlock.c
  - 학생 레코드를 질의하는 프로그램
- wrlock.c
  - 학생 레코드를 수정하는 프로그램

- 기본 원리
  - 수정 프로그램에서 어떤 레코드를 수정하는 중에는 질의 프로그램에서 그 레코드를 읽을 수 없도록 레코드 잠금을 한다.

rdlock.c

- 검색할 학번을 입력받는다.
- 해당 레코드를 읽기 전에 그 레코드에 대해 읽기 잠금 (F_RDLCK)을 한다.

```
fcntl(fd,F_SETLKW, &lock);
```

- 해당 레코드 영역에 대해 잠금을 한 후에 레코드를 읽는다.

rdlock.c

```
 1 #include <stdio.h>
 ...
 7
 8  /* 잠금을 이용한 학생 데이터베이스 질의 프로그램 */
 9  int main(int argc, char *argv[])
10 {
11     int fd, id;
12     struct student record;
13     struct flock lock;
14
15     if (argc < 2) {
16        fprintf(stderr, "사용법 : %s 파일₩n", argv[0]);
17        exit(1);
18     }
```

rdlock.c

```
20   if ((fd = open(argv[1], O_RDONLY)) == -1) {
21       perror(argv[1]);
22       exit(2);
23   }
24
25   printf("₩n검색할 학생의 학번 입력:");
26   while (scanf("%d", &id) == 1) {
27       lock.l_type = F_RDLCK;
28       lock.l_whence = SEEK_SET;
29       lock.l_start = (id-START_ID)*sizeof(record);
30       lock.l_len = sizeof(record);
31       if (fcntl(fd,F_SETLKW, &lock) == -1) { /* 읽기 잠금 */
32           perror(argv[1]);
33           exit(3);
34       }
35
```

rdlock.c

```
36      lseek(fd, (id-START_ID)*sizeof(record), SEEK_SET);
37      if ((read(fd, (char *) &record, sizeof(record)) > 0) && (record.id != 0))
38        printf("이름:%s₩t 학번:%d₩t 점수:%d₩n", record.name, record.id,
39                record.score);
40      else printf("레코드 %d 없음₩n", id);
41
42      lock.l_type = F_UNLCK;
43      fcntl(fd,F_SETLK, &lock); /* 잠금 해제 */
44      printf("₩n검색할 학생의 학번 입력:");
45    }
46
47    close(fd);
48    exit(0);
49 }
```

wrlock.c

- 수정할 학번을 입력받는다.
- 기본 원리
  - 해당 레코드를 읽기 전에 그 레코드에 쓰기 잠금(F_WRLCK) 한다.

```
fcntl(fd,F_SETLKW, &lock);
```

  - 해당 레코드 영역에 대해 잠금을 한 후에 레코드를 수정한다.
  - 이 레코드를 수정한 후에 이 레코드에 대한 잠금을 해제한다.

wrlock.c

```
 1 #include <stdio.h>
 ...
 7
 8 /* 잠금을 이용한 학생 데이터베이스 수정 프로그램 */
 9 int main(int argc, char *argv[])
10 {
11     int fd, id;
12     struct student record;
13     struct flock lock;
14
15     if (argc < 2) {
16         fprintf(stderr, "사용법 : %s 파일 ₩n", argv[0]);
17         exit(1);
18     }
19
```

wrlock.c

```
20   if ((fd = open(argv[1], O_RDWR)) == -1) {
21       perror(argv[1]);
22       exit(2);
23   }
24
25   printf("₩n수정할 학생의 학번 입력:");
26   while (scanf("%d", &id) == 1) {
27       lock.l_type = F_WRLCK;
28       lock.l_whence = SEEK_SET;
29       lock.l_start = (id-START_ID)*sizeof(record);
30       lock.l_len = sizeof(record);
31       if (fcntl(fd,F_SETLKW, &lock) == -1) { /* 쓰기 잠금 */
32           perror(argv[1]);
33           exit(3);
34       }
35
```

wrlock.c

```
36      lseek(fd, (long) (id-START_ID)*sizeof(record), SEEK_SET);
37      if ((read(fd, (char *) &record, sizeof(record)) > 0) && (record.id != 0))
38          printf("이름:%s₩t 학번:%d₩t 점수:%d₩n",
39                      record.name, record.id, record.score);
40      else printf("레코드 %d 없음₩n", id);
41
42      printf("새로운 점수: ");
43      scanf("%d", &record.score);
44      lseek(fd, (long) -sizeof(record), SEEK_CUR);
45      write(fd, (char *) &record, sizeof(record));
46
47      lock.l_type = F_UNLCK;
48      fcntl(fd, F_SETLK, &lock);     /* 잠금 해제 */
49      printf("₩n수정할 학생의 학번 입력:");
50  }
51
52  close(fd);
53  exit(0);
54 }
```

7.3 잠금 함수

간단한 잠금 함수

```
#include <unistd.h>
int lockf(int fd, int cmd, off_t len);
cmd : 쓰기 잠금 설정, 검사 혹은 해제
잠금 영역: 현재 파일 위치부터 len 길이 만큼
성공하면 0 실패하면 –1을 반환한다.
```

- F_LOCK : 지정된 영역에 대해 쓰기 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 잠금되어 있는지 검사한다. 잠금이 설정되어 있지 않으면 0을 반환하고 잠금이 설정되어 있으면 –1을 반환한다.
- F_ULOCK : 지정된 영역의 잠금을 해제한다.

wrlockf.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <unistd.h>
4   #include <fcntl.h>
5   #include "student.h"
6
8   /* 잠금 함수를 이용한 학생 데이터베이스 수정 프로그램 */
9   int main(int argc, char *argv[])
10 {
11      int fd, id;
12      struct student record;
13
14      if (argc < 2) {
15          fprintf(stderr, "사용법 : %s file₩n", argv[0]);
16          exit(1);
17      }
```

wrlockf.c

```
19      if ((fd = open(argv[1], O_RDWR)) == -1) {
20              perror(argv[1]);
21              exit(2);
22      }
23
24      printf("₩n수정할 학생의 학번 입력:");
25      while (scanf("%d", &id) == 1) {
26              lseek(fd, (long) (id-START_ID)*sizeof(record), SEEK_SET);
27              if (lockf(fd, F_LOCK, sizeof(record)) == -1) { // 쓰기 잠금
28                      perror(argv[1]);
29                      exit(3);
30              }
31
32              if ((read(fd, (char *) &record, sizeof(record)) > 0) &&  (record.id != 0))
33              printf("이름:%s₩t 학번:%d₩t 점수:%d₩n",
34                              record.name, record.id, record.score);
35              else printf("레코드 %d 없음₩n", id);
```

wrlockf.c

```
36
37          printf("새로운 점수: ");
38          scanf("%d", &record.score);
39          lseek(fd, (long) -sizeof(record), SEEK_CUR);
40          write(fd, (char *) &record, sizeof(record));
41
42          lseek(fd, (long) (id-START_ID)*sizeof(record), SEEK_SET);
43          lockf(fd, F_ULOCK, sizeof(record)); // 잠금 해제
44          printf("₩n수정할 학생의 학번 입력:");
45      }
46
47      close(fd);
48      exit(0);
49 }
```

7.4 권고 잠금과 강제 잠금

권고 잠금과 강제 잠금

- 권고 잠금(advisory locking)
  - 이미 잠금된 파일 영역에 대해서도 잠금 규칙을 무시하고 읽거나 쓸 수 있음.
  - 잠금을 할 수 있지만 강제되지는 않음.
  - 모든 관련 프로세스들이 자발적으로 잠금 규칙을 준수해야 한다.
  - BSD, Mac OS, Linux 운영체제

- 강제 잠금(mandatory locking)
  - 잠금된 파일 영역에 대해 잠금 규칙을 무시하고 읽거나 쓸 수 없음
  - 커널이 잠금 규칙을 강제하므로 시스템의 부하가 증가
  - System V, 솔라리스 계열 운영체제에서 제공됨.
  - Linux의 경우 파일 시스템을 마운트할 때 “-o mand” 옵션을 사용해서 마운트해야 제공됨

강제 잠금

- 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 mandatory.txt
```

- 강제 잠금 규칙

대상 영역의 현재 잠금 상태	넌블로킹 파일 디스크립터		블로킹 파일 디스크립터	
	읽기	쓰기	읽기	쓰기
읽기 잠금	OK	오류(EAGAIN)	OK	블로킹
쓰기 잠금	오류(EAGAIN)	오류(EAGAIN)	블로킹	블로킹

권고 잠금과 강제 잠금 : file_lock.c

```
#include <stdio.h>
#include <stdlib.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
      c = getchar();
   }
}
```

file_lock 실행 예

- 강제 잠금

```
$ file_lock mandatory.txt
_
```

- 강제 잠금

```
$ ls >> mandatory.txt
_
```

- 권고 잠금

```
$ file_lock advisory.txt
_
```

- 권고 잠금

```
$ ls >> advisory.txt
$
```

핵심 개념

- 한 레코드 혹은 파일에 대한
  - 읽기 잠금은 여러 프로세스가 공유할 수 있지만
  - 쓰기 잠금은 공유할 수 없으며 한 프로세스만 가질 수 있다.

- fcntl() 시스템 호출을 이용하여 지정된 영역에 대해 잠금 검사, 잠금 설정 혹은 잠금 해제를 할 수 있다.

- lockf() 함수를 이용하여 지정된 영역에 대해 잠금 검사, 잠금 설정 혹은 잠금 해제를 할 수 있다.

- 권고 잠금은 이미 잠금된 파일 영역에 대해서도 잠금 규칙을 무시하고 읽거나 쓰는 것이 가능한 반면에 강제 잠금은 잠금 규칙을 무시하고 읽거나 쓰는 것이 불가능하다.