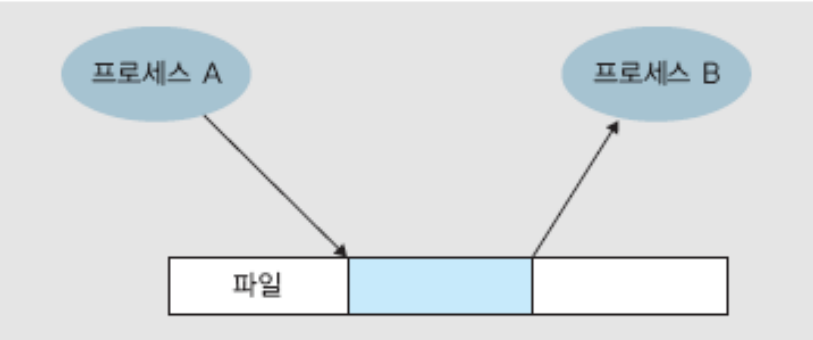
Linux Programming
6장. 파일 및 레코드 잠금

sisong@ut.ac.kr
한국교통대학교 컴퓨터공학전공
송석일

7.1 파일 잠금

파일 및 레코드 잠금의 원리

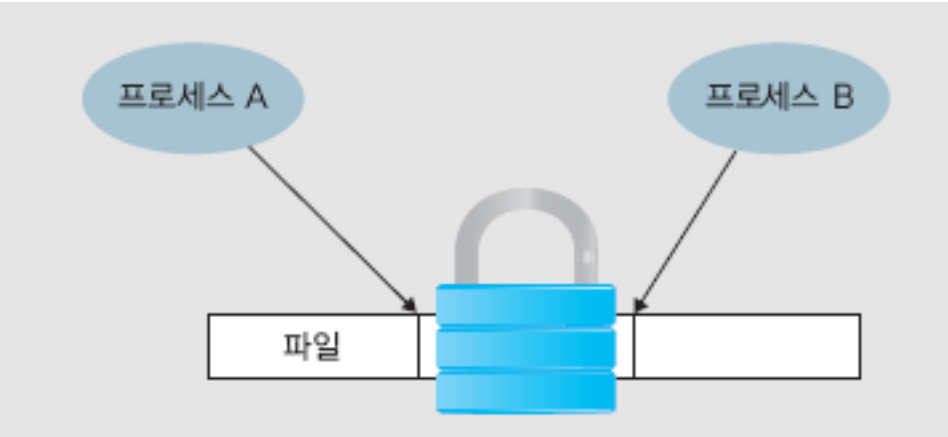
- 파일을 이용한 프로세스간 데이터 공유



- 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)

- 잠금 (Lock) : 파일, 레코드(파일의 일부 영역)
  - 한 프로세스가 파일 또는 레코드를 읽거나 수정할 때 다른 프로세스 접근 제한
  - 잠금된 파일 또는 레코드는 한 번에 하나의 프로세스만 접근



  - 읽기-읽기 는 문제 없음
  - 쓰기-쓰기, 쓰기-읽기 는 잠금 필요

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)
```

- 오픈된 파일에 대해서 잠금을 적용하거나 해제

operation	설명
LOCK_SH	공유 잠금으로 한개 이상의 프로세스들이 파일에 대한 공유 잠금이 가능
LOCK_EX	배타 잠금으로 한번에 하나의 프로세스만이 파일에 대한 배타 잠금 가능 파일에 이미 배타 잠금이 걸려 있으면 기다림
LOCK_UN	파일에 걸려 있는 잠금을 해제
LOCK_NB	파일에 잠금이 걸려 있으면 기다리지 않고 반환하며 이런 경우에 errno에 EWOULBLOCK가 설정 잠금을 확인하고 다른 일을 할 때 유용

```
1.    /* 파일 잠금 예제 프로그램 */
2.    int main(int argc, char **argv )
3.    {
4.       int fd;
5.       fd = open(argv[1], O_WRONLY | O_CREAT, 0600);
6.       if (flock(fd, LOCK_EX) != 0) {
7.          printf("flock error\n");
8.          exit(0);
9.       }
10.      for (int i = 0; i < 5; i++) {
11.         printf("file lock %d : %d\n", getpid(), i);
12.         sleep(1);
13.      }
14.      if (flock(fd, LOCK_UN) != 0) {
15.         printf("unlock error\n");
16.      }
17.      close(fd);
18.   }
```

```
$ 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
```

파일 잠금 예제

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <fcntl.h>
4.  int main(int argc, char **argv) {
5.      static struct flock lock;
6.      int fd, ret, c;
7.      if (argc < 2) {
8.          fprintf(stderr, "사용법: %s 파일\n", argv[0]);
9.          exit(1);
10.     }
11.     fd = open(argv[1], O_WRONLY);
12.     if(fd == -1) {
13.         printf("파일 열기 실패 \n");
14.         exit(1);
15.     }
```

```
1.      lock.l_type = F_WRLCK;
2.      lock.l_start = 0;
3.      lock.l_whence = SEEK_SET;
4.      lock.l_len = 0;
5.      lock.l_pid = getpid();
6.      ret = fcntl(fd, F_SETLKW, &lock);
7.      if(ret == 0) {  // 파일 잠금 성공하면
8.          c = getchar();
9.      }
10. }
```

7.2 레코드 잠금

레코드 잠금

- 파일 잠금 vs 레코드 잠금
  - 파일 잠금은 파일 전체 잠금
  - 레코드 잠금은 파일의 일부 영역(레코드 : 기준위치, 크기)에 대해 잠금
- fcntl( ) 함수
  - 파일 및 레코드 잠금
  - 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()

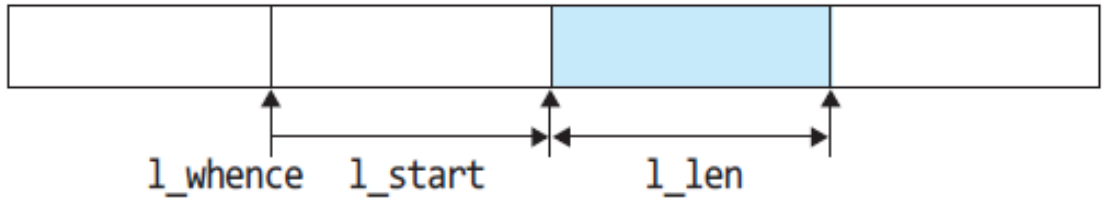
```
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
int fcntl(int fd, int cmd, struct flock *lock);
```

- cmd에 따라 잠금 검사 혹은 잠금 설정
- 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

- 기본 동작과정
  - 검색할 학번을 입력
  - 해당 레코드를 읽기 전에 그 레코드에 대해 읽기 잠금(F_RDLCK)
  - 해당 레코드 영역에 대해 잠금을 한 후에 레코드 읽기

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <unistd.h>
4.  #include <fcntl.h>
5.  #include "student.h"

6.  int main(int argc, char *argv[])
7.  {
8.     int fd, id;
9.     struct student record;
10.    struct flock lock;
11.    if (argc < 2) {
12.       fprintf(stderr,  "사용법 : %s file\n", argv[0]);
13.       exit(1);
14.    }
15.    if ((fd = open(argv[1], O_RDONLY)) == -1) {
16.       perror(argv[1]);
17.       exit(2);
18.    }
```

```
20.    printf("\n검색할 학생의 학번 입력:");

21.    while (scanf("%d", &id) == 1) {
22.       lock.l_type = F_RDLCK;
23.       lock.l_whence = SEEK_SET;
24.       lock.l_start = (id-START_ID)*sizeof(record);
25.       lock.l_len = sizeof(record);
26.       if (fcntl(fd,F_SETLKW, &lock) == -1) {
27.          perror(argv[1]);
28.          exit(3);
29.       }

30.       lseek(fd, (id-START_ID)*sizeof(record), SEEK_SET);
31.       if ((read(fd, (char *) &record, sizeof(record)) > 0)
32.             &&  (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);

36.       lock.l_type = F_UNLCK;
37.       fcntl(fd,F_SETLK, &lock);   /* 잠금 해제 */
38.       printf("\n검색할 학생의 학번 입력:");
39.    }

40.    close(fd);
41.    exit(0);
42. }
```

wrlock.c

- 기본 동작 과정
  - 수정할 학번을 입력
  - 해당 레코드를 읽기 전에 그 레코드에 쓰기 잠금(F_WRLCK)
  - 해당 레코드 영역에 대해 잠금을 한 후에 레코드를 수정
  - 이 레코드를 수정한 후에 이 레코드에 대한 잠금을 해제

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <unistd.h>
4.  #include <fcntl.h>
5.  #include "student.h"

6.  int main(int argc, char *argv[])
7.  {
8.     int fd, id;
9.     struct student record;
10.    struct flock lock;
11.    if (argc < 2) {
12.       fprintf(stderr,  "사용법 : %s file\n", argv[0]);
13.       exit(1);
14.    }
15.    if ((fd = open(argv[1], O_RDWR)) == -1) {
16.       perror(argv[1]);
17.       exit(2);
18.    }
19.    printf("\n수정할 학생의 학번 입력:");
20.    while (scanf("%d", &id) == 1) {
21.       lock.l_type = F_WRLCK;
22.       lock.l_whence = SEEK_SET;
23.       lock.l_start = (id-START_ID)*sizeof(record);
24.       lock.l_len = sizeof(record);
25.       if (fcntl(fd,F_SETLKW, &lock) == -1)
26.          perror(argv[1]);
27.          exit(3);
28.       }
29.       lseek(fd,  (long) (id-START_ID)*sizeof(record),
30.             SEEK_SET);
31.       if ((read(fd, (char *) &record, sizeof(record)) > 0)
32.          && (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);

36.       printf("새로운 점수: ");
37.       scanf("%d", &record.score);
38.       lseek(fd, (long) -sizeof(record), SEEK_CUR);
39.       write(fd, (char *) &record, sizeof(record));

40.       lock.l_type = F_UNLCK;
41.       fcntl(fd, F_SETLK, &lock);  /* 잠금 해제 */
42.       printf("\n수정할 학생의 학번 입력:");
43.    }

44.    close(fd);
45.    exit(0);
46. }
```

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 반환

- F_LOCK : 지정된 영역에 대해 쓰기 잠금을 설정. 이미 잠금이 설정되어 있으면 잠금이 해제될 때까지 블록킹
- F_TLOCK : 지정된 영역에 대해 잠금을 설정. 이미 잠금이 설정되어 있으면 기다리지 않고 오류(-1)를 반환
- F_TEST : 지정된 영역 잠금 검사. 잠금이 설정되어 있지 않으면 0을 반환하고 잠금이 설정되어 있으면 –1을 반환
- F_ULOCK : 지정된 영역의 잠금을 해제

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <unistd.h>
4.  #include <fcntl.h>
5.  #include "student.h"

6.  int main(int argc, char *argv[])
7.  {
8.     int fd, id;
9.     struct student record;
10.    if (argc < 2) {
11.       fprintf(stderr,  "사용법 : %s file\n", argv[0]);
12.       exit(1);
13.    }
14.    if ((fd = open(argv[1], O_RDWR)) == -1) {
15.       perror(argv[1]);
16.       exit(2);
17.    }
18.    printf("\n수정할 학생의 학번 입력:");
19.    while (scanf("%d", &id) == 1) {
20.       lseek(fd,  (long) (id-START_ID)*sizeof(record),
21.             SEEK_SET);
22.       if (lockf(fd, F_LOCK, sizeof(record)) == -1) {
23.          perror(argv[1]);
24.          exit(3);
25.       }
```

```
26.       if ((read(fd, (char *) &record, sizeof(record)) > 0)
27.             && (record.id != 0))
28.          printf("이름:%s\t 학번:%d\t 점수:%d\n",
29.                record.name, record.id, record.score);
30.       else printf("레코드 %d 없음\n", id);

31.       printf("새로운 점수: ");
32.       scanf("%d", &record.score);
33.       lseek(fd, (long) -sizeof(record), SEEK_CUR);
34.       write(fd, (char *) &record, sizeof(record));

35.       lseek(fd,  (long) (id-START_ID)*sizeof(record),
36.             SEEK_SET);
37.       lockf(fd, F_ULOCK, sizeof(record));
38.       printf("\n수정할 학생의 학번 입력:");
39.    }
40.
          close(fd);
41.    exit(0);
42. }
43.
```

7.4 권고 잠금과 강제 잠금

권고 잠금과 강제 잠금

- **권고 잠금(advisory locking) : 지금까지 배운 잠금**
  - 이미 잠금된 파일 영역에 대해서도 잠금 규칙을 무시하고 읽기/쓰기 가능
  - 잠금을 할 수 있지만 강제되지는 않음
  - 모든 관련 프로세스들이 자발적으로 잠금 규칙을 준수 해야 함
  - BSD, Mac OS, Linux 운영체제

- 강제 잠금(mandatory locking)
  - 잠금된 파일 영역에 대해 잠금 규칙을 무시하고 읽거나 쓸 수 없음
  - 커널이 잠금 규칙을 강제해야 하므로 시스템의 부하 증가
  - System V, 솔라리스 계열 운영체제
  - Linux는 파일 시스템 마운트 시 "-o mand" 옵션으로 지정 가능