# 고급 프로세스간 통신 #1

# Record Locking

- 여러 프로세스가 동시에 한 파일에 접근하는 경우 경주 상황(race condition)이 발생할 수 있음
- 한가지 해결 방안 : Record locking
  - 파일의 일부분을 잠그는 행위
- fcntl을 이용

```
#include <fcntl.h>
Int fcntl(int filedes, int cmd, struct flock* ldata);
```

  - cmd
    - F_GETLK
      - ldata를 통해 전달된 데이터에 기반한 록 정보를 얻는다
    - F_SETLK
      - 파일에 록을 적용하고, 불가능하면 return
    - F_SETLKW
      - 파일에 록을 적용하고, 다른 프로세스가 소유 중이면 sleep
      - 나중에 시그널에 의해 인터럽트 될 수 있음
  - struct flock의 주요 member
    - l_type : 록 유형 { F_RDLCK | F_WRLCK | F_UNLCK }
    - l_whence : lseek와 마찬기지로 변위 유형 { SEEK_CUR | … }
    - l_start
    - l_len
    - l_pid : F_GETLK의 경우에만 유효하며, 기존에 록을 지정한 프로세스 id

# flock Sample Code #1

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

void main()
{
    int fd;
    struct flock my_lock;

    my_lock.l_type = F_WRLCK;
    my_lock.l_whence = SEEK_SET;
    my_lock.l_start = 0;
    my_lock.l_len = 10;

    fd = open("locktest", O_RDWR);

    if (fcntl(fd, F_SETLKW, &my_lock) == -1) {
        perror("parent : locking");
        exit(1);
    }
    printf("parent : locked record\n");

switch (fork()) {
        case -1:
            perror("fork");
            exit(2);
        case 0 :
            my_lock.l_len = 5;
            if (fcntl(fd, F_SETLKW, &my_lock) == -1) {
                perror ("child : locking");
                exit(3);
            }
            printf("child : locked\n");
            printf("child : exiting\n");
            exit(0);
    }

    sleep(5);
    printf("parent : exiting\n");
}
```

# flock Sample Code #2

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

void main()
{
	int fd;
	struct flock my_lock;

	my_lock.l_type = F_WRLCK;
	my_lock.l_whence = SEEK_SET;
	my_lock.l_start = 0;
	my_lock.l_len = 10;

	fd = open("locktest", O_RDWR);

	if (fcntl(fd, F_SETLKW, &my_lock) == -1) {
		perror("parent : locking");
		exit(1);
	}
	printf("parent : locked record\n");

switch (fork()) {
		case -1:
			perror("fork");
			exit(2);
		case 0 :
			my_lock.l_len = 5;
			if (fcntl(fd, F_SETLKW, &my_lock) == -1) {
				perror ("child : locking");
				exit(3);
			}
			printf("child : locked\n");
			printf("child : exiting\n");
			exit(0);
	}

	sleep(5);
	my_lock.l_type = F_UNLCK;
	if (fcntl(fd, F_SETLK, &my_lock) == -1) {
		perror("parent : unlocking");
		exit(4);
	}
	printf("parent : exiting\n");
	exit(0);
}
```

# Record Locking 고찰

- 2개의 프로세스 사이에 하나의 파일에 대한 접근은 배타적으로 이루어질 수 있음
- 그러나 2개의 프로세스가 2개의 파일에 대한 록이 필요한 경우, 각각 하나의 파일에 대한 록을 가지고 다른 파일에 대한 록을 기다리는 교착상태가 일어날 수 있음
- 따라서 이에 대한 대비(timeout 매커니즘 등)가 있어야 함
- 다음 예제 실행 예

  A: lock succeeded (proc 7070)
  parent : sleeping
  B: lock succeeded (proc 7071)
  D: Deadlock situation detected/avoided
  C: lock succeeded (proc 7071)
  child : exiting

- 운영체제에서 “deadlock detected/avoided” 절차로 종료

# flock Sample Code #3

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

void main()
{
      int fd;
      struct flock first_lock;
      struct flock second_lock;

      first_lock.l_type = F_WRLCK;
      first_lock.l_whence = SEEK_SET;
      first_lock.l_start = 0;
      first_lock.l_len = 10;


      second_lock.l_type = F_WRLCK;
      second_lock.l_whence = SEEK_SET;
      second_lock.l_start = 10;
      second_lock.l_len = 5;

      fd = open("locktest", O_RDWR);

      if (fcntl(fd, F_SETLKW, &first_lock) == -1) {
            perror("A");
            exit(1);
      }

      printf("A: lock succeeded (proc %d)\n", getpid());

 switch (fork()) {
            case -1:
                  perror("error on fork");
                  exit(2);
            case 0 :
                  if (fcntl(fd, F_SETLKW, &second_lock) == -1) {
                        perror("B");
                        exit(3);
                  }

                  printf("B: lock succeeded (proc %d)\n", getpid());

                  if (fcntl(fd, F_SETLKW, &first_lock) == -1) {
                        perror("C");
                        exit(4);
                  }

                  printf("C: lock succeeded (proc %d)\n", getpid());

                  printf("child : exiting\n");
                  exit(0);
            default :

                  printf("parent : sleeping\n");
                  sleep(10);

                  if (fcntl(fd, F_SETLKW, &second_lock) == -1) {
                        perror("D");
                        exit(5);
                  }

                  printf("D: lock succeeded (proc %d)\n", getpid());
      }

      printf("parent : exiting\n");
      exit(0);
}
```

# 고급 IPC 설비

- Unix에서 제공하는 고급 IPC 설비
  - 메시지 전달
  - 세마포어
  - 공유 메모리
- IPC 설비 키
  - UNIX 상의 IPC 설비들을 구별하기 위한 값
  - key_t 타입
  - 파일 경로에 연관된 정보에 기반한 키 값을 돌려주는 함수

```
#include <sys/ipc.h>
key_t ftok(const char *path, int id);
```

- IPC get 연산
  - mqid = msgget((key_t) 0100,  0644, |IPC_CREAT | IPC_EXEC);
  - semget
  - shmget
- IPC 제어 연산
  - msgctl, semctl, shmctl
- IPC 설비 상태 구조
  - 주요 구성원
    - uid_t cuid;
    - gid_t cgid;
    - uid_t uid;
    - gid_t gid;
    - mode_t umode;

# 메시지 전달(1)

- msgget
  #include <sys/msg.h>
  int msgget(key_t key, int permflags);
  - permflags
    - IPC_CREAT
      - key에 해당하는 메시지 큐가 존재하지 않으면 생성, 이 플래그가 없는 경우 이미 메시지 큐가 있는 경우에만 메시지 큐 식별자를 반환
    - IPC_EXCL
      - IPC_CREAT와 동시에 쓰는 경우, 이미 key에 해당하는 메시지 큐가 존재하는 경우 -1, errno는 EEXIST 값이 저장됨
- msgsnd
  #include <sys/msg.h>
  int msgsnd(int mqid, const void *message, size_t size, int flags);
  - mqid가 가리키는 큐에 메시지 추가
  - message 형태는 다음을 따라야 함

```
struct mymsg {
        long mtype;             /* long int라는 사실에 주목할 것 ! */
        char mtext[SOMEVALUE];
```

  - flags
    - 0 : 메시지를 보낼 때 필요한 자원이 없는 경우(큐의 전체 길이가 시스템 최대값 초과) 수면
    - IPC_NOWAIT : 메시지를 전달할 수 없는 경우 바로 return (복귀값 -1, errno EAGAIN)

# 메시지 전달(2)

- msgrcv

  #include <sys/msg.h>

  int msgrcv(int mqid, void *message, size_t size, long msg_type, int flags);
  - mqid가 가리키는 큐에서 메시지를 읽고 큐에서 제거
  - message : 받아들일 메시지를 저장할 장소
  - size : 저장할 수 있는 최대 길이
  - msg_type : 어떤 메시지를 받아들일지 결정
    - 0 : 가장 일찍 들어온 메시지
    - 양수값 : msg_type이 해당 값을 갖는 첫번째 메시지
    - 음수값 : msg_type의 절대 값보다 작거나 같은 것 중 최소값을 갖는 첫 번째 메시지 (숫자가 작을수록 높은 우선 순위)
  - flags
    - IPC_NOWAIT가 없는 경우 : 메시지를 받을 수 없는 경우 수면 상태
    - IPC_NOWAIT : 메시지를 받을 수 없는 경우 바로 return (복귀값 -1, errno EAGAIN)
    - MSG_NOERROR : 메시지 내용이 size보다 큰 경우 초과분을 잘라내고 정상 처리
    - MSG_NOERROR가 없는데 길이가 size보다 큰 경우 : msgrcv 실패

# mqueue Sample Code #1(1)

```
/* mq.h */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>
#include <errno.h>

#define QKEY    (key_t) 0100
#define QPERM   0660

#define MAX_NAME_LENGTH 50
#define MAX_PRIORITY    10

typedef struct {
        long mtype;
        char mtext[MAX_NAME_LENGTH + 1];
} q_entry;
```

# mqueue Sample Code #1(2)

```
#include "mq.h"

int enter_q (char *objname, int priority)
{
      int len, s_qid;
      q_entry s_entry;

      if ((len = strlen(objname)) > MAX_NAME_LENGTH) {
            fprintf(stderr, "name too long\n");
            return (-1);
      }

      if ((priority > MAX_PRIORITY) || (priority < 0)) {
            fprintf(stderr, "invalid priority level\n");
            return(-2);
      }

      if ((s_qid = msgget(QKEY, IPC_CREAT | QPERM)) == -1) {
            perror("msgget failed");
            return (-3);
      }

      s_entry.mtype = (long) priority;
      strncpy(s_entry.mtext, objname, MAX_NAME_LENGTH);

      if (msgsnd(s_qid, (void *)&s_entry, (size_t) len, 0) == -1) {
            perror("msgsnd failed");
            return (-4);
      }
      else
            return(0);
}
```

```
void main(int argc, char **argv)
{
      int priority;

      if (argc != 3) {
            fprintf(stderr, "usage : %s objname priority\n", argv[0]);
            exit(1);
      }

      if (((priority = atoi(argv[2])) <= 0) || (priority > MAX_PRIORITY))
{
            fprintf(stderr, "invalid priority\n");
            exit(2);
      }


      if (enter_q(argv[1], priority) < 0) {
            fprintf(stderr, "enter failure\n");
            exit(3);
      }

      exit(0);
}
```

# mqueue Sample Code #1(3)

```
#include "mq.h"

void main()
{
        int mlen, r_qid;
        q_entry r_entry;

        if ((r_qid = msgget(QKEY, IPC_CREAT | QPERM)) == -1) {
                perror("msgget failed");
                exit(-1);
        }

        for (;;) {
                if ((mlen = msgrcv(r_qid, &r_entry, sizeof(q_entry), (-1 * MAX_PRIORITY), MSG_NOERROR)) == -1) {
                        perror("msgrcv failed");
                        exit(-2);
                }
                else {
                        r_entry.mtext[mlen] = '₩0';
                        printf("₩nPriority : %ld ₩tName : %s₩n", r_entry.mtype, r_entry.mtext);
                }
        }
}
```

# mqueue Sample Code 실행 예

```
cara% client test1 7
cara% client test2 2
cara% client test3 6
cara% client test4 1
cara% client test5 8
cara% client test6 4
cara% server

Priority : 1     Name : test4

Priority : 2     Name : test2

Priority : 4     Name : test6

Priority : 6     Name : test3

Priority : 7     Name : test1

Priority : 8     Name : test5
^Ccara%
```

# 메시지 제어

- msgctl
  - 3가지 목적
    - 메시지 큐의 상태정보 얻기
    - 메시지 큐에 관련된 제한 변경
    - 시스템 상에서 큐를 통째로 제거

```
#include <sys/msg.h>
int msgctl(int mqid, int command, struct msqid_ds *msq_stat);
```

    - mqid : 유효한 메시지 큐 식별자
    - msqid_ds 주요 구성원
      - struct ipc_perm msg_perm; /* 소유권 – 허가 */
      - msgqnum_t msg_qnum; /* 큐 상의 메시지 수 */
      - msglen_t msg_qbytes; /* 큐 상의 최대 바이트 수 */
      - pid_t msg_lspid; /* 마지막 msgsnd를 수행한 프로세스 번호 */
      - pid_t msg_lrpid; /* 마지막 msgrcv를 수행한 프로세스 번호 */
      - time_t msg_stime; /* 마지막 msgsnd 시간 */
      - time_t msg_rtime; /* 마지막 msgrcv 시간 */
      - time_t msg_ctime; /* 마지막 변경 시간 */
    - command
      - IPC_STAT : 메시지 큐 상태정보를 msq_stat에 넣도록 지시
      - IPC_SET : 메시지 큐 상태 정보 변경
        - msq_stat.msg_perm.uid
        - msq_stat.msg_perm.gid
        - msq_stat.msg_perm.mode
        - msq_stat.msg_qbytes
      - IPC_RM : 시스템에서 이 메시지 큐 삭제
        - 이때 msq_stat은 NULL로 설정

# mqueue Sample Code #2

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <time.h>

void main(int argc, char* argv[])
{
        key_t    mkey;
        int      msq_id;
        struct msqid_ds msq_status;

        if (argc != 2) {
                fprintf(stderr, "usage : %s keyval₩n", argv[0]);
                exit(1);
        }

        mkey = (key_t) atoi(argv[1]);

        if ((msq_id = msgget(mkey, 0)) == -1) {
                perror("msgget failed");
                exit(2);
        }

        if (msgctl(msq_id, IPC_STAT, &msq_status) == -1) {
                perror("msgctl failed");
                exit(3);
        }

        printf("₩nKey : %d ₩tmsg_qid : %d", mkey, msq_id);
        printf("₩n%d messages on queue", msq_status.msg_qnum);
        printf("₩nLast send by %d at %s", msq_status.msg_lspid, ctime(&msq_status.msg_stime));
        printf("₩nLast recv by %d at %s", msq_status.msg_lrpid, ctime(&msq_status.msg_rtime));
}
```

# ipcs와 ipcrm 명령

- IPC 설비를 이용하기 위한 2가지 명령
  - ipcs : IPC 설비의 현재 상태 출력

```
cara% ipcs
IPC status from <running system> as of 2011년 5월 24일 화요일 오후 05시 26분 07초
T         ID      KEY        MODE        OWNER    GROUP
Message Queues:
q          0   0x40       --rw-rw----      kgu      prof
Shared Memory:
Semaphores:
cara%
```

  - ipcrm : 시스템으로부터 IPC 설비 제거 (소유자 또는 super user만 이용 가능)

```
cara% ipcrm -q 0
cara% ipcs
IPC status from <running system> as of 2011년 5월 24일 화요일 오후 05시 28분 04초
T         ID      KEY        MODE        OWNER    GROUP
Message Queues:
Shared Memory:
Semaphores:
cara%
```