

# UNIX 시스템 프로그래밍

» 10장. 시스템 V의 프로세스간 통신

# IPC 설비

## ▶ key

- message\_queue, semaphore, shared memory segment에 대한 identifier (file 이름에 해당)
- 서로 다른 process들도 동일 IPC 객체는 같은 key 값으로 접근 !!!
- 시스템에서 unique한 key값을 사용하여야 함.

## ▶ key 값 생성

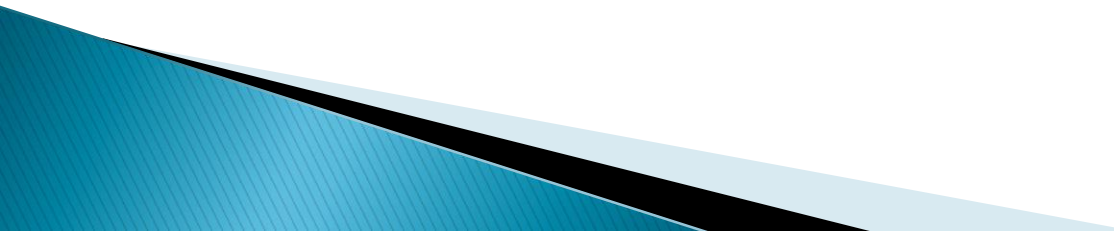
```
#include<sys/ipc.h>
```

```
key_t ftok(const char *path, int id);
```

- 해당 파일의 st\_dev, st\_ino와 id로 key값 생성

# IPC 객체 상태 구조

```
▶ struct ipc_perm{  
    uid_t cuid : 생성자의 uid;  
    gid_t cgid : 생성자의 gid;  
    uid_t uid : 소유자 uid;  
    gid_t gid : 소유자 gid;  
    mode_t mode : permission (execution은 의미 없음);  
};
```



# IPC 정보 검색 및 삭제

- ▶ ipc 관련 정보 검색
  - `$ ipcs`
- ▶ ipc 삭제
  - `$ipcrm -m shmid or -q msqid or -s semid`

# message passing

- ▶ message queue를 통한 message 전달
  - msgget : queue 생성
  - msgsnd : message 보내기
  - msgrcv : message 받기

# msgget 시스템 호출

## ▶ 사용법 :

```
#include <sys/msg.h>
```

```
int msgget(key_t key, int permflags);
```

- key : message queue의 key 값
- permflags = queue에 대한 access permission
  - | IPC\_CREAT :
    - 해당 queue가 없으면, 생성한 후 return; 있으면, return
    - 이 flag가 설정 되지 않은 경우에는, queue가 존재하는 경우에만 return
  - | IPC\_EXCL :
    - 해당 queue가 존재하지 않는 경우만 성공, 아니면 -1 return
- return 값 : 음수가 아닌 queue identifier

# msgsnd 시스템 호출

## ▶ 사용법:

```
#include <sys/msg.h>
```

```
int msgsnd(int mqid, const void *message, size_t size,  
int flags);
```

- mqid : message queue identifier
- message 의 주소 : 보낼 message가 저장된 주소
- size : message의 크기
- flags = IPC\_NOWAIT
  - send가 불가능하면 즉시 return (queue가 가득 찬 경우)
  - flag가 설정 되지 않으면 (0이면), 성공 시까지 blocking
- return 값은 0 or -1

# msgsnd 시스템 호출 (2)

## ▶ message의 구조 :

- long type의 정수 값을 갖는 mtype과 임의의 message의 내용으로 구성.
- message의 size는 message 내용의 크기만.
- message 구조체의 예 :

```
struct mymsg{  
    long mtype;           // message type (양의 정수)  
    char mtext[SOMEVALUE]; // message 내용  
};
```



# msgrcv 시스템 호출

## ▶ 사용법:

```
#include <sys/msg.h>
```

```
int msgrcv(int mqid, void *message, size_t size, long  
msg_type, int flags);
```

- mqid : message queue identifier
- message 주소 : 받은 message를 저장할 저장 장소의 주소
- size : 준비 된 저장 장소의 크기
- msg\_type =
  - 0 : queue의 첫 message
  - > 0 : 해당 값을 갖는 첫 message
  - < 0 : mtyep값이 절대값 보다 작거나 같은 것 중 최소값을 갖는 첫 message

# msgrcv 시스템 호출 (2)

- flags = IPC\_NOWAIT
  - receive가 불가능하면 즉시 return (queue에 해당 msg가 없는 경우)
  - return 값은 -1; errno = EAGAIN
  - flag가 설정 되지 않으면 (값이 0이면), 성공 시까지 blocking
- flags = MSGNOERROR
  - message가 size보다 길면 초과분을 자른다.
  - flag가 설정 되지 않으면 size 초과 시 error
- return 값 :
  - receive 성공 시 : 받은 message의 길이
  - 실패 시 : -1
  - access permission 때문에 실패한 경우 errno=EACCESS

# message 송수신의 예

```
struct q_entry{  
    long mtype;  
    int mnum;  
};
```

```
struct q_entry msg;
```

```
qid=msgget(0111, 0600|IPC_CREAT);  
while(msgrcv(qid, &msg, sizeof(int), 1, 0)>0){  
    msg.mtype=2;  
    msg.mnum=msg.mnum+8;  
    msgsnd(qid, &msg, sizeof(int), 0);  
}
```

# msgctl 시스템 호출

## ▶ msgctl 호출

- message queue에 대한 정보 획득
- message queue 제거

## ▶ 사용법

```
#include <sys/msg.h>
```

```
int msgctl(int mqid, int command, struct msqid_ds  
*msq_stat);
```

- mqid : message queue identifier
- command =
  - IPC\_STAT : msg queue의 상태 정보 확인
  - IPC\_RMID : msg\_queue 삭제

# msgctl 시스템 호출 (2)

## ▶ msqid\_ds 구조:

- struct ipc\_perm msg\_perm; // 소유권
- msgqnum\_t msg\_qnum; // msg 수
- msglen\_t msg\_qbytes; // bytes 수
- pid\_t msg\_lspid; // last sender
- pid\_t msg\_lrpid; // last receiver
- time\_t msg\_stime; // last sending time
- time\_t msg\_rtime; // last receipt time
- time\_t msg\_ctime; // last s/r time

# semaphore

- ▶ `p(sem)` or `wait(sem)`

- If ( $\text{sem} > 0$ )

- decrement `sem` by one;

- else{

- wait until `sem` becomes non-zero;  
then decrement;

- }

- ▶ `v(sem)` or `signal(sem)`

- increment `sem` by one;

- if (queue of waiting processes not empty)  
restart first process in wait queue;

# semaphore (2)

## ▶ 사용예:

```
p(sem);  
something interesting;  
v(sem);
```

# semget 시스템 호출

## ▶ 사용법 :

```
#include <sys/sem.h>
```

```
#include <sys/types.h>
```

```
#include <sys/ipc.h>
```

```
int semget(key_t key, int nsems, int permflags);
```

- key : semaphore 집합 이름
- nsems : semaphore 집합 내의 semaphore 수
- permflags : 0600, IPC\_CREAT, IPC\_EXCL, ...
- return 값 : semaphore 집합 identifier



# semget 시스템 호출 (2)

	index 0	index 1	index 2	index 3
semid	semval =2	semval =4	semval =1	semval =3

nsems=4

# semget 시스템 호출 (3)

- ▶ 집합 내 각 semaphore와 연관된 값
  - semval : semaphore 값 (semaphore 값의 초기화 필요)
  - sempid : 최근 semaphore를 access 한 process id
  - semncnt : semaphore 값이 증가하기를 기다리는 process 수
  - semzcnt : semaphore 값이 0이 되기를 기다리는 process 수

# semctl 시스템 호출

## ▶ 사용법 :

```
#include <sys/sem.h>
```

```
int semctl (int semid, int sem_num, int command,  
union semun arg);
```

- semid : semaphore identifier
- sem\_num : 집합 내 특정 semaphore 지정
- command
  - IPC\_STAT : 상태 정보를 arg.stat에 저장
  - IPC\_RMID : semaphore 집합 삭제

# semctl 시스템 호출 (2)

- command = 단일 semaphore에 영향을 미치는 기능
  - GETVAL : semval 값을 return
  - SETVAL : semavl 값을 arg.val 값으로 지정
  - GETPID : sempid 값을 return
  - GETNCNT : semncnt 값을 return
  - GETZCNT : semzcnt 값을 return
- command = semaphore 집합 전체 에 영향을 미치는 기능
  - GETALL : 모든 semval 값을 arg.array에 저장
  - SETALL : arg.array 값으로 모든 semval 값을 지정

# semctl 시스템 호출 (3)

▶ struct semun arg :

```
union semun{  
    int val;  
    struct semid_ds *buf;  
    unsigned short *array;  
};
```

# semaphore 만들고 초기값 설정하기

```
union semun{  
    int val;  
    struct semid_ds *buf;  
    ushort *array;  
};
```

```
union semun arg;
```

```
semid=semget((key_t) 0123, 1, 0600|IPC_CREAT  
|IPC_EXCL);
```

```
arg.val=3;  
semctl(semid, 0, SETVAL, arg);
```

# semaphore 만들고 초기값 설정하기 (2)

```
union semun{  
    int val;  
    struct semid_ds *buf;  
    ushort *array;  
};  
union semun arg;  
ushort buf[3];
```

```
semid=semget((key_t) 0246, 3, 0600|IPC_CREAT|IPC_EXCL);  
for (i=0; i<3; i++)  
    buf[i]=i+1;  
arg.array=buf;  
semctl(semid, 0, SETALL, arg);
```

# semop 시스템 호출

## ▶ 사용법 :

```
#include <sys/sem.h>
```

```
int semop(int semid, struct sembuf *op_array,  
size_t num_ops);
```

- semid : semaphore identifier
- oparray : 수행 할 연산 지정
- num\_ops : op\_array내의 sembuf의 수 (여러 개의 semaphore에 대한 연산을 동시에 지정 할 수 있음.)



# semop 시스템 호출 (2)

- ▶ struct sembuf{  
    unsigned short sem\_num;  
    short sem\_op;  
    short sem\_flg;  
};
  - sem\_num : semaphore index
  - sem\_op : 수행 할 연산 (양의 정수 또는 음의 정수 또는 0)
  - sem\_flg : IPC\_NOWAIT or SEM\_UNDO

# semop 시스템 호출 (3)

- ▶ sem\_op : 수행할 연산
  - 음수 : p() or wait() 연산

```
if (semval >= |sem_op|)
```

```
    set semval to semval - |sem_op|;
```

```
else
```

```
    wait until semval reaches or exceeds |sem_op|;
```

```
    then set semval to semval - |sem_op|;
```

# semop 시스템 호출 (4)

- ▶ `sem_op` : 수행할 연산
  - 양수 : `v()` or `signal()` 연산  
set `semval` to `semval + |sem_op|`;
  - 0 :  
semval 값의 변화는 없음;  
sem\_op는 semval이 0이 될 때까지 waiting;

# semaphore 연산 실행의 예

```
struct sembuf p_buf;
```

```
p_buf.sem_num=0;
```

```
p_buf.sem_op=-1;
```

```
semop(semid, &p_buf, 1);
```

```
printf("process %d in critical section\n", pid);
```

```
sleep(10);
```

```
printf("process %d leaving critical section\n", pid);
```

```
p_buf.sem_num=0;
```

```
p_buf.sem_op=1;
```

```
semop(semid, &p_buf, 1);
```

