

시스템 V IPC 기초[1]

□ 시스템 V IPC 통신기법

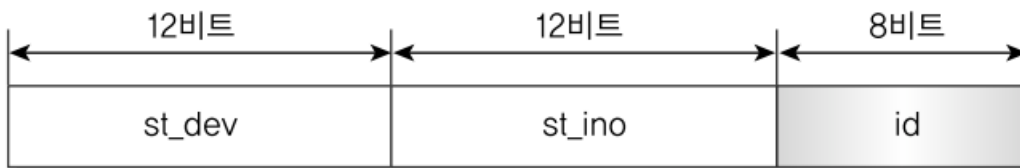
FIFO 나 pipe 도 IPC 임.

같은 운영체제에서

- 시스템 V 계열 유닉스에서 개발해 제공하는 프로세스 간 통신방법 IPC에 소켓을 집어넣는 경우도 있음.
- 메시지 큐, 공유 메모리, 세마포어 유저 프로그램에서도 사용할 수 있는 세마포어 대표적인 것들

□ 공통 요소

- 시스템 V IPC를 사용하기 위해서는 IPC 객체를 생성해야함.
- IPC 객체를 생성하기 위해 공통적으로 사용하는 기본 요소는 키와 식별자



□ 키 생성 키는 절대 같아서는 안됨.

- 키로 IPC PRIVATE 지정
- ftok 함수로 키 생성

[그림 10-1] 키의 구조

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

- 인자로 파일시스템에 이미 존재하는 임의의 파일의 경로명과 1~255사이의 번호 지정
- 키의 구조에서 id(8비트)에 인자로 지정한 번호 저장. 번호에 0은 지정하지 않는다.
- /* 각자 현재 directory로 수정할 것, key값을 줄 경우 뒤3자리 정수를 이용할 것 */

경로가 매우 길어서 최종 파일명에 도달하기 전 잘릴 수 있음.

id

s16010'980' 반드시 예제에서 사용하는 키 값을 수정하여서 사용할 것!

시스템 V IPC 기초[2]

□ IPC 공통 구조체

메시지 큐, 공유 메모리, 세마포어에 공통적으로 들어가는 부분.

- IPC객체를 생성하면 IPC 공통 구조체가 정의된다.

```
struct ipc_perm {  
    uid_t uid;  
    gid_t gid;  
    uid_t cuid;  
    gid_t cgid;  
    mode_t mode;  
    uint_t seq;  
    key_t key;  
    int pad[4];  
};
```

- uid, gid : 구조체 소유자ID 및 소유그룹ID
- cuid, cgid : 구조체를 생성한 사용자ID, 그룹ID 유효
- mode : 구조체에 대한 접근 권한
- seq : 슬롯의 일련번호
- key : 키값
- pad : 향후 사용을 위해 예약된 영역

굳이 바꿔줄 필요는 없음.
구조체 안에 이러한 것이 대충 있구나 알면 됨.



시스템 V IPC 관련 명령[1]

□ 시스템 V IPC 정보 검색: ipcs 명령

- 시스템 V IPC의 정보를 검색하고 현재 상태 확인

```
ipcs [-aAbciJmopqstZ] [-D mtype]
```

- -m : 공유 메모리에 관한 정보만 검색 ipcs -m
- -q : 메시지 큐에 관한 정보만 검색
- -s : 세마포어에 관한 정보만 검색
- -a : -b, -c, -o, -p, -t 옵션으로 검색하는 항목 모두 출력
- -A : 전체 항목을 모드 검색
- -b : 각 방법의 최대값 검색
- -c : IPC 객체를 생성한 사용자의 로그인명과 그룹명 검색
- -D mtype : 메시지 큐에서 mtype으로 지정한 메시지만 검색
- -i : 공유 메모리 세그먼트에 연결된 ISM의 개수 출력
- -J : IPC 객체 생성자의 프로젝트명 출력
- -o : 현재 사용되고 있는 정보 출력
- -p : PID 정보 출력
- -t : 시간 정보 출력

프로그램이 끝난 후에 반드시 내가 생성한 IPC 객체가 있는 지 ipcs 를 통해 확인하여 ipcrm 을 통해 지워주어야 함.
ex) ipcrm -q [msqid]



시스템 V IPC 관련 명령[2]

□ IPCS 명령 사용 예

- 현재 동작중인 IPC 객체가 하나도 없는 경우

```
# ipcs
IPC status from <running system> as of 2009년 2월 18일 수요일 오전 09시 36분 41초
T          ID          KEY          MODE          OWNER          GROUP
Message Queues:
Shared Memory:
Semaphores:
```

- -A 옵션 지정시 : 모든 항목 출력 LINUX OS에서는 안됨. SUN OS에서만 되는 것 같음.

```
# ipcs -A
IPC status from <running system> as of 2009년 2월 18일 수요일 오전 10시 36분 41초
T          ID          KEY          MODE          OWNER          GROUP          CREATOR          CGROUP CBYTES
QNUM QBYTES LSPID LRPID   STIME          RTIME          CTIME          PROJECT
Message Queues:
T          ID          KEY          MODE          OWNER          GROUP          CREATOR          CGROUP NATTCH
SEGSZ CPID  LPID   ATIME   DTIME   CTIME  ISMATTCH          PROJECT
Shared Memory:
T          ID          KEY          MODE          OWNER          GROUP          CREATOR          CGROUP NSEMS
OTIME      CTIME          PROJECT
Semaphores:
```

시스템 V IPC 관련 명령[3]

□ 시스템 V IPC 정보 삭제: ipcrm

```
ipcrm [-m shmid] [-q msqid] [-s semid] [-M shmkey] [-Q msgkey] [-S emkey]
```

- -m shmid : 공유 메모리 삭제
- -q msqid : 메시지 큐 삭제
- -s semid : 세마포어 삭제
- -M shmkey : shmkey로 지정한 공유 메모리 삭제
- -Q msgkey : msgkey로 지정한 공유 메모리 삭제
- -S semkey : semkey로 지정한 공유 메모리 삭제



메시지 큐[1]

□ 메시지 큐

- 파이프와 유사하나 파이프는 스트림 기반으로 동작하고 메시지 큐는 메시지 단위로 동작
- 각 메시지의 최대 크기는 제한되어 있음
- 각 메시지에는 메시지 유형이 있어 수신 프로세스는 어떤 유형의 메시지를 받을 것인지 선택 가능

□ 메시지 큐 생성: msgget(2)

```
#include <sys/msg.h>
int msgget(key_t key, int msgflg);
```

아까 생성했던 key 번호 or 아이디 끝 번호 3자리

- key : IPC_PRIVATE 또는 ftok로 생성한 키값
- msgflg : 플래그와 접근 권한 지정
 - IPC_CREAT : 새로운 키면 식별자를 새로 생성
 - IPC_EXCL : 이미 존재하는 키면 오류 발생
- 메시지 큐 식별자를 리턴(msqid_ds 구조체)

msqid 리턴



메시지 큐[2]

□ msqid_ds 구조체

```
struct msqid_ds { 공통적으로 갖는  
    struct ipc_perm  
    msg_perm;  
    struct msg *msg_first;  
    struct msg *msg_last;  
    msglen_t msg_cbytes;  
    msgqnum_t msg_qnum;  
    msglen_t msg_qbytes;  
    pid_t msg_lspid;  
    pid_t msg_lrpid;  
    time_t msg_stime;  
    int32_t msg_pad1;  
    time_t msg_rtime;  
    int32_t msg_pad2;  
    time_t msg_ctime;  
    int32_t msg_pad3;  
    short msg_cv;  
    short msg_qnum_cv;  
    long msg_pad4[3];  
};
```

- msg_perm: IPC공통 구조체
- msg_first: 첫번째 메시지에 대한 포인터
- msg_last: 마지막 메시지에 대한 포인터
- msg_cbytes: 현재 메시지큐에 있는 총 바이트수
- msg_qnum: 메시지 큐에 있는 메시지 개수
- msg_qbytes: 메시지 큐의 최대 크기
- msg_lspid: 마지막으로 메시지를 보낸 프로세스ID
- msg_lrpid: 마지막으로 메시지를 읽은 프로세스ID
- msg_stime: 마지막으로 메시지를 보낸 시각
- msg_rtime: 마지막으로 메시지를 읽은 시각
- msg_ctime: 마지막으로 메시지 큐의 권한변경시각
- msg_pad1,2,3: 예비공간



메시지 큐[3]

□ 메시지 전송: msgsnd(2)

```
#include <sys/msg.h>
int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

메시지 내용(mtext[1])의 사이즈

- msqid : 메시지 큐 식별자 메시지 id
- msgp : 메시지 버퍼 주소
- msgsz : 메시지 버퍼 크기
- msgflg : 블록모드(0, 메시지 큐가 찬 경우 대기)
비블록 모드(IPC_NOWAIT, 대기 없이 바로 오류를 return)

- 메시지 버퍼 구조체 구조체 이름은 마음대로.

```
struct msqbuf {
    long mtype;
    char mtext[1];
};
```

이 변수의 이름은 무조건 이대로.

- mtype : 메시지 유형으로 양수를 지정 메시지 번호
- mtext : 메시지 내용 저장 메시지 내용



[예제 10-1] 메시지 큐 생성 및 메시지 전송하기(client.c)

```
...
06 struct mymsgbuf {
07     long mtype;
08     char mtext[80];
09 };
10
11 int main(void) {
12     key_t key;
13     int msgid;
14     struct mymsgbuf mesg;
15
16     key = ftok("keyfile", 1);
17     msgid = msgget(key, IPC_CREAT|0644);
18     if (msgid == -1) {
19         perror("msgget");
20         exit(1);
21     }
22 }
```

메시지 버퍼 정의

키 값 생성

메시지 큐 생성

없으면, 생성.
있으면, 기존 것 사용.

실제 예제에서는 학번 마지막 3개의 번호로 지정되어 있음.



[예제 10-1] 메시지 큐 생성 및 메시지 전송하기

```
23  mesg.mtype = 1; 메시지 번호.
24  strcpy(mesg.mtext, "Message Q Test\n"; 보낼 메시지 만들기
25
26  if (msgsnd(msgid, (void *)&mesg, 80, IPC NOWAIT) == -1) {
27      perror("msgsnd");
28      exit(1);
29  }
30
31  return 0;
32  }
```

ex10_1.out

ipcs -qo

IPC status from <running system> as of 2009년 2월 18일 수요일 오후 2시 01분 14초

T	ID	KEY	MODE	OWNER	GROUP	CBYTES	QNUM
---	----	-----	------	-------	-------	--------	------

Message Queues:

q	1	0x100719c	--rw-r--r--	root	other	80	1
---	---	-----------	-------------	------	-------	----	---



메시지 큐[4]

□ 메시지 수신: msgrcv(2)

```
#include <sys/msg.h>
ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long int msgtyp,
int msgflg);
```

자신이 받고 싶은 메시지
번호

- msqid : 메시지 큐 식별자
- msgp : 메시지 버퍼 주소
- msgsz : 메시지 버퍼 크기
- msgtyp : 읽어올 메시지 유형
- msgflg : 블록모드(0)/비블록모드(IPC_NOWAIT)

* MSG_NOERROR : 메시지의 내용이 size보다 길면 초과분을 잘라낸다. 지정되지 않았을 경우, msgrcv가 실패하게 됨.

- msgtyp에 지정할 값
 - 0 : 메시지 큐의 다음 메시지를 읽어온다.
 - 양수 : 메시지 큐에서 msgtyp로 지정한 유형과 같은 메시지를 읽어온다.
 - 음수 : 메시지의 유형이 msgtyp로 지정한 값의 절대값과 같거나 작은 메시지들 중 최소값을 갖는 첫 번째 메시지를 읽어온다 여러 개의 메시지를 순서대로 읽어오려고 할 때 쓰면 좋음.



[예제 10-2] 메시지 수신하기(server.c)

```
...
05 struct mymsgbuf {
06     long mtype;
07     char mtext[80];
08 };
09
10 int main(void) {
11     struct mymsgbuf inmsg;
12     key_t key;
13     int msgid, len;
14
15     key = ftok("keyfile", 1);
16     if ((msgid = msgget(key, 0)) < 0) {
17         perror("msgget");
18         exit(1);
19     }
20
21     len = msgrcv(msgid, &inmsg, 80, 0, 0);
22     printf("Received Msg = %s, Len=%d\n", inmsg.mtext, len);
23 }
```

메시지 버퍼 정의

송신측과 같은 키값 생성

snd 부분과 같아야 함.

실제 예제에서는 학번의 마지막 3개의 숫자.

메시지 수신

ex10_2.out

Received Msg = Message Q Test, Len=80

ipcs -qo

IPC status from <running system> as of 2009년 2월 18일 수요일 오후 2시 03분 48초

T	ID	KEY	MODE	OWNER	GROUP	CBYTES	QNUM
---	----	-----	------	-------	-------	--------	------

Message Queues:

q	1	0x100719c	--rw-r--r--	root	other	0	0
---	---	-----------	-------------	------	-------	---	---

- Client로부터 보내는 여러 개의 메시지를 server에서 mtype의 순번으로 받을 수 있는 프로그램을 작성하라

Client :

```
$/client msg1 3
```

```
$/client msg2 4
```

```
$/client msg3 1
```

```
$/server
```

server 1, 3, 4 순서대로 받게 프로그램을 작성.

msg3 1

msg1 3

msg2 4

msgrcv(msqid, &mymsg, 80, -1*(MAX), 0);
3번 반복.



메시지 큐[5]

□ 메시지 제어: msgctl(2) 메시지 큐의 정보를 획득하는 함수

```
#include <sys/msg.h>
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

- msqid : 메시지 큐 식별자
- cmd : 수행할 제어기능
- buf : 제어 기능에 사용되는 메시지 큐 구조체 주소 buf에 정보를 가져오거나 buf에 있는 내용으로 정보를 수정.
- cmd에 지정할 값
 - IPC_RMID : 메시지 큐 제거
 - IPC_SET : 메시지 큐 정보 중 msg_perm.uid, msg_perm.gid, msg_perm.mode, msg_qbytes 값을 세번째 인자로 지정한 값으로 변경 super user 이거나 소유자이어야 작동 가능.
 - IPC_STAT : 현재 메시지 큐의 정보를 buf에 저장



[예제 10-3] 메시지 큐 삭제하기 (test1.c)

```
...
05 int main(void) {
06     key_t key;
07     int msgid;
08
09     key = ftok("keyfile", 1);
10     msgid = msgget(key, IPC_CREAT|0644);
11     if (msgid == -1) {
12         perror("msgget");
13         exit(1);
14     }
15
16     printf("Before IPC_RMID\n");
17     system("ipcs -q");
18     msgctl(msgid, IPC_RMID, (struct msqid_ds *)NULL);
19     printf("After IPC_RMID\n");
20     system("ipcs -q");
21
22     return 0;
23 }
```

키값 생성

메시지 큐 삭제



[예제 10-3] 실행결과

```
# ex10_3.out
Before IPC_RMID
IPC status from <running system> as of 2009년 2월 18일 수요일 오후 2시 21분 47초
T          ID          KEY          MODE          OWNER          GROUP
Message Queues:
q          1          0x100719c  --rw-r--r--          root          other
After IPC_RMID
IPC status from <running system> as of 2009년 2월 18일 수요일 오후 2시 21분 47초
T          ID          KEY          MODE          OWNER          GROUP
Message Queues:
```

test2.c

