Advanced IPC

한양대학교 소프트웨어학부

Dept. of Division of Computer Science

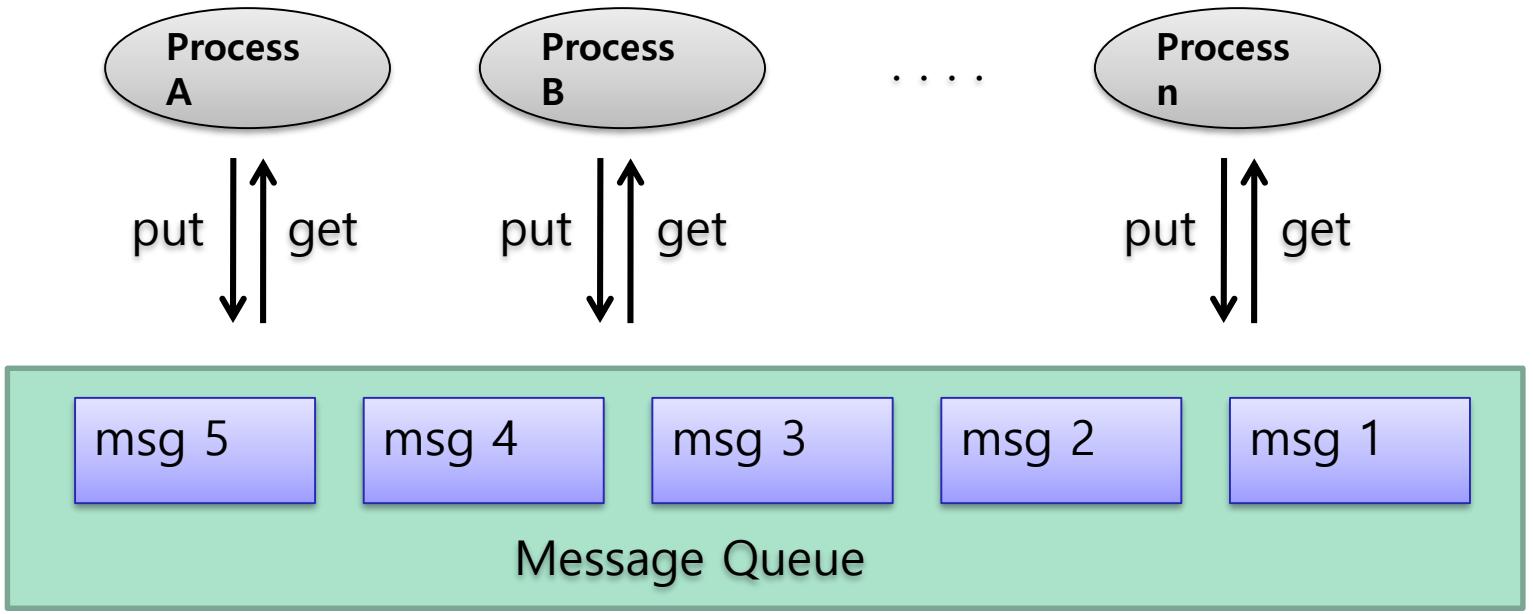
Hanyang University

CSE4009
System Programming

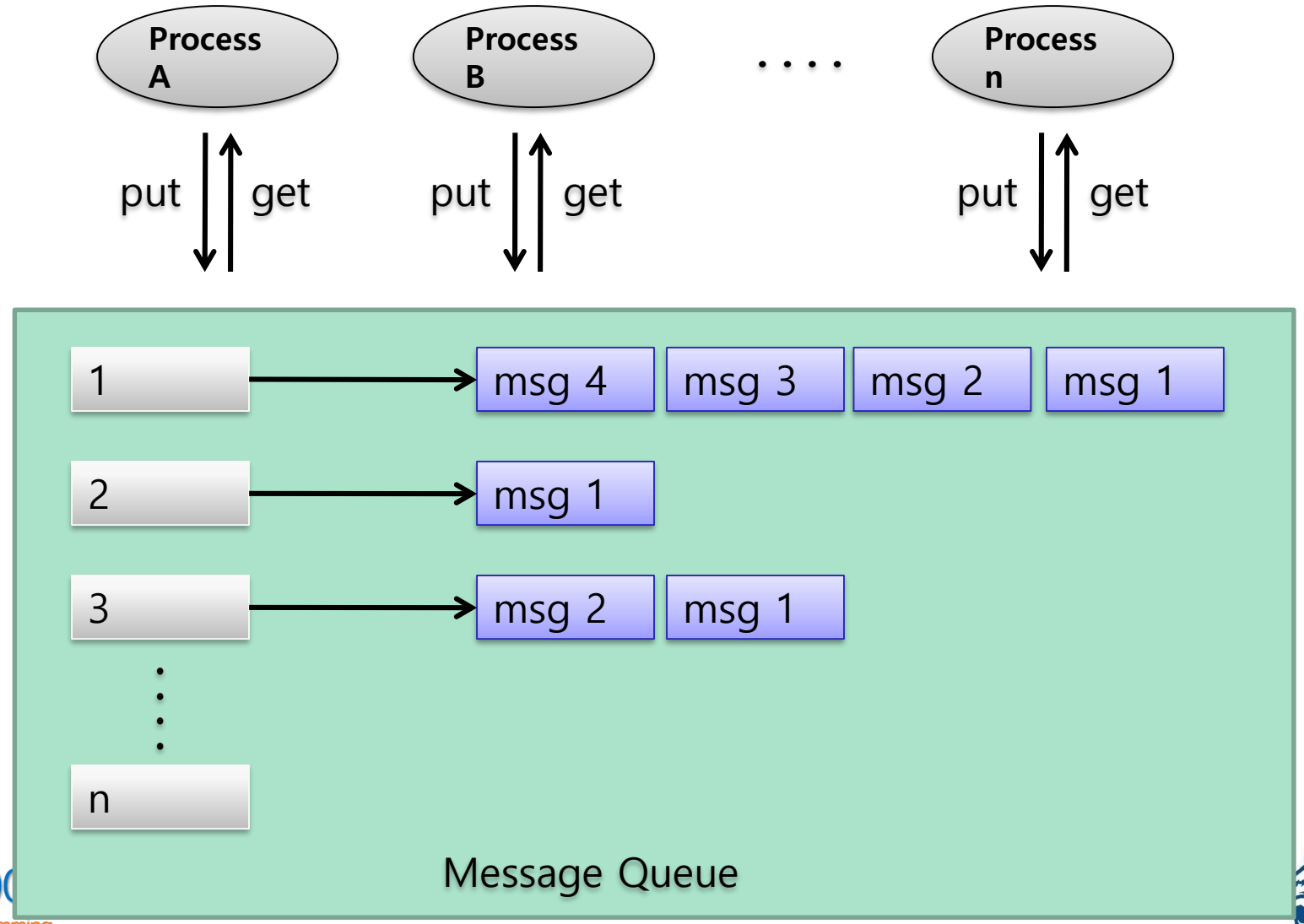
HANYANG UNIVERSITY

Message Queue

- 스트림 채널 외에 "메시지 단위"의 송수신용 큐
- 메시지 전송에 우선순위 부여가 가능
- 메시지큐를 이용한 프로세스간 통신



타입이 있는 메시지 큐


Process A
Process B
. . . .
Process n
put
get
put
get
put
get
1
msg 4
msg 3
msg 2
msg 1
2
msg 1
3
msg 2
msg 1
n
Message Queue

Message Queue – msgget()

사용법

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

key : 메시지큐를 구분하기 위한 고유 키
msgflag : 메시지큐 생성시 옵션을 지정(bitmask 형태)

IPC_CREATE, IPC_EXCL 등의 상수와 파일 접근 권한 지정

msqid_ds 구조체
: 메시지큐가 생성될 때마다 메시지큐에 관한 정보를 담는 메시지큐 객체가 생성

마지막으로 송신 또는 수신한 프로세스 PID, 송수신 시간, 큐의 최대 바이트 수, 메시지큐 소유자 정보 등이 저장

msgflag 옵션

- IPC_CREATE
  - 동일한 key를 사용하는 메시지 큐가 존재하면 그 객체에 대한 ID를 정상적으로 리턴
  - 존재하지 않는다면 메시지큐 객체를 생성하고 ID를 리턴

- IPC_EXCL
  - 동일한 key를 사용하는 메시지 큐가 존재하면 -1을 리턴
  - 단독으로 사용하지 못하고 IPC_CREATE와 같이 사용해야 함

- IPC_PRIVATE
  - key가 없는 메시지큐 생성
  - 명시적으로 key 값을 정의하여 사용할 필요가 없는 경우 이용
  - 메시지큐 ID를 서로 공유할 수 있는 부모와 자식 프로세스 사이에 사용 가능
  - 외부의 다른 프로세스는 이 메시지큐에 접근 불가

Message Queue 객체

```
struct msqid_ds {
        struct ipc_perm msg_perm;      // 메시지큐 접근권한
        struct msg *msg_first;         // 처음 메시지
        struct msg *msg_last;          // 마지막 메시지
        time_t msg_stime;              // 마지막 메시지 송신시각
        time_t msg_rtime;              // 마지막 메시지 수신시각
        time_t msg_ctime;              // 마지막으로 change가 수행된 시각
        struct wait_queue *wwait;
        struct wait_queue *rwait;
        ushort msg_cytes;
        ushort msg_qnum;
        ushort msg_qbytes;             // 메시지큐 최대 바이트수
        ushort msg_lspid;              // 마지막으로 msgsnd를 수행한 PID
        ushort msg_lrpid;              // 마지막으로 받은 PID
};

struct ipc_perm {                      // 해당 객체에 연관된 각종 관리 정보를 수록
        key_t key;
        ushort udi;                    // owner의 euid와egid
        ushort gid;
        ushort cuid;                   // 생성자의 euid와 egid
        ushort cgid;
        ushort mode;                   // 접근모드의 하위 9bits
        ushort seq;                    // 순서번호(sequence number)
};
```

msgsnd()

사용법

```
int msgsnd(int msqid, struct msgbuf *msgp, size_t msgsz, int msgflg);
struct msgbuf {
    long mtype;          // 메시지타입> 0
    chat mtext[1];       // 메시지데이터
}
```

- 메시지를 큐에 넣을 때 사용
- mtype은 1 이상이어야 함
- msgbuf의 첫 4바이트는 반드시 long 타입
- mtext는 문자열, binary 등 임의의 데이터 사용가능
- msgflg는 0으로 한 경우 메시지큐 공간이 부족하면 블록됨
- msgflg를 IPC_NOWAIT로 하면 메시지큐 공간이 부족한 경우 블록되지 않고 EAGAIN 에러코드와 함께 -1을 리턴
- Return value
  [성공시:0]
  [실패시:-1]

msgrcv()

사용법

ssize_t **msgrcv**(int msqid, struct msgbuf *msgp, size_t msgsz, long msgtype, int msgflg);

메시지큐로부터 메시지를 읽는 함수

msqid : 메시지큐 객체ID

msgp : 메시지큐에서 읽은 메시지를 저장하는 수신공간

msgsz : 수신공간의 크기

msgflg : 메시지가 없는 경우 취할 동작
- 0이면 대기
- IPC_NOWAIT이면 EAGAIN 에러코드와 -1을 리턴
- MSG_NOERROR로 설정하면 msgsz 크기만큼만 읽음
  (읽은 메시지가 수신공간 크기보다 크면 E2BIG 에러가 발생)

msgtype
- 0: 타입의 구분없이 메시지큐에 입력된 순서대로 읽음
- 양수: 그 값을 갖는 메시지를 읽음
- 음수: 절대값보다 작거나 같은 것 중에서 최소값부터 순서대로 읽음

msgctl()

사용법

int **msgctl**(int msqid, int cmd, struct msqid_ds *buf);

메시지큐에 관한 정보 읽기, 동작 허가 권한 변경, 메시지큐 삭제 등을 제어

- msqid: 메시지큐 객체ID
- cmd: 제어 명령 구분
  - IPC_STAT :메시지큐 객체에 대한 정보를 buf에 넣도록 시스템에 지시
  - IPC_SET : r/w 권한, euid, egid, msg_qbyte를 변경하는 명령
  - IPC_RMID : 메시지큐를 삭제하는 명령
- buf: cmd 명령에 따라 동작
  msqid_ds 구조의 구조를 저장

IPC_SET
: r/w권한, euid, egid, msg_qbyte만 변경이 가능
: IPC_STAT 명령으로 메시지큐의 객체를 얻은 후 변경시키고 IPC_SET call

IPC_RMID
: 삭제 명령을 내렸을 때 아직 읽지 않은 메시지가 있어도 즉시 삭제

Example - Message Queue (1) - header

```c
/* prio_queue.h */

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

#define MSGQKEY (key_t)0111 /* putty 사용시 각자 다르게 생성 */
#define PERMISSION 0777     /* 메세지큐 접근 권한 */
#define MAXLENGTH 100       /* 최대 메시지 길이 */
#define MAXPRIO 20          /* 최대 우선순위 => 클수록 늦게 */

struct msg_buf {
        long mtype;
        char mtext[MAXLENGTH + 1];
};
```

Example - Message Queue (2) - client

```c
/* client_queue.c */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "prio_queue.h"

int msg_put(int msgq_id, char *request, int prio)
{
        int len;
        struct msg_buf req_msg;

        if ((len = strlen(request)) > MAXLENGTH) {
                perror("[c]request name too long");
                return -1;
        }

        if ((prio > MAXPRIO) || (prio < 0)) {
                perror("[c]wrong priority\n");
                return -1;
        }
        req_msg.mtype = (long)prio;
        strncpy(req_msg.mtext, request, MAXLENGTH);

        if (msgsnd(msgq_id, &req_msg, len, 0) == -1) {
                perror("[c]message send failed");
                return -1;
        } else return 0;
}
```

Example - Message Queue (3) - client

```c
int main(int argc, char* argv[])
{
        int msg_prio, msgq_id, running = 1;
        char request[MAXLENGTH + 1];

        msgq_id = msgget(MSGQKEY, IPC_CREAT|PERMISSION);

        printf("%d\n", msgq_id);

        if (msgq_id == -1) {
                perror("[c]msg queue create failed");
                exit(1);
        }
        while(running) {
                strncpy(request, argv[1], MAXLENGTH);
                printf("[c]argv[1]: %s\n", request);
                msg_prio = atoi(argv[2]); /* 문자열을 숫자로 변환 */
                printf("[c]argv[2]: %d\n", msg_prio);

                if (msg_put(msgq_id, request, msg_prio) < 0) {
                        perror("[c]msg send failed");
                        exit(1);
                }

                if (!strcmp(request, "end")) {   /* 끝내고자 할 때 : end와 20을 입력 */
                        printf("[c]>>>> NULL <<<<\n");
                        strcpy(request, "end");
                        msg_prio = MAXPRIO;

                        if (msg_put(msgq_id, request, msg_prio) < 0) {
                                perror("[c]msg send failed");
                                exit(1);
                        }
                }
                running = 0;
        }
        exit(0);
}
```

Example - Message Queue (4) - server

```c
/* server_queue.c */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "prio_queue.h"

int msg_service(int msgq_id) {
        int len;
        struct msg_buf req_msg;

        if ((len = msgrcv(msgq_id, &req_msg, MAXLENGTH, (-1*MAXPRIO), MSG_NOERROR)) == -1) {
                perror("[s]message rcv failed");
                return -1;
        } else {
                req_msg.mtext[len] = '\0'; /* 문자열 끝에 널 문자 삽입 */
                printf("[s]-----> %s", req_msg.mtext);

                if (strcmp(req_msg.mtext, "end") == 0) { /* 받은 문자열이 'end' 일 경우 */
                        printf("\n[s]!!!!!!!!\n");
                        return 1;
                } else {
                        printf("\n[s]priority: %ld name:%s\n", req_msg.mtype, req_msg.mtext);
                        return 0;
                }
        }
}
```

Example - Message Queue (5) - server

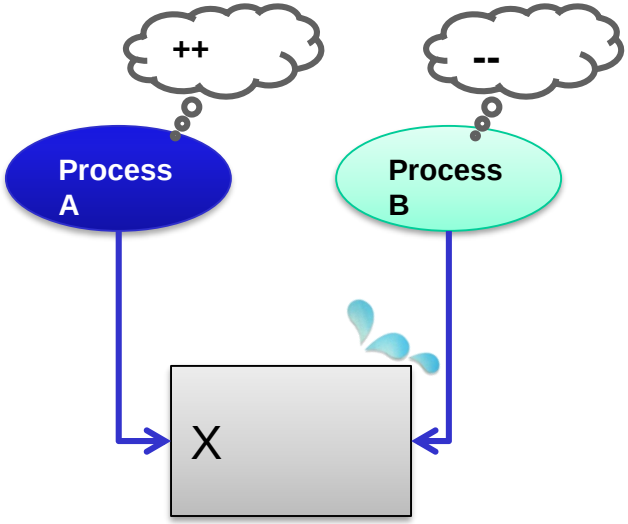
```
29 int main(void) {
30         int msg_prio, msgq_id, re_value, running = 1;
31
32         msgq_id = msgget(MSGQKEY, PERMISSION | IPC_CREAT); /* 메시지큐 생성 */
33
34         if (msgq_id == -1) {
35                 perror("[s]msg queue create failed");
36                 exit(1);
37         }
38
39         while (running) {
40                 re_value = msg_service(msgq_id);
41
42                 if (re_value < 0) {
43                         perror("[s]msg service failed");
44                         exit(1);
45                 } else if (re_value == 1) running = 0;
46         }
47
48         if (msgctl(msgq_id, IPC_RMID, 0) == -1) {
49                 perror("[s]msgq remove failed");
50                 exit(1);
51         }
52 }
```

Semaphore

- 프로세스간 통신에서 발생할 수 있는 동기화 문제를 해결하기 위해 사용

- 동기화 문제 예

```
Process A의 작업:
{
        printf("A: before increase x=%d₩n", x);
        x++;
        printf("A: after increase x=%d₩n", x);
}
Process B의 작업:
{
        printf("B: before decrease x=%d₩n", x);
        x--;
        printf("B: after decrease x=%d₩n", x);
}
```



Semaphore (cont'd)

- 동기화 문제 해결 방법
  공유데이터를 access하는 프로세스 수를 한 순간에 하나로 제한
  Semaphore를 사용하여 제한할 수 있음

- 동작 방법
  공유데이터에 접근하기 전에 **semaphore s의 값을 확인**
    - 1이면 s를 0으로 변경하고 access
    - 0 이면 대기
    - access가 끝나면 s값을 1로 다시 변경

- Semaphore 값은 최대 5를 가질 수 있음
  1, 0 은 binary semaphore

Semaphore – semget

사용법

```
int semget(key_t key, int nsemd, int semflg);
```

정상적으로 호출이 성공하면 semaphore 집합 식별자라는 메시큐 큐 식별자와 유사한 역할을 하는 값이 돌아옴 – [semid]

key : 세마포어를 구분하기 위한 키
struct semid_ds 구조체에 정보를 저장
다른 프로세스는 이 키를 알아야 사용할 수 있음

nsemd : 세마포어 집합을 구성하는 멤버의 수
3으로 설정하면 3개의 세마포어 집합을 얻음

semflg : 세마포어 생성 옵션, 세마포어 객체에 대한 접근 권한

Semaphore – semid structure

메세지 큐와 마찬가지로, 커널은 각각의 semaphore 집합을 위한 특별한 내부 자료 구조를 관리함

```
struct semid_ds {
        struct ipc_perm sem_perm;       // 접근 허가내용, ipc.h
        time_t sem_otime;               // 최근 세마포어 조작시각
        time_t sem_ctime;               // 최근 변경 시각
        struct sem *sem_base;           // 첫 세마포어 포인터
        struct wait_queue *eventn;
        struct wait_queue *eventz;
        struct sem_undo *undo;
        ushort sem_nsems;               // 세마포어 멤버수
};
```

Semaphore – semflg

- IPC_CREAT
  - 같은 key값을 사용하는 세마포어가 존재하면 해당 객체에 대한 ID 리턴
  - 같은 key값의 세마포어가 존재하지 않으면 새로운 세마포어를 생성하고 그 ID를 리턴
- IPC_EXCL
  - 같은 key값을 사용하는 세마포어가 존재하면 semget() 호출은 실패하고 -1을 리턴
  - IPC_CREAT와 같이 사용해야 함
- IPC_PRIVATE
  - key가 없는 세마포어를 생성
    - 명시적으로 key 값을 사용할 필요가 없는 경우에 사용
    - 세마포어 생성후 fork()를 호출하면 자식 프로세스는 세마포어 객체의 ID를 상속받으므로 접근이 가능

Semaphore – semop

사용법

```
int semop(int semid, struct sembuf *opeations, unsigned nsops);
```

기본적인 세마포어 연산을 실제로 수행하는 시스템 호출
–semid : 세마포어 ID
–operations : 몇 번째 멤버 세마포어를 연산할지 구분하는 번호
sembuf 구조체 사용
–nsops : oeprations가 몇 개의 리스트를 가지는지 명시

```
struct sembuf {
        short sem_num;  // 멤버 세마포어 번호(0번이 첫 번째 멤버)
        short sem_op;   // 세마포어 연산내용
        short sem_flg;  // 조작 플래그
};
```

Semaphore – structure

- sem_num
  - 세마포어 집합 중 몇 번째 멤버 세마포어를 연산할지 구분하는 번호
  - 첫 번째 멤버 세마포어는 0
- sem_op
  - 증가 또는 감소 시킬 값
- sem_flg
  - 0, IPC_NOWAIT, SEM_UNDO 등을 bitmask 형태로 취함
    - 0
      - 디폴트인 블록킹 모드로 semop()를 수행
      - 어떤 세마포어 값을 N만큼 감소 시키려 할 때 현재값이 N-1 이하면 세마포어 값이N 이상이 될 때까지 블록
    - IPC_NOWAIT
      - 세마포어 값이 부족해도 블록되지 않고 리턴
      - 리턴 값은 -1이고 에러코드는 EAGAIN
    - SEM_UNDO
      - 프로세스의 종료 시 커널은 해당 세마포어 연산을 취소

Semaphore – semctl

사용법

int **semctl**(int semid, int member_index, int cmd, union semun semarg);

세마포어의 사용 종료, 세마포어 값 읽기 및 설정, 특정 멤버 세마포어를 기다리는 프로세스의 수 확인

- semid : 제어할 대상의 세마포어 ID
- member_index : 세마포어 멤버번호
- cmd : 수행할 동작
- semarg : cmd의 종류에 따라 다르게 사용되는 인자 (semun 타입)

```
union semun { // 설정 또는 값을 구하는 변수
        int val;                    // SETVAL을 위한 값
        struct semid_ds buf;        // IPC_STAT, IPC_SET을 위한 버퍼
        unsigned short int *array;  // GETALL, SETALL을 위한 배열
        struct seminfo *__buf;      // IPC_INFO를 위한 버퍼
        void *__pad;                // dummy
};
```

Semaphore – cmd

- IPC_STAT
  - semid_ds 타입의 세마포어 객체를 얻어오는 명령
  - union semun semarg 인자에 세마포어 객체가 리턴
  - semctl() 함수의 member_index 인자는 사용되지 않음

```
struct semid_ds semobj;
union semun semarg;
semarg.buf = &semobj;
semctl(semid, 0, IPC_STAT, semarg);
```

- SETVAL
  - 공유데이터의 수를 설정해 주는 작업(세마포어 객체 초기화)
  - semarg에 설정할 값을 지정하여 semarg.val 변수에 입력

```
union semun semarg;
unsigned short semvalue = 10;
semarg.val = semvalue;
semctl(semid, 2, SETVAL, semarg);
```

Semaphore – cmd (cont'd)

- SETALL
  - 세마포어 집합내의 모든 세마포어의 값을 초기화하는 명령
  - semarg.array 인자에 초기값을 배열 형태로 지정
  - member_index는 사용하지 않음

```
unsigned short values[] = {3, 14, 1, 25};
union semun semarg;
semarg.array = values;
semctl(semid, 0, SETALL, semarg);
```

- GETVAL
  - 특정 멤버 세마포어의 현재 값을 얻어오는 명령

```
N = semctl(semid, 2, GETVAL, 0);
```

Semaphore – cmd (cont'd)

- GETALL
  - 모든 멤버 세마포어의 현재 값을 읽는데 사용

```
union semun semarg;
unsigned short semvalues[3];
semarg.array = semvalues;
semctl(semid, 0, GETALL, semarg);
for (i=0; i<3; i++)
        printf("%d번 멤버 세마포어의 값= %d₩n", i, semvalue[i]);
```

- GETNCNT
  - 세마포어 값이 현재보다 더 큰 값을 갖기를 기다리는 프로세서의 수를 얻는데 사용

```
semctl(semid, member_index, GETNCNT, 0);
```

Semaphore – cmd (cont'd)

- GETPID
  - 특정 멤버 세마포어에 대한 마지막으로 semop() 함수를 수행한 프로세스의 PID를 얻는데 사용

```
int pid = semctl(semid, member_index, GETPID, 0);
```

- IPC_RMID
  - 세마포어를 삭제하는 명령

```
semctl(semid, 0, IPC_RMID, 0);
```

Example – Semaphore (1)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/ipc.h>
#include <sys/sem.h>

void errquit(char *msg)
{
        perror(msg);
        exit(0);
}

#define PEN 0
#define NOTE 1

/* 세마포어 조작 */
struct sembuf increase[] = { {0, +1, SEM_UNDO}, {1, +1, SEM_UNDO} };
struct sembuf decrease[] = { {0, -1, SEM_UNDO}, {1, -1, SEM_UNDO} };

/* 세마포어 초기값 : 연필 한 자루와 노트 두 권*/
unsigned short seminitval[] = { 1, 2 };

union semnu {
        int val;
        struct semid_ds *buf;
        unsigned short int *array;
        struct seminfo *_buf;
} semarg;

int semid; /* 세마포 ID */
void do_work(); /* 각 프로세스가 수행할 작업 */

```

Example - Semaphore (2)

```
37 int main(int argc, char* argv[])
38 {
39         semid = semget(0x1234, 2, IPC_CREAT | 0600);
40         if (semid == -1)
41                 errquit("semget");
42
43         /* 세마포어 초기화 */
44         semarg.array = seminitval;
45         if (semctl(semid, 0, SETALL, semarg) == -1)
46                 errquit("semctl");
47
48         /* 표준출력 non-buffering */
49         setvbuf(stdout, NULL, _IONBF, 0);
50
51         /* 총 4개의 프로세스 생성 */
52         fork(); fork();
53
54         do_work();
55         /* 세마포어 삭제 */
56         semctl(semid, 0, IPC_RMID, 0);
57
58         return 0;
59 }
60
```

Example - Semaphore (3)

```
61 void do_work()
62 {
63         int count = 0;
64
65         while (count < 3) {
66                 semop(semid, &decrease[PEN], 1);
67                 printf("[pid: %5d]연필을 들고\n", getpid());
68
69                 semop(semid, &decrease[NOTE], 1);
70                 printf("[pid: %5d]노트를 열고\n", getpid());
71
72                 printf("[pid: %5d]공부 시작\n", getpid());
73
74                 semop(semid, &increase[PEN], 1);
75                 printf("[pid: %5d]연필을 내려 놓음\n", getpid());
76
77                 semop(semid, &increase[NOTE], 1);
78                 printf("[pid: %5d]노트를 닫음\n\n", getpid());
79
80                 sleep(1);
81                 count++;
82         }
83 }
```

Shared Memory

- 공유메모리
  - 프로세스들이 공통으로 사용할 수 있는 메모리 영역
  - 특정 메모리 영역을 다른 프로세스와 공유하여 프로세스간 통신이 가능
  - 데이터를 한 번 읽어도 데이터가 계속 남아 있음
  - 같은 데이터를 여러 프로세스가 중복하여 읽어야 할 때 효과적

Shared Memory - shmget

사용법

```
#include <sys/ipc.h>
int shmget(key_t key, int size, int shmflg);
```

int 타입의 공유메모리 ID를 리턴
- struct shmid_ds 구조체에 정보를 저장

key : 새로 생성될 공유메모리를 식별하기 위한 값
- 다른 프로세스가 접근하기 위해서는 이 키 값을 알아야 함

shmflg : 공유메모리 생성 옵션
- bitmask 형태의 인자
- IPC_CREAT, IPC_EXCL, 파일접근권한

Shared Memory - shmflg

- IPC_CREAT를 설정한 경우
  - 같은 key값을 사용하는 공유메모리가 존재하면 해당 객체에 대한 ID를 리턴
  - 같은 key값의 공유메모리가 존재하지 않으면 새로운 공유메모리를 생성하고 그 ID를 리턴

- IPC_EXCL 과 IPC_CREAT를 같이 설정한 경우
  - 같은 key값을 사용하는 공유메모리가 존재하면 shmget() 호출은 실패하고 -1을 리턴
  - IPC_CREAT와 같이 사용해야 함

- IPC_PRIVATE
  - key값이 없는 공유메모리를 생성
    - 명시적으로 key값을 사용할 필요가 없는 경우에 사용
    - 공유메모리 ID를 서로 공유할 수 있는 부모와 자식 프로세스 사이에 사용가능
    - 외부의 다른 프로세스는 이 공유메모리에 접근 불가

Shared Memory - shmid_ds 구조체

```
sturct shmid_ds {
        struct ipc_perm shm_perm;       // 동작허가사항
        int shm_segsz;                  // 세그먼트의 크기
        (bytes)time_t shm_atime;        // 마지막attach 시각
        time_t shm_dtime;               // 마지막detach 시각
        time_t shm_ctime;               // 마지막change 시각
        unsigned short shm_cpid;        // 생성자의 PID
        unsigned short shm_lpid;        // 마지막접근자의 PID
        short shm_nattch;               // 현재attaches no.

        // 아래는 private
        unsigned short shm_npages;      // 세그먼트의크기(pages)
        unsigned long *shm_pages;       // array of ptrs to frames ->SHMMAX
        sturct vm_area_struct *attaches;// descriptors for attaches
};
```

Shared Memory - shmat

사용법

void *shmat(int shmid, const void *shmadr, int shmflg);

공유메모리 생성 후 실제 사용 전 물리적 주소를 자신의 프로세스의 가상메모리 주소로 매핑

- shmid : 공유메모리 객체 ID
- shmaddr : 첨부시킬 프로세스의 메모리주소
  - 0 : 커널이 자동으로 빈 공간을 찾아서 처리
- shmflg : 공유메모리옵션
  - 0 : 읽기/쓰기모드
  - SHM_RDONLY : 읽기전용
- 호출 성공 시 첨부된 주소를 리턴, error 시 NULL 포인터를 리턴

```
int shmid = shmget(0x1234, 1023, IPC_CREAT | 0600);
(void *)myaddr = shmat(shmid, (void *)0, 0);
if (myaddr = (void *)-1) {
        perror("공유메모리를 attach하지 못했습니다.₩n");
        exit(0);
}
```

Shared Memory - shmdt

사용법

int **shmdt**(const void *shmaddr);

자신이 사용하던 메모리 영역에서 공유메모리를 분리

- shmaddr : shmat()가 리턴했던 주소, 현재 프로세스에 첨부된 공유메모리의 시작주소
- 공유메모리의 분리가 공유메모리의 삭제를 의미하지는 않음
  - 다른 프로세스는 계속 그 공유메모리를 사용할 수 있음
- shmid_ds 구조체의 shm_nattach 멤버변수
  - shmat()로 공유메모리를 첨부하면 1 증가
  - 공유메모리를 분리하면 1 감소

Shared Memory - shmctl

사용법

int **shmctl**(int shmid, int cmd, struct shmid_ds *buf);

- 공유메모리의 정보 읽기, 동작 허가 권한 변경, 공유메모리 삭제 등의 공유메모리 제어
  - shmid : 공유메모리객체ID
  - cmd : 수행할 명령
    - IPC_STAT : 공유메모리 객체에 대한 정보를 얻어 오는 명령
    - IPC_SET : r/w 권한, euid, eguid를 변경하는 명령
    - IPC_RMID : 공유메모리를 삭제하는 명령
  - buf : cmd 명령에 따라 의미가 변경
    - 공유메모리 객체 정보를 얻어오는 명령: 얻어온 객체를 buf에 저장
    - 동작 허가 권한을 변경하는 명령: 변경할 내용을 저장
- 여러 프로세스가 병행하여 쓰기/읽기 작업을 수행하면 동기화 문제가 발생할 수 있음
  - 하나의 공유데이터를 둘 이상의 프로세스가 동시에 접근함으로써 발생할 수 있는 문제
  - 데이터의 값이 부정확하게 사용되는 문제가 있음

Example – Shared Memory

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int main(void)
{
        int shmid, pid, *cal_num;
        void *shared_memory = (void *)0;

        /* 공유 메모리 공간 생성 */
        shmid = shmget((key_t)1234, sizeof(int), 0666|IPC_CREAT);

        if (shmid == -1) {
                perror("shmget failed: ");
                exit(0);
        }

        /* 공유메모리를 사용하기 위해 프로세스 메모리에 붙인다 */
        shared_memory = shmat(shmid, (void *)0, 0);
        if (shared_memory == (void *)-1) {
                perror("shmat failed: ");
                exit(0);
        }

        cal_num = (int *)shared_memory;
        if ((pid = fork()) == 0) {
                *cal_num = 1;
                while(1) {
                        *cal_num = *cal_num + 1;
                        printf("[CHILD] %d\n", *cal_num);
                        sleep(1);
                }
        } else if (pid > 0) {
                while(1) {
                        sleep(1);
                        printf("[PARENT] %d\n", *cal_num);
                }
        }
}
```

Shared Memory - 응용실습

- Shared Memory 실습 코드를 이용하여, 자식 프로세스는 SIGINT 시그널을 받는 즉시, shmctl을 이용하여 메모리에서 사용하였던 공유 메모리를 제거하는 프로그램을 작성하시오!

(단, 부모 프로세스는 SIGINT 시그널의 처리를 SIG_IGN로 등록하고, 자식 프로세스 종료되는 것을 확인한 후 정상 종료하도록 프로그램 작성)

Advanced IPC 함수명의 유사성

	Message Queue	Shared Memory	Semaphore
1. IPC 할당방법	**msgget()**	**shmget()**	**semget()**
2. IPC 제어 방법 (상태 변경, 해제)	**msgctl()**	**shmctl()**	**semctl()**
3. IPC 작동방법	**msgsnd()**	**shmat()**	**semop()**
	msgrcv()	**shmdt()**	

Q & A

- Thank you :)