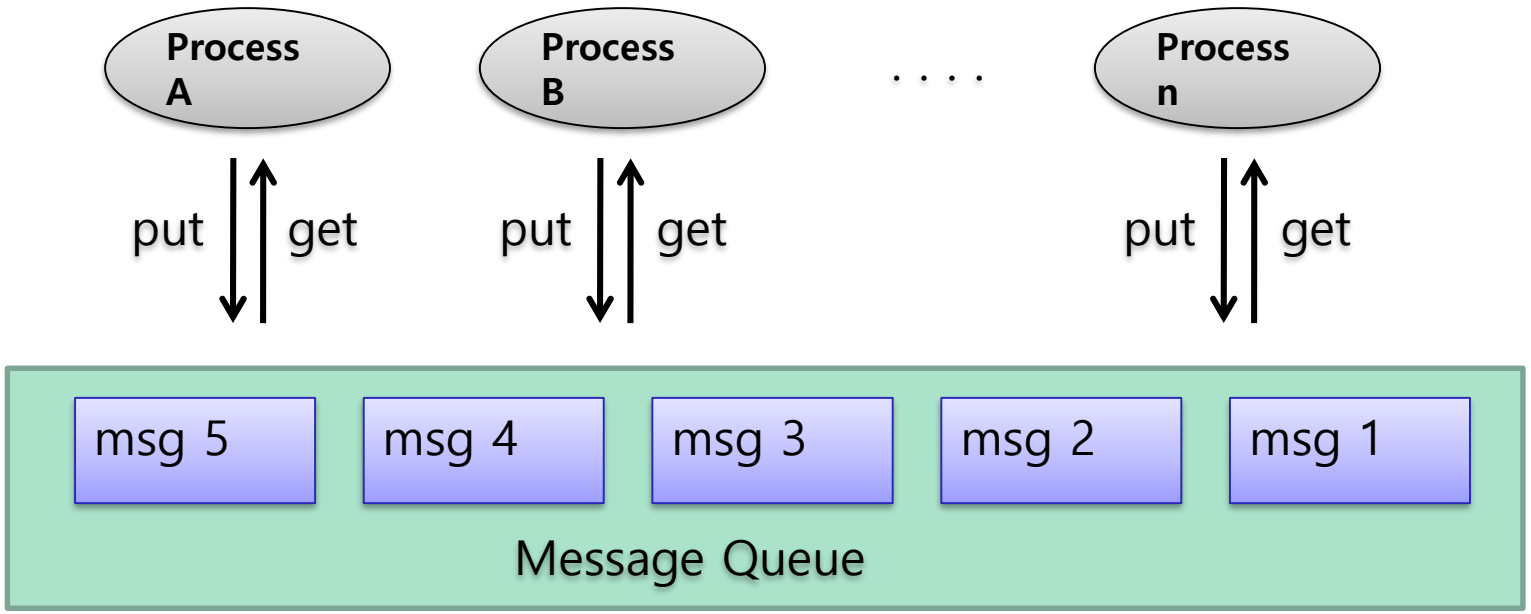
Advanced IPC
part 1

한양대학교 컴퓨터공학과

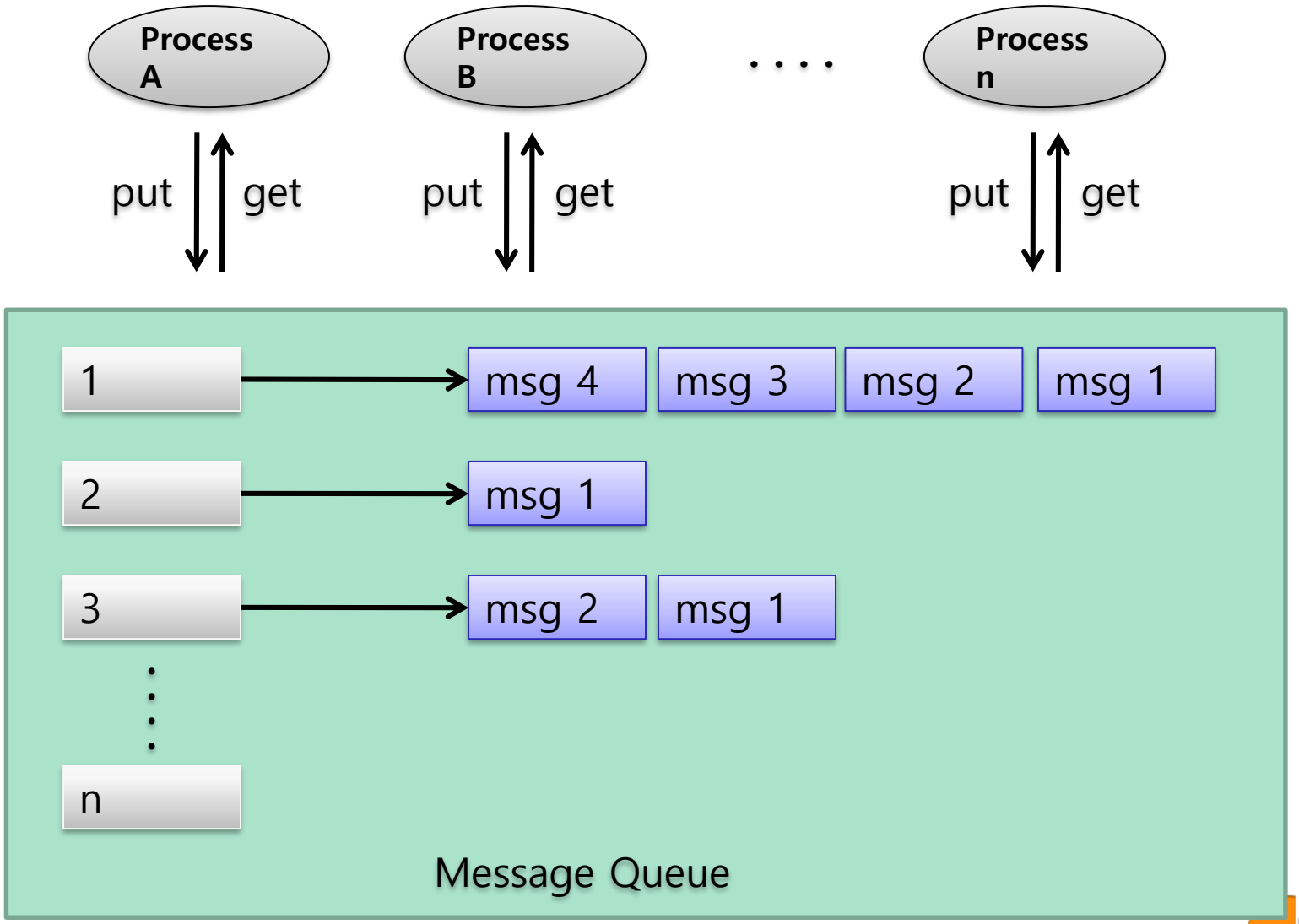
Dept. of Computer Science & Engineering
Hanyang University

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;                  // owner의 euid와egid
        ushort udi;
        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

```
31 int main(int argc, char* argv[])
32 {
33         int msg_prio, msgq_id, running = 1;
34         char request[MAXLENGTH + 1];
35
36         msgq_id = msgget(MSGQKEY, IPC_CREAT|PERMISSION);
37
38         printf("%d\n", msgq_id);
39
40         if (msgq_id == -1) {
41                 perror("[c]msg queue create failed");
42                 exit(1);
43         }
44         while(running) {
45                 strncpy(request, argv[1], MAXLENGTH);
46                 printf("[c]argv[1]: %s\n", request);
47                 msg_prio = atoi(argv[2]); /* 문자열을 숫자로 변환 */
48                 printf("[c]argv[2]: %d\n", msg_prio);
49
50                 if (msg_put(msgq_id, request, msg_prio) < 0) {
51                         perror("[c]msg send failed");
52                         exit(1);
53                 }
54
55                 if (!strcmp(request, "end")) {   /* 끝내고자 할 때 : end와 20을 입력 */
56                         printf("[c]>>>> NULL <<<<\n");
57                         strcpy(request, "end");
58                         msg_prio = MAXPRIO;
59
60                         if (msg_put(msgq_id, request, msg_prio) < 0) {
61                                 perror("[c]msg send failed");
62                                 exit(1);
63                         }
64                 }
65                 running = 0;
66         }
67         exit(0);
68 }
```

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 }
```

Shared Memory

- 공유메모리
  - 프로세스들이 공통으로 사용할 수 있는 메모리 영역
  - 특정 메모리 영역을 다른 프로세스와 공유하여 프로세스 간 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)

Q & A

- Thank you :)