Interprocess Communication

- Pipes
- FIFOs
- System V IPC
  - Message Queues
  - Shared Memory
  - Semaphores

pipe

- IPC using regular files
  - unrelated processes can share
  - fixed size
  - lack of synchronization

- IPC using pipes
  - for transmitting data between related processes
  - can transmit an unlimited amount of data
  - automatic synchronization on open()

Pipe in a Unix Shell

- In a UNIX shell, the pipe symbol is: **|** (the vertical bar)
- In a shell, UNIX pipes look like:

```
$ ls -alg | more
```

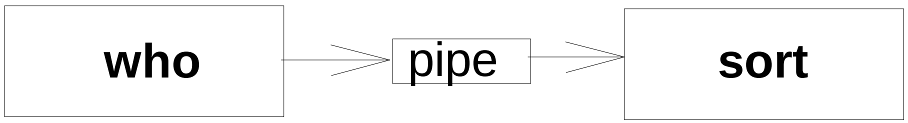
  - where the standard output of the program at the left (i.e., the producer) becomes the standard input of the program at the right (i.e., the consumer).

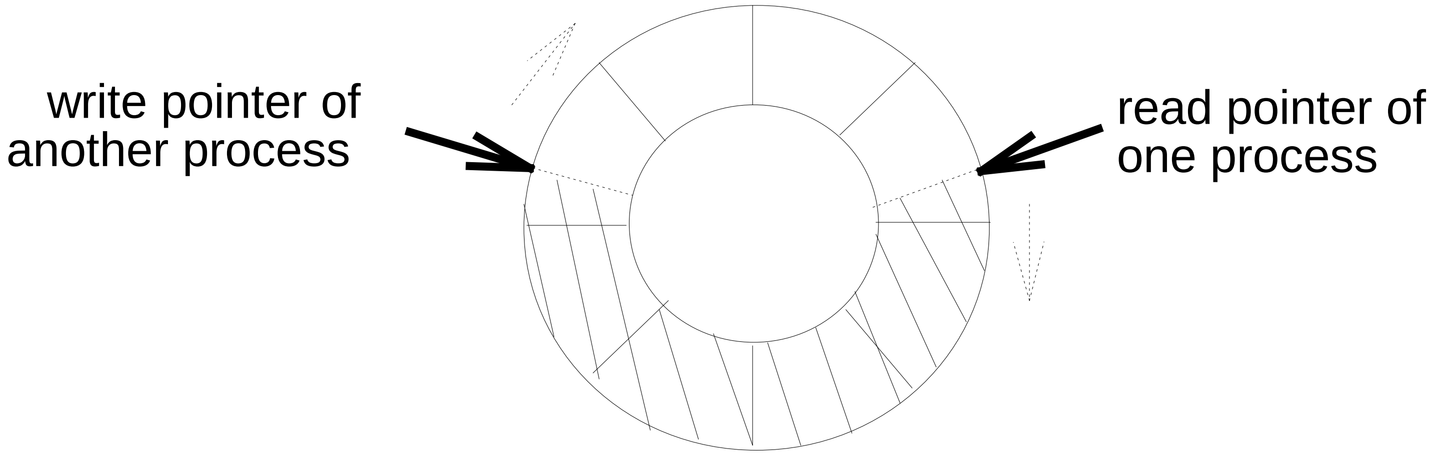
- We can have longer pipes:

```
$ ps –ef | sort | more
```

Example

`$ who | sort`





pipe

Data transmitting

- data is written into pipes using the write() system call
- data is read from a pipe using the read() system call
- automatic blocking when full or empty

Types of pipes

- (unnamed) pipes
- named pipes (FIFOs)

pipe(1/4)

- In UNIX, pipes are the oldest form of IPC.
- Limitations of Pipes:
  - Half duplex (data flows in one direction)
  - Can only be used between processes that have a common ancestor (Usually used between the parent and child processes)
  - Processes cannot pass pipes and must inherit them from their parent
  - If a process creates a pipe, all its children will inherit it

pipe(2/4)

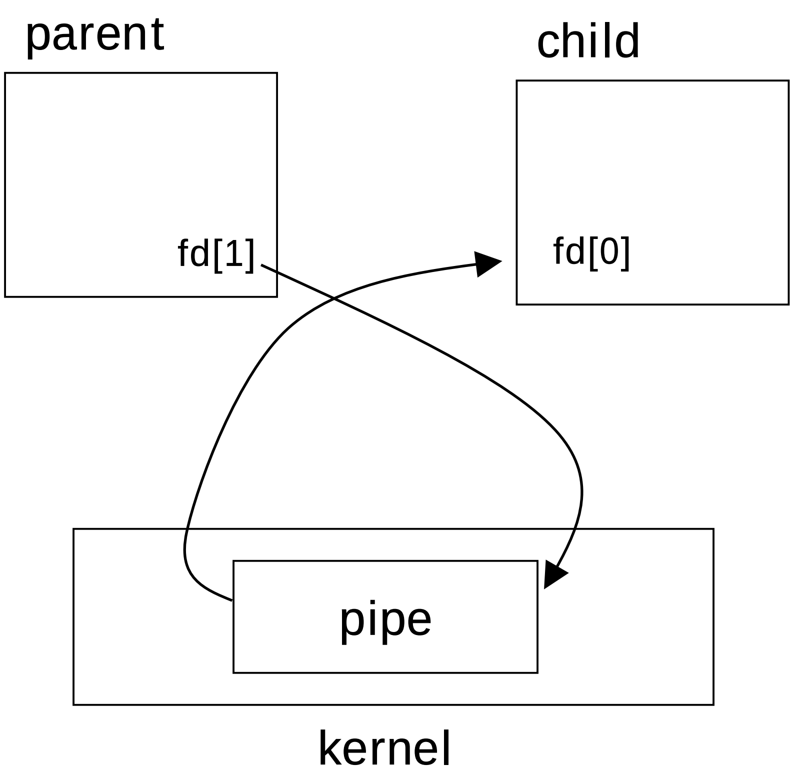
```
#include <unistd.h>

int pipe(int fd[2])
                              Returns: 0 if OK, -1 on error
```

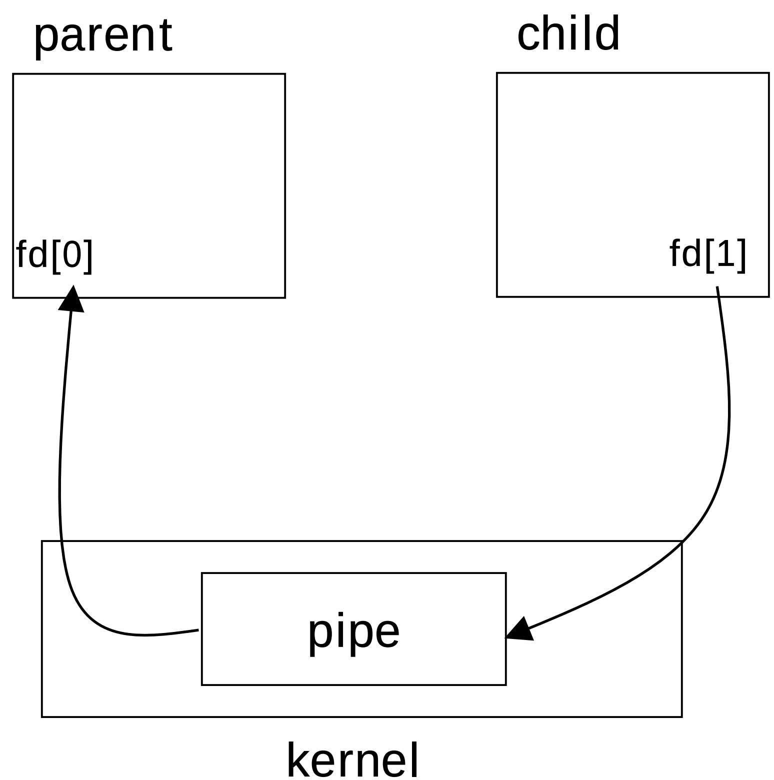
- Two file descriptors are returned through the *fd* argument
  - *fd*[0]: can be used to read from the pipe, and
  - *fd*[1]: can be used to write to the pipe
- Anything that is written on *fd*[1] may be read by *fd*[0].
  - This is of no use in a single process.
  - However, between processes, it gives a method of communication
- The pipe() system call gives parent-child processes a way to communicate with each other.

pipe(3/4)

parent → child:
- parent closes fd[0]
- child closes fd[1]

parent ← child:
- parent closes fd[1]
- child closes fd[0]





pipe(3/4)

- Read from a pipe with write end closed: (fd[1]이 close된 경우)
  - returns 0 to indicate EOF

- Write to a pipe with read end closed: (fd[0]가 close된 경우)
  - SIGPIPE generated,
  - write() returns error (errno == EPIPE)

```c
#include <stdio.h> // pipe.c

#define READ  0
#define WRITE 1

char* phrase = "Stuff this in your pipe and smoke it";

int main( ) {
    int fd[2], bytesRead;
    char message[100];
    pipe(fd);
    if (fork() == 0) {  // child
        close(fd[READ]);
        write(fd[WRITE], phrase, strlen(phrase)+1);
        fprintf(stdout, "[%d, child] write completed.\n", getpid());
        close(fd[WRITE]);
    }
    else {  // parent
        close(fd[WRITE]);
        bytesRead = read(fd[READ],   message, 100);
        fprintf(stdout, "[%d, parent] read completed.\n", getpid());
        printf("Read %d bytes: %s\n", bytesRead,message);
        close(fd[READ]);
    }
}
```

System V IPC

Message Queues

- Send and receive amount of data called “messages”.
- The sender classifies each message with a type.

Shared Memory

- Shared memory allows two or more processes to share a given region of memory.
- Readers and writers may use semaphore for synchronization.

Semaphores

- Process synchronization and resource management
- For example, a semaphore might be used to control access to a device like printer.

Identifier & Key

- Identifier : each IPC structure has a nonnegative integer
- Key: when creating an IPC structure, a *key* must be specified (**`key_t`**)

```
id = xxxget(key, …)
```

- How to access the same IPC? → key in a common header
  - Define a key in a common header
  - Client and server agree to use that key
  - Server creates a new IPC structure using that key
  - Problem when the key is already in use
    - (msgget, semget, shmget returns error)
    - Solution: delete existing key, create a new one again!

IPC System Calls

- **`msg/sem/shm get`**
  - Create new or open existing IPC structure.
  - Returns an IPC identifier
- **`msg/sem/shm ctl`**
  - Determine status, set options and/or permissions
  - Remove an IPC identifier
- **`msg/sem/shm op`**
  - Operate on an IPC identifier
  - For example(Message queue)
    - add new msg to a queue (msgsnd)
    - receive msg from a queue (msgrcv)

Permission structure

- ipc_perm is associated with each IPC structure.
- Defines the permissions and owner.

```
struct ipc_perm {
  uid_t uid;   /* owner's effective user id */
  gid_t gid;   /* owner's effective group id */
  uid_t cuid;  /* creator's effective user id */
  gid_t cgid;  /* creator’s effective group id */
  mode_t mode; /* access modes */
  ulong seq;   /* slot usage sequence number */
  key_t key;   /* key */
};
```

message queue

- Linked list of messages
  - Stored in kernel
  - Identified by message queue identifier (in kernel)

- msgget
  - Create a new queue or open existing queue.
- msgsnd
  - Add a new message to a queue
- msgrcv
  - Receive a message from a queue
  - Fetching order: based on type

message queue

- Each queue has a structure

```
struct msqid_ds {
  struct ipc_perm msg_perm;
  struct msg *msg_first; /* ptr to first msg on queue */
  struct msg *msg_last;  /* ptr to last msg on queue */
  ulong msg_cbytes;      /* current # bytes on queue */
  ulong msg_qnum;        /* # msgs on queue */
  ulong msg_qbytes;      /* max # bytes on queue */
  pid_t msg_lspid;       /* pid of last msgsnd() */
  pid_t msg_lrpid;       /* pid of last msgrcv() */
  time_t msg_stime;      /* last-msgsnd() time */
  time_t msg_rtime;      /* last-msgrcv() time */
  time_t msg_ctime;      /* last-change time */
};
```

- We can get the structure using msgctl() function.
- Actually, however, we don’t need to know the structure in detail.

msgget()

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

int msgget(key_t key, int flag);
                         Returns: msg queue ID if OK, -1 on error
```

- Create new or open existing queue
- flag : ipc_perm.mode
- Example

```
msg_qid = msgget(DEFINED_KEY, IPC_CREAT | 0666);
```

msgctl()

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

int msgctl(int msqid, int cmd, struct msqid_ds *buf);
                     Returns: 0 if OK, -1 on error
```

- Performs various operations on a queue
- **cmd = IPC_STAT:**

  fetch the msqid_ds structure for this queue, storing it in buf
- **cmd = IPC_SET:**

  set the following four fields from buf: msg_perm.uid, msg_perm.gid, msg_perm.mode, and msg_qbytes
- **cmd = IPC_RMID:**

  remove the message queue.

msgsnd()

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

int msgsnd(int msqid, const void *ptr, size_t nbytes, int flag);
                              Returns: 0 if OK, -1 on error
```

- msgsnd() places a message at the end of the queue.
  - ptr: pointer that points to a message
  - nbytes: length of message data
  - if flag = IPC_NOWAIT: IPC_NOWAIT is similar to the nonblocking I/O flag for file I/O.
- Structure of messages

```
struct mymsg {
  long mtype;          /* positive message type */
  char mtext[512];     /* message data, of length nbytes */
};
```

msgrcv()

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

int msgrcv(int msqid, void *ptr, size_t nbytes, long type, int flag);
                        Returns: data size in message if OK, -1 on error
```

- msgrcv() retrieves a message from a queue.
- type == 0: the first message on the queue is returned
- type > 0: the first message on the queue whose message type equals type is returned
- type < 0: the first message on the queue whose message type is the lowest value less than or equal to the absolute value of type is returned
- flag may be given by IPC_NOWAIT

sender.c

```
#include <stdio.h>  // sender.c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define DEFINED_KEY 0x10101010

main(int argc, char **argv)
{
    int msg_qid;
    struct {
        long mtype;
        char content[256];
    } msg;

    fprintf(stdout, "=========SENDER==========\n");

    if((msg_qid = msgget(DEFINED_KEY, IPC_CREAT | 0666)) < 0) {
        perror("msgget: "); exit(-1);
    }

    msg.mtype = 1;
    while(1) {
        memset(msg.content, 0x0, 256);
        gets(msg.content);
        if(msgsnd(msg_qid, &msg, sizeof(msg.content), 0) < 0) {
            perror("msgsnd: "); exit(-1);
        }
    }
}
```

receiver.c

```
#include <stdio.h>  // receiver.c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define DEFINED_KEY 0x10101010

main(int argc, char **argv)
{
    int msg_qid;
    struct {
        long mtype;
        char content[256];
    } msg;

    fprintf(stdout, "=========RECEIVER==========\n");

    if((msg_qid = msgget(DEFINED_KEY, IPC_CREAT | 0666)) < 0) {
        perror("msgget: "); exit(-1);
    }

    while(1) {
        memset(msg.content, 0x0, 256);
        if(msgrcv(msg_qid, &msg, 256, 0, 0) < 0) {
            perror("msgrcv: "); exit(-1);
        }
        puts(msg.content);
    }
}
```

Shared memory

- Allows multiple processes to share a region of memory
  - Fastest form of IPC: no need of data copying between client & server
- If a shared memory segment is attached
  - It become a part of a process data space, and shared among multiple processes
- Readers and writers may use semaphore to
  - synchronize access to a shared memory segment

Shared memory structure

- Each shared memory has a structure

```
struct shmid_ds {
 struct ipc_perm shm_perm;
 struct anon_map *shm_amp; /* pointer in kernel */
 int shm_segsz;            /* size of segment in bytes */
 ushort shm_lkcnt;   /* # of times segment is being locked */
 pid_t shm_lpid;     /* pid of last shmop() */
 pid_t shm_cpid;     /* pid of creator */
 ulong shm_nattch;   /* # of current attaches */
 ulong shm_cnattch;  /* used only for shminfo() */
 time_t shm_atime;   /* last-attach time */
 time_t shm_dtime;   /* last-detach time */
 time_t shm_ctime;   /* last-change time */
};
```

- We can get the structure using shmctl() function.
- Actually, however, we don't need to know the structure in detail.

shmget()

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

int shmget(key_t key, int size, int flag);
                  Returns: shared memory ID if OK, -1 on error
```

- Obtain a shared memory identifier
- size: is the size of the shared memory segment
- flag: ipc_perm.mode
- Example

```
shmid = shmget(key, size, PERM|IPC_CREAT|IPC_EXCL|0666);
```

shmctl

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

int shmctl(int shmid, int cmd, struct shmid_ds *buf);
                                 Returns: 0 if OK, -1 on error
```

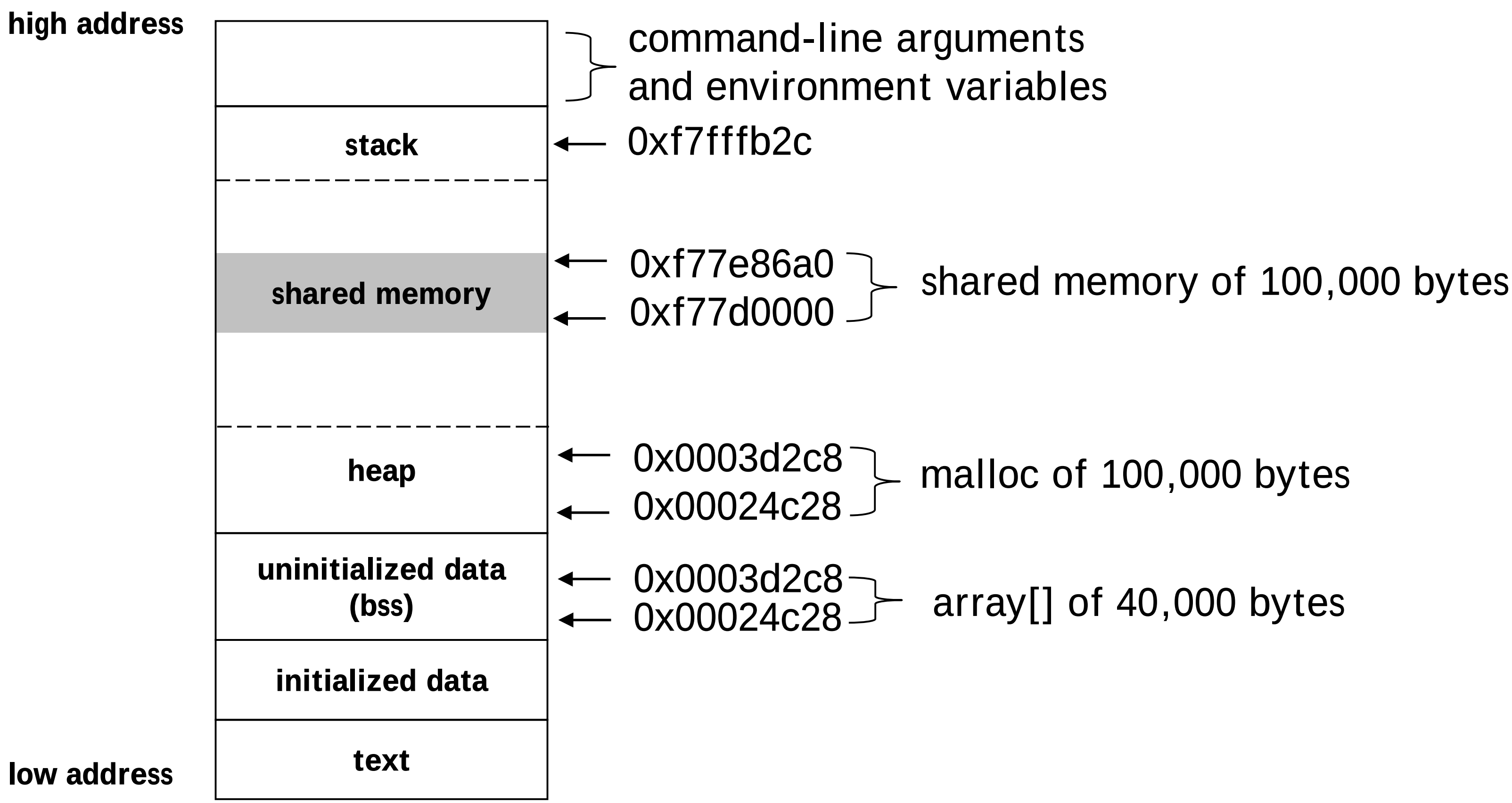
- Performs various shared memory operations
- **cmd = IPC_STAT**:
  fetch the shmid_ds structure into *buf*
- **cmd = IPC_SET**:
  set the following three fields from *buf*: shm_perm.uid, shm_perm.gid, and shm_perm.mode
- **cmd = IPC_RMID**:
  remove the shared memory segment set from the system

shmat()

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

void *shmat (int shmid, void *addr, int flag);
   Returns: pointer to shared memory segment if OK, -1 on error
```

- Attached a shared memory to an address
- flag = SHM_RDONLY: the segment is read-only
- addr==0: at the first address selected by the kernel (recommended!)
- addr!=0: at the address given by addr

Memory Layout



shmdt()

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

void shmdt (void *addr);
                                Returns: 0 if OK, -1 on error
```

- Detach a shared memory segment

```c
#include  <sys/types.h> // shm.c
#include  <sys/ipc.h>
#include  <sys/shm.h>

#define  ARRAY_SIZE   100000
#define  MALLOC_SIZE  100000
#define  SHM_SIZE     100000

err_sys(char *p) { perror(p); exit(-1); }

char  array[ARRAY_SIZE];  /* uninitialized data = bss */

int main(void) {
  int shmid;  char  *ptr, *shmptr;

  printf("array[] from %x to %x\n", &array[0], &array[ARRAY_SIZE]);
  printf("stack around %x\n", &shmid);

  if ((ptr = malloc(MALLOC_SIZE)) == NULL) err_sys("malloc error");
  printf("malloced from %x to %x\n", ptr, ptr+MALLOC_SIZE);

  if ((shmid = shmget(0x01010101, SHM_SIZE, IPC_CREAT | 0666)) < 0)
     err_sys("shmget error");

  if ((shmptr = shmat(shmid, 0, 0)) == (void *) -1) err_sys("shmat error");
  printf("shared memory attached from %x to %x\n", shmptr, shmptr+SHM_SIZE);

  // if (shmctl(shmid, IPC_RMID, 0) < 0) err_sys("shmctl error");

  exit(0);
}
```

shm.c

실행 결과

$ shm

array[] from 20bd8 to 39278

stack around ffbffa9c

malloced from 39288 to 51928

shared memory attached from ff260000 to ff2786a0

$ ipcs -ma

IPC status from <running system> as of 2009년 11월 14일 토요일 오후 06시 45분 09초

T ID KEY MODE OWNER GROUP CREATOR CGROUP NATTCH SEGSZ CPID LPID ATIME DTIME CTIME

Shared Memory:

m 1 0x1010101 --rw-rw-rw- yshin prof yshin prof 0 100000 2413 2465 18:42:50 18:42:50 18:32:37

$ ipcrm –m 1

$ ipcs -ma

```c
#include <stdio.h>            // shm1.c
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define KEY  0x101010
#define SIZE 1024

main()
{
    int shmid;
    char *shmaddr;

    if ((shmid=shmget(KEY, SIZE, IPC_CREAT|0666)) == -1) {
       perror("shmid failed");
       exit(1);
    }

    if ((shmaddr=shmat(shmid, NULL, 0)) == (void *)-1) {
       perror("shmat failed");
       exit(1);
    }
    strcpy((char *)shmaddr, "HELLO KIM!!");

    if (shmdt(shmaddr) == -1) {
       perror("shmdt failed");
       exit(1);
    }
}
```

```c
#include <stdio.h>        // shm2.c
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define KEY 0x101010
#define SIZE 1024

main()
{
    int shmid;
    void *shmaddr;

    if((shmid=shmget(KEY, SIZE, IPC_CREAT|0666)) == -1) {
       perror("shmid failed");
       exit(1);
    }

    if((shmaddr=shmat(shmid, NULL, 0)) == (void *)-1) {
       perror("shmat failed");
       exit(1);
    }
    printf("data read from shared memory : %s\n", (char
*)shmaddr);

    if(shmdt(shmaddr) == -1) {
       perror("shmdt failed");
       exit(1);
    }

    if(shmctl(shmid, IPC_RMID, 0) == -1) {
       perror("shmctl failed");
       exit(1);
    }
}
```

semaphore

- A counter to provide access to shared data object for multiple processes (**복수의 프로세스가 데이터를 공유하는데 사용하는 카운터**)

- To obtain a shared resource:
  - 1. Test semaphore that controls the resource (**확인하여**)
  - 2. If value > 0, value--, grant use (**양수이면**, **감소시키고 사용하고**)
  - 3. If value == 0, sleep until value > 0 (0**이면 기다림**)
  - 4. Release resource, value ++ (**다 쓴 후에는 다시 양수로 만듦**)

- Step 1, 2 must be an atomic operation

Semaphore structure

- Each semaphore has a structure

```
struct semid_ds {
  struct ipc_perm sem_perm;
  struct sem *sem_base; /*ptr to first semaphore in set */
  ushort sem_nsems;     /* # of semaphors in set */
  time_t sem_otime;     /* last-semop() time */
  time_t sem_ctime;     /* last-change time */
};

struct sem {
  ushort semval;  /* semaphore value, always >= 0 */
  pid_t  sempid;  /* pid for last operation */
  ushort semncnt; /* # processes awaiting semval > currval */
  ushort semzcnt; /* # processes awaiting semval = 0 */
};
```

- We can get the structure using semctl() function.
- Actually, however, we don’t need to know the structure in detail.

semget()

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

int semget(key_t key, int nsems, int flag);
                    Returns: semaphore ID if OK, -1 on error
```

- Obtain a semaphore ID
- nsems: sem_nsens (# of semaphores in set)
- flag: ipc_perm.mode

semctl()

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

int semctl(int semid, int semnum, int cmd, union semun arg);

union semun {
  int                val;   /* for SETVAL */
  struct semid_ds *buf;   /* for IPC_START and IPC_SET */
  ushort           *array; /* for GETALL and SETALL */
};
```

- To use semaphore, please refer to the textbook and manuals related semaphore.

ipcs, ipcrm

- ipcs:System V IPC의 상태를 확인하는 명령어
  - `$ ipcs` // IPC 정보를 확인 (q, m, s 모두)
  - `$ ipcs –q ($ ipcs –qa)` // Message Queue 정보를 확인
  - `$ ipcs –m ($ ipcs –ma)` // Shared Memory 정보를 확인
  - `$ ipcs –s ($ ipcs –sa)` // Semaphore 정보를 확인

- ipcrm: 정의된(생성된) IPC를 삭제함
  - `$ ipcrm –q id` // Message Queue를 삭제
  - `$ ipcrm –m id` // Shared Memory를 삭제
  - `$ ipcrm –s id` // Semaphore를 삭제