System Programming

12. IPC methods

Seung-Ho Lim
Dept. of Computer & Electronic Systems Eng.

IPC methods

- Pipe
  - can be used only b/w related processes (e.g. parent and child)
- **FIFO**
  - named pipes that can be used b/w unrelated processes
- **Message Queues**
- **Shared Memory (b/w processes)**

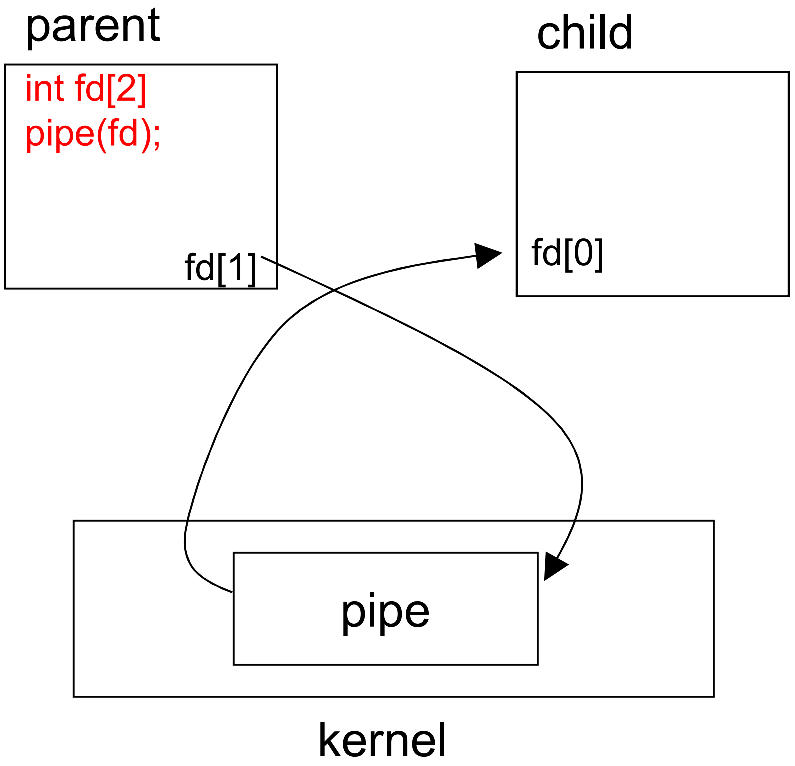
FIFO
(named pipe)

Review on `pipe()` (1)

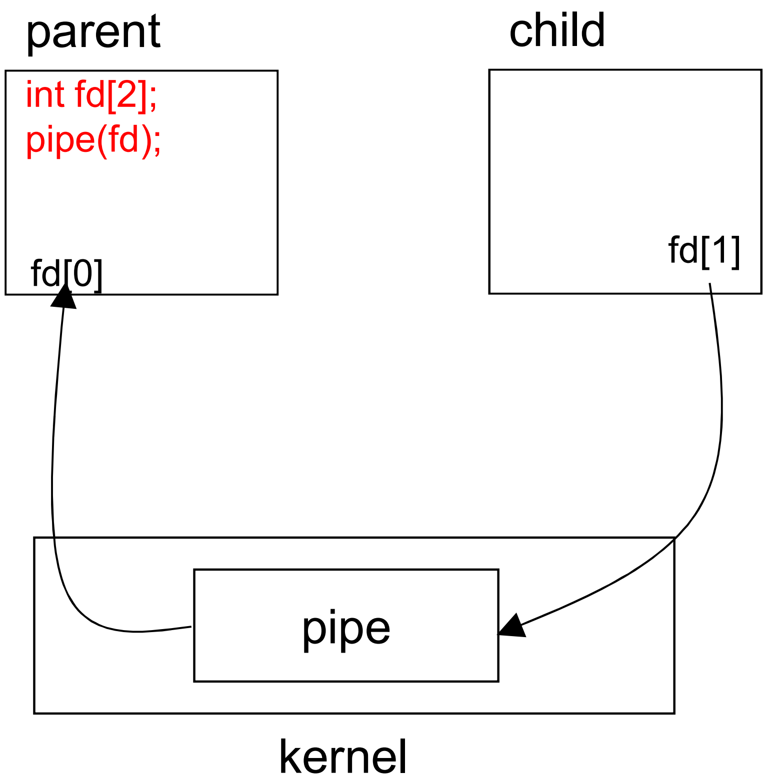
- Pipes are the oldest form of IPC
- Data transmitting
  - data is written into pipes using the write() system call
  - data is read from a pipe using the read() system call
  - automatic blocking when full or empty
- Limitations of pipes:
  - half duplex (data flows in one direction)
  - can only be 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
- Note
  - unused file descriptor of the pipe must be closed (if not?)

Review on `pipe()` (2)

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



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



FIFO

- Pipes can be used only between related processes. (e.g., parent and child processes)
- FIFOs are "named pipes"
  - can be used between unrelated processes.
- A type of file
  - `stat.st_mode` == FIFO
  - can test with `S_ISFIFO()` macro

mkfifo

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

int mkfifo(const char *pathname, mode_t mode);
```

- creating a FIFO
  - is similar to creating a file.
- parameters
  - `pathname`: filename
  - `mode`: permisson, same as for file open() function
- return
  - 0 if OK / -1 on error
- using a FIFO is similar to using a file.
  - we can open, close, read, write, unlink, etc., to the FIFO.

Fifo send-recv example

fifo-recv.c

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#define SIZE 128
#define FIFO "fifo"

int main(int argc, char *argv[]) {
    int fd;
    char buffer[SIZE];

   if(mkfifo(FIFO, 0666) == -1) {
      perror("mkfifo failed");
      exit(1);
    }

    if((fd=open(FIFO, O_RDWR)) == -1) {
      perror("open failed");
      exit(1);
    }

    while(1) {
      if(read(fd, buffer, SIZE) == -1) {
         perror("read failed");
         exit(1);
      }
      if(!strcmp(buffer, "quit"))
         exit(0);
      printf("receive message: %s\n", buffer);
    }
}
```

fifo-send.c

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#define SIZE 128
#define FIFO "fifo“

main(int argc, char *argv[])
{
    int fd,i;
    char buffer[SIZE];

    if((fd=open(FIFO,O_WRONLY)) == -1) {
       perror("open failed");
       exit(1);
    }

    for(i=1 ; i<argc ; i++) {
       strcpy(buffer, argv[i]);

       if(write(fd,buffer,SIZE) == -1) {
          perror("write failed");
          exit(1);
       }
    }
    exit(0);
}
```

Fifo send-recv example

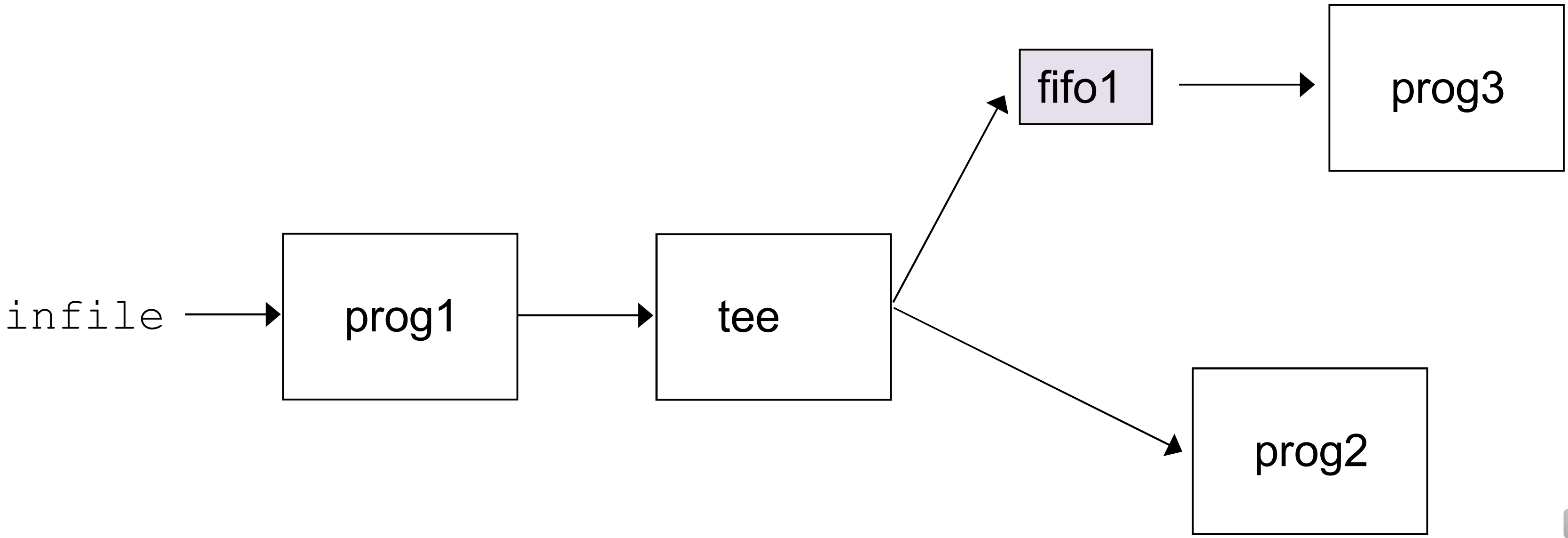
```
# ./fifo-recv
receive message: system
receive message: programming
```

```
#./fifo-send system programming
```

FIFO Example in shell (1)

```
$ mkfifo fifo1
$ prog3 < fifo1 &
$ prog1 < infile | tee fifo1 | prog2
```

- in this example, `tee`(1) command
  - copies its standard input to both its standard output and to the file named on its command line.



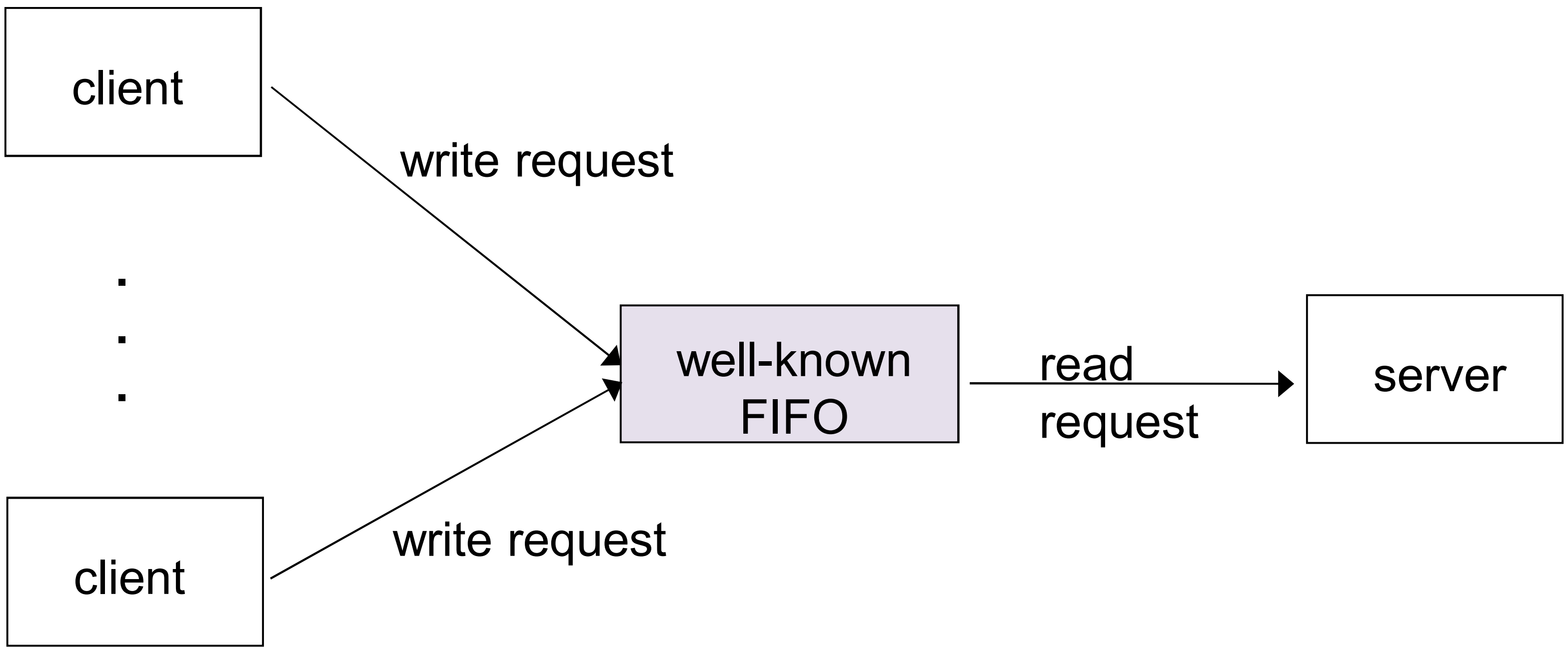
FIFO Example in shell (2)

```
$ cat myfifo &
$ cat hello.c | tee myfifo
#include <stdio.h>

void main(void)
{
        printf("Hello World! \n");
}

#include <stdio.h>

void main(void)
{
        printf("Hello World! \n");
}
```

아래 명령이 완료
된 후에 출력됨

FIFO Example for client-server

- Server creates a “well-known” FIFO to communicate with clients.



System V IPC

message queue
shared memory

IPC methods of System V

- 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.

Identifiers & Keys

- **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 object?
  - Define a key in a common header (by a programmer)
  - 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`, `shmget` returns error)
    - solution: delete existing key, create a new one again!

IPC system calls

- **`msg/shm get`**
  - create new or open existing IPC structure.
  - returns an IPC identifier if OK (or -1 on error with `errno` value)
- **`msg/shm ctl`**
  - determine status, set options and/or permissions
  - remove an IPC identifier
- **`msg/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

Message Queues (1/2)

- 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
  - message fetching order: based on a specified type

Message Queues (2/2)

- 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 */
};
```

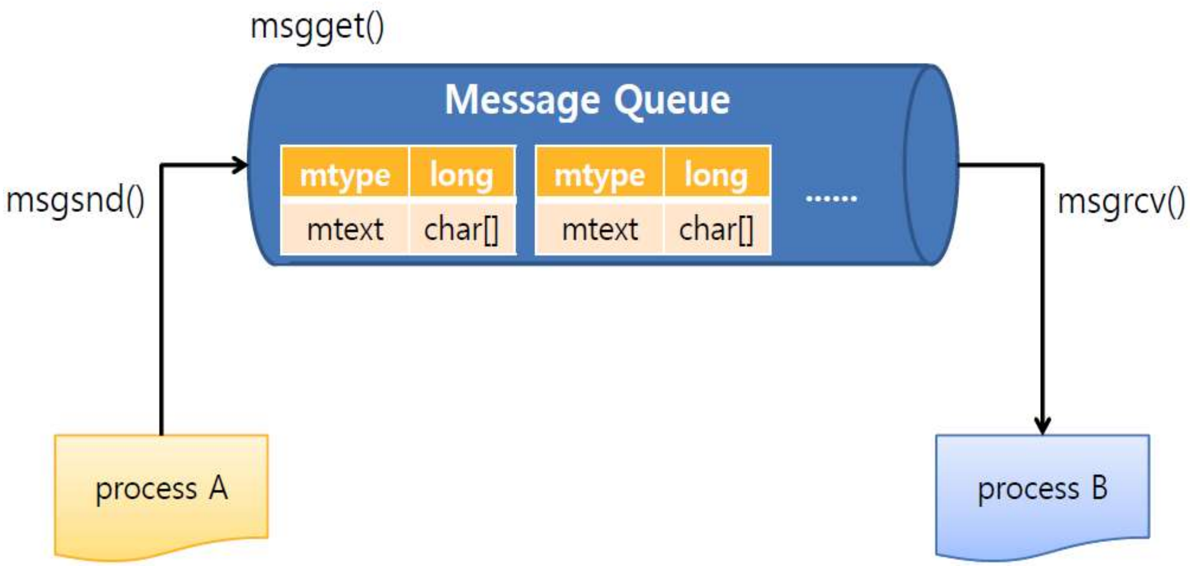
  - get the structure using `msgctl`() function.

- Actually, we don't need to know the structure in detail.

Message Queue structure

- message structure

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



Message Queue Parameters

- Each message queue is limited in terms of both
  - the maximum number of messages it can contain
  - the maximum number of bytes it may contain
- New messages cannot be added if either limit is hit
  - new writes will normally block
- On linux,
  - these limits are defined as in `/usr/include/linux/msg.h`):
  - also, can be changed by `sysctl` command or configured in `/etc/sysctl.conf`

name	description	defaults
MSGMNB	Max bytes in a queue	16,384
MSGMNI	Max # of message queue identifiers	32,000
MSGMAX	Max size of message (bytes)	8,192

msgget

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

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

- parameter
  - key : message queue key (id)
  - flag : same as in open()/create()
- return
  - message queue id if OK
  - -1 on error
- example:

  msg_qid = msgget(DEFINED_KEY, IPC_CREAT | 0666);

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);
```

- parameters
  - `msqid` : message queue id
  - `ptr` : user message pointer (to the message structure)
  - `nbytes` : message size (of the message contents)
  - `flag` : blocking or not when the buffer space is not enough
    - 0 for blocking (default)
    - IPC_NOWAIT for nonblocking I/O
- return
  - 0 if OK / -1 on error

msgrcv (1)

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

int msgrcv
(int msqid, void *ptr, size_t nbytes, long msgtype, int flag);
```

- parameters
  - *msqid* : message queue id
  - *ptr* : receive buffer address
  - *nbytes* : size of the buffer
  - *msgtype*: (*see the next page*)
  - *flag* :
    - 0:default (blocked when no message)
    - IPC_NOWAIT: return if no message
- return
  - 0 if OK / -1 on error

`msgrcv` (2)

- msgtype
  - == 0: the first message on the queue is returned
  - > 0: the first message on the queue whose message type equals to the `msgtype` is returned
  - < 0: the first message on the queue whose message type is the lowest value (less than or equal to *absolute* value of the `msgtype`) is returned

msgctl (1)

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

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

- performs various operation on a queue management
- parameters
  - `msqid` : message queue id
  - `cmd` : queue control (IPC_STAT, IPC_SET, IPC_RMID)
  - `buf` : `msqid_ds` structure pointer (for IPC_SET or IPC_STAT)
- return
  - 0 if OK / -1 on error

msgctl (2)

- commands

cmd	description
IPC_STAT	copy the message queue descriptor structure to the user buffer.
IPC_SET	set the system's message queue descriptor structure as given by the user buffer.
IPC_RMID	remove the message queue, and wake up all the sender and receiver processes. when the processes resume, error-returned. (errno = EIDRM)

ipcs command

- ipcs: System V IPC resource 상태를 확인

```
$ ipcs          // IPC 정보를 확인 (q, m, s 모두)
$ ipcs -q       // Message Queue 정보를 확인
$ ipcs -m       // Shared Memory 정보를 확인
$ ipcs -s       // Semaphore 정보를 확인
```

- ipcrm: 생성된 IPC resource를 제거

```
$ ipcrm -q id       // Message Queue를 제거
$ ipcrm -m id       // Shared Memory를 제거
$ ipcrm -s id       // Semaphore를 제거
```

Simple header file

mymessage.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/stat.h>
#include <sys/types.h>

#define MSIZE   256

struct umsg {
   long mtype;
   char mtext[MSIZE];
};
```

Receiver Example (1)

crecv.c

```
#include "mymessage.h"

int main( int argc, char *argv[])
{     int qid, size;
      struct umsg mymsg;
      long type;

      if(argc != 3) {
            fprintf (stderr, "usage : %s [key] [type]\n", argv[0]);
            perror ("msqq"); exit(1);
      }
     if((qid = msgget((key_t) atoi(argv[1]), IPC_CREAT | 0660)) == -1)
     {      perror ("msgget");
            exit (1);
     }
     type = (long) atoi(argv[2]);
     if (type == 0) {
       msgctl(qid, IPC_RMID, NULL);
       exit(0);
     }
```

Receiver Example (2)

crecv.c

```
    while (1) {
            memset(mymsg.mtext, 0, MSIZE);
            if((size = msgrcv (qid, &mymsg, (size_t)MSIZE, (long) atoi(argv[2]), 0)) == -1)
            {   perror ("msgrcv");
                exit (2);
             } else  {
                mymsg.mtext[strlen(mymsg.mtext)] = '\0';
                printf ("At receiver, mtype = %ld, mtext(%d) = %s\n",
                                                    mymsg.mtype, size, mymsg.mtext);
                if (mymsg.mtext=="quit") break;
            }
    }
}
```

Sender Example (1)

csend.c

```
#include "mymessage.h"

int main (int argc, char *argv[])
{     int qid;
      long type;
      struct umsg mymsg;

      if( argc != 3) {
               fprintf (stderr, "usage : %s [key] [type]\n", argv[0]);
               perror ("msgq");
              exit(1);
      }
      if(( qid = msgget ((key_t) atoi(argv[1]), IPC_CREAT | 0660)) == -1) {
               perror("msgget");
              exit(1);
      }
```

Sender Example (2)

`csend.c`

```
    type = (long) atoi(argv[2]);
    mymsg.mtype = type;

     while (1) {
            printf("> type message: ");
            fgets(mymsg.mtext, sizeof(mymsg.mtext), stdin);
            mymsg.mtext[strlen(mymsg.mtext)] = '\0';
            if (msgsnd(qid, &mymsg, strlen(mymsg.mtext), 0) == -1) {
                    perror("msgsnd");
                    exit(2);
            }
            if (strncmp(mymsg.mtext, "quit", 4)==0) break;
    }
}
```

Run & Results

```
$ ./csend 1234 77
> type message : Hello
...........
```

```
$ ./crecv 1234 77
At receiver, mtype = 77, mtext(6) = Hello
```

```
$ ipcs -qa
------ Shared Memory Segments ------
key         shmid     owner     perms     bytes      nattch     status

------ Semaphore Arrays ------
key         shmid     owner     perms     nsems      status

------ Message Queues ------
key            msqid  owner     perms     used-bytes    messages
0x000004d2     0      shlim     660       6                  1
```

Shared Memory

Why need a shared memory?

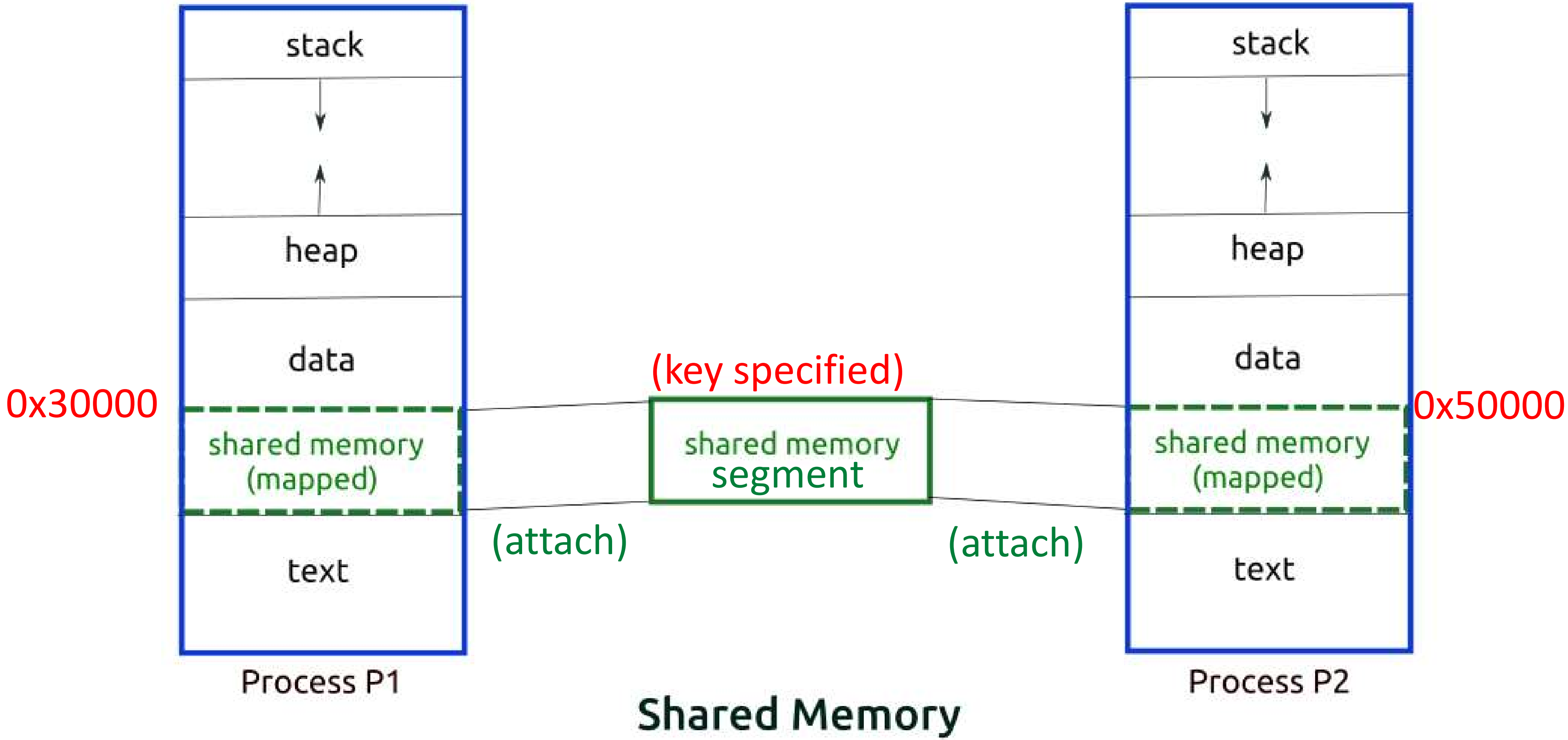
- Note
  - a process has its own address space which cannot be accessed by other processes
    - strongly enforces “process protection”
  - shared memory is a OS support memory (address space) which can be shared by difference processes.

- cf. multiple threads in a process
  - threads share code, data, and heap of the process
  - so, the information sharing b/w the threads can be easily achieved

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 to a process
  - it becomes a part of a process address space, and shared with other processes
- Collaborating processes may use semaphore to synchronize access to the shared memory segment

Shared Memory b/w Processes



shmget

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

int shmget(key_t key, int size, int flag);
```

- parameters
  - `key` : shared memory id
  - `size` : shared memory size
  - `flag` : options (same as in file open/create())
- return
  - shared memory id if OK / -1 on error
- example:

  shmId = shmget(key, size, IPC_CREAT|IPC_EXCL|0666);

shmat

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

void *shmat(int shmid, void *addr, int flag);
```

- attach (map) a shared memory segment to the address space of calling process
- parameters
  - shmid :
  - addr : 0 (recommended) →use the address selected by kernel
    - if nonzero, the given address is tried at first if possible
  - flag : 0 (default) / SHM_RDONLY (for read-only)
- return
  - address (pointer) mapped to the shared segment if OK
  - -1 on error

shmdt

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

void shmdt (void *addr);
```

- detach (unmap) a shared memory segment from calling process
- parameter
  - addr : address to detach
- return
  - 0 if OK / -1 on error

shmctl

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

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

- performs various shared memory management operations
- parameters
  - `shmid` : shared memory id
  - `cmd` : IPC_SET / IPC_STAT / IPC_RMID (same as in the message queue)
  - `buf` : address of a `shmid_ds` structure
- return
  - 0 if OK / -1 on error

shmid_ds srtuctre

```
struct shmid_ds {
   struct ipc_perm      shm_perm;
   int                  shm_segsz;
   struct anon_map      *shm_amp;
   ushort_t             shm_lkcnt;
   pid_t                shm_lpid;
   pid_t                shm_cpid;
   shmatt_t             shm_nattch;
   ulong_t              shm_cnattch;
   time_t               shm_atime;
   long                 shm_pid1;
   time_t               shm_dtime;
   long                 shm_pid2;
   time_t               shm_ctime;
   long                 shm_pid3;
   long                 shm_pid4[4];
};
```

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

Memory Layout Example (1)

memlayout.c

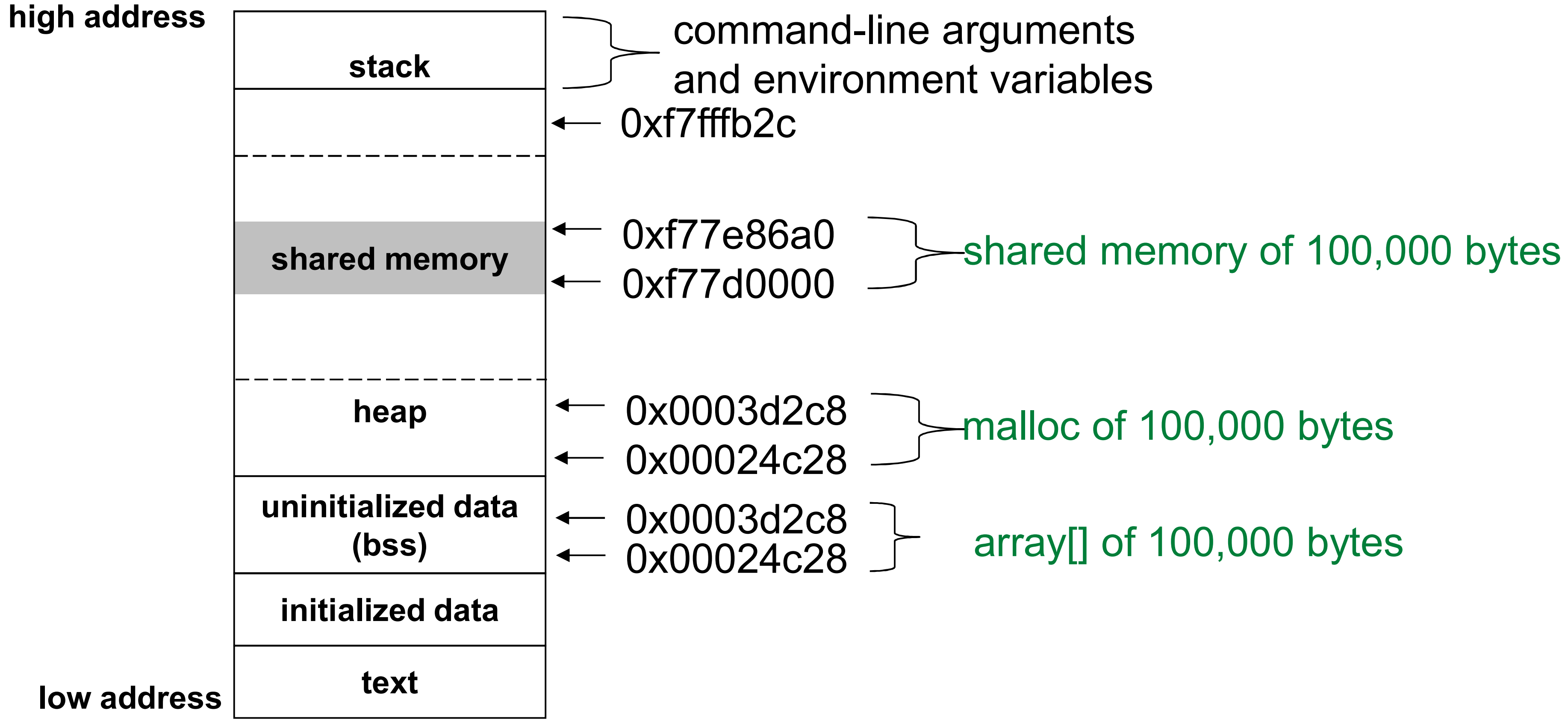
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

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

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);
```

Memory Layout Example (2)

memlayout.c

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

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

   if ((shmptr = shmat(shmid, 0, 0)) == (void *) -1)
      perror("shmat error");

   printf("shared memory attached from %x to %x \n",
                                                   shmptr, shmptr + SHM_SIZE);

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

   return 0;
}
```

Memory Layout Example (3)



위 주소는 절대적인 값이 아니며, 시스템 상태에 따라 달라질 수 있음

Producer Example (1)

shm-pro.c

```c
#include <stdio.h>
#include <semaphore.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <semaphore.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <signal.h>

int key_pressed = 0;

static void sig_handler(int signo)
{
   key_pressed++;
   printf("continue to the next stage.... \n");
   if (key_pressed==1 && signo==SIGINT)
               signal(SIGINT, sig_handler);
   else if (signo==SIGINT)
               signal(SIGINT, SIG_DFL);
}
```

Producer Example (2)

shm-pro.c

```
int main(void)
{               int shmid;
                size_t shsize = 1024;
                const int key = 16000;
                char *shm;
                sem_t *mysem;
                int i ;

                signal(SIGINT, sig_handler);

                sem_unlink("mysem"); // remove the old semaphore if any
                if((mysem = sem_open("mysem", O_EXCL | O_CREAT, 0777, 1)) == SEM_FAILED) {
                   perror("Sem Open Error");
                   exit(1);
                }

                if((shmid = shmget((size_t)key, shsize, IPC_CREAT|0666))<0) {
                   perror("shmget");
                   exit(1);
                }
```

Producer Example (3)

shm-pro.c

```
if((shm = (char*) shmat(shmid, NULL, 0)) == (char*)-1) {
        perror("shmat");
        exit(1);
}

for(int i=0; i<10; i++) {
        shm[i] = 0;
}

// write data to the shared memory
while(!key_pressed) {
        for(i=0; i<10; i++) {
                sem_wait(mysem);
                for(int i=0; i<10; i++) {
                        shm[i] = (shm[i]+1)%10;
                }
                sem_post(mysem);
                sleep(1);
        }
}
```

Producer Example (4)

shm-pro.c

```
        // read data from the shared memory
        while(key_pressed==1) {
                for(i=0; i<10; i++) {
                        sem_wait(mysem);
                        for(int i=0; i<10; i++)
                                printf("%c", (char) shm[i]);
                        sem_post(mysem);
                        sleep(1);
                        printf("\n\n");
                        fflush(stdout);
                }
        }
        sem_close(mysem);
        sem_unlink("mysem");

        shmdt(shm);
        shmctl(shmid, IPC_RMID, 0);

        return 0;
}
```

Consumer Example (1)

shm-con.c

```
// header 생략

int key_pressed = 0;

void sig_handler(int signo)
{
		key_pressed++;
		printf("continue to the next stage ...... \n");

		if (key_pressed==1 && signo==SIGINT)
			signal(SIGINT, sig_handler);
		else if (signo==SIGINT)
			signal(SIGINT, SIG_DFL);
}
```

Consumer Example (2)

shm-con.c

```
int main(void)
{
        int shmid;
        size_t shsize = 1024;
        const int key = 16000;
        char *shm;
        char c ;
        int i;
        sem_t *mysem;


        signal(SIGINT, sig_handler);

        if((mysem = sem_open("mysem", 0, 0777, 0)) == SEM_FAILED) {
                perror("Sem Open Error");
                exit(1);
        }
```

Consumer Example (3)

shm-con.c

```
if((shmid = shmget((key_t)key, shsize, IPC_CREAT | 0666)) <0) {
        perror("shmget");
        exit(1);
}

if((shm = (char*) shmat(shmid, NULL, 0)) == (char*) -1) {
        perror("shmat");
        exit(1);
}
while (!key_pressed) {
        for(i=0; i<10; i++) {
                sem_wait(mysem);
                for(int i=0; i<10; i++)
                        printf("%d", (shm[i]));
                putchar('\n');
                sem_post(mysem);
                sleep(1);
        }
}
```

Consumer Example (4)

shm-con.c

```
        while (key_pressed==1) {
                c = 'A' ;
                for(i=0; i<10; i++) {
                        sem_wait(mysem);
                        for(int i=0; i<10; i++) {
                                shm[i] = c ;
                        }
                        sem_post(mysem);
                        sleep(1);

                        if (++c > 'Z') c='A';
                }
        }
        sem_close(mysem);
        shmdt(shm);

        return 0;
}
```

Results

```
#./shm-pro
^Ccontinue to the next stage....
AAAAAAAAAA
BBBBBBBBBB
CCCCCCCCCC
DDDDDDDDDD
EEEEEEEEEE
FFFFFFFFFF
GGGGGGGGGG
HHHHHHHHHH
IIIIIIIIII
JJJJJJJJJJ
AAAAAAAAAA
BBBBBBBBBB
^CCCCCCCCCCcontinue to the next stage....
CCCCCCCCCC
DDDDDDDDDD
EEEEEEEEEE
FFFFFFFFFF
GGGGGGGGGG
```

```
#./shm-con
0000000000
1111111111
2222222222
3333333333
4444444444
5555555555
6666666666
7777777777
8888888888
9999999999
0000000000
1111111111
2222222222
3333333333
4444444444
5555555555
^Ccontinue to the next stage ......
0000000000
0000000000
^Ccontinue to the next stage ......
```

Results

```
# ipcs

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

------ Shared Memory Segments --------
key             shmid      owner      perms      bytes      nattch     status
0x00003e80 294913     shlim      666        1024       2

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
```

HW

- Fifo(named pipe) IPC를 이용하여 server/client 간의 채팅 프로그램을 작성하시오.
  - Server 및 client 동작 방식
    - 두개의 fifo를 생성(Server에서만 생성)
      - » 하나의 fifo는 server의 메시지를 client에서 보내는 용도
      - » 하나의 fifo는 client의 전송 메시지를 읽는 용도
    - Thread를 하나 생성
    - Thread
      - » 읽기 용도의 fifo를 open하고, fifo로부터 데이터를 읽음(blocking read)
      - » client의 전송 메시지를 읽어서 출력
      - » 이 과정을 무한 반복
      - » 전송받은 메시지가 “quit”인 경우 프로그램 종료
    - Main함수
      - » Fgets를 통해서 한 문장을 입력받음
      - » 입력받은 문자열(메시지)를 client에게 전송
      - » 이 과정을 무한 반복
      - » 사용자의 입력이 “quit”일 경우 “quit”를 전송하고, 쓰레드드 및 프로그램 종료

HW

- 예상 결과

```
./server
[SERVER]
[CLIENT] hello
[SERVER] hi, my name is seungho
[SERVER]
[CLIENT] hello, my name is lim
[SERVER]
[CLIENT] good day
[SERVER] bye
[SERVER] quit
Quit chatting
```

```
./client
[CLIENT] hello
[CLIENT]
[SERVER] hi, my name is seungho
[CLIENT] hello, my name is lim
[CLIENT] good day
[CLIENT]
[SERVER] bye
[CLIENT]
[SERVER] quit
[CLIENT] Quit chatting
```

- Due date
  - 6/11(목)