SWE2024

SUNG KYUN KWAN UNIVERSITY

System-V IPC

Prof. Joonwon Lee (joonwon@skku.edu)
TA – Jaehyung Park (jaeseanpark@gmail.com)
TA – Luke Albano (lukealbano@arcs.skku.edu)
Sungkyunkwan University

Review of Inter-Process Communication

- Message passing model
  - Pipe
  - Named pipe (FIFO)
  - Message queue

Week 8 (Pipe & Redirection)

- Shared memory model
  - Shared memory

System V IPC Interface

- System V IPC provides
  - Shared memory, message queue, semaphores, ...
- Key & Identifier
  - Key: identify user (process) **like file name**
    - Key can be hard coded or obtained by ftok() function
  - Identifier: uniquely identify an **IPC object**
- Permission
  - XYZ form
  - Like file permission
  - X: owner user, Y: owner group, Z: every other
  - 100_2: read (-r) 010_2: write (-w) 001_2: modify (-a)

System V IPC API

- **key_t ftok (const char* pathname, int proj_id)**
  - To create deterministic key with string
  - **pathname**: any valid file path
  - **proj_id**: some integer must be non-zero

    (only lower 8 bits are used)

```c
int main(void)
{
        key_t k1 = ftok(".", 'a');
        printf("key1:\t%d\n", k1);
        key_t k2 = ftok(".", 'a' + 1);
        printf("key2:\t%d\n", k2);

        return 0;
}
```

```
key1:   1627521876
key2:   1644299092
```

```
NAME
       ftok - convert a pathname and a project identifier to a System V IPC key

SYNOPSIS
       #include <sys/types.h>
       #include <sys/ipc.h>

       key_t ftok(const char *pathname, int proj_id);

DESCRIPTION
       The  ftok() function uses the identity of the file named by the given pathname (which must refer to an existing, accessible file) and the
       least significant 8 bits of proj_id (which must be nonzero) to generate a key_t type System V IPC key, suitable for use  with  msgget(2),
       semget(2), or shmget(2).

       The  resulting  value  is the same for all pathnames that name the same file, when the same value of proj_id is used.  The value returned
       should be different when the (simultaneously existing) files or the project IDs differ.

RETURN VALUE
       On success, the generated key_t value is returned.  On failure -1 is returned, with errno indicating the error as for the stat(2)  system
       call.
```

System V IPC Interface Data Structure

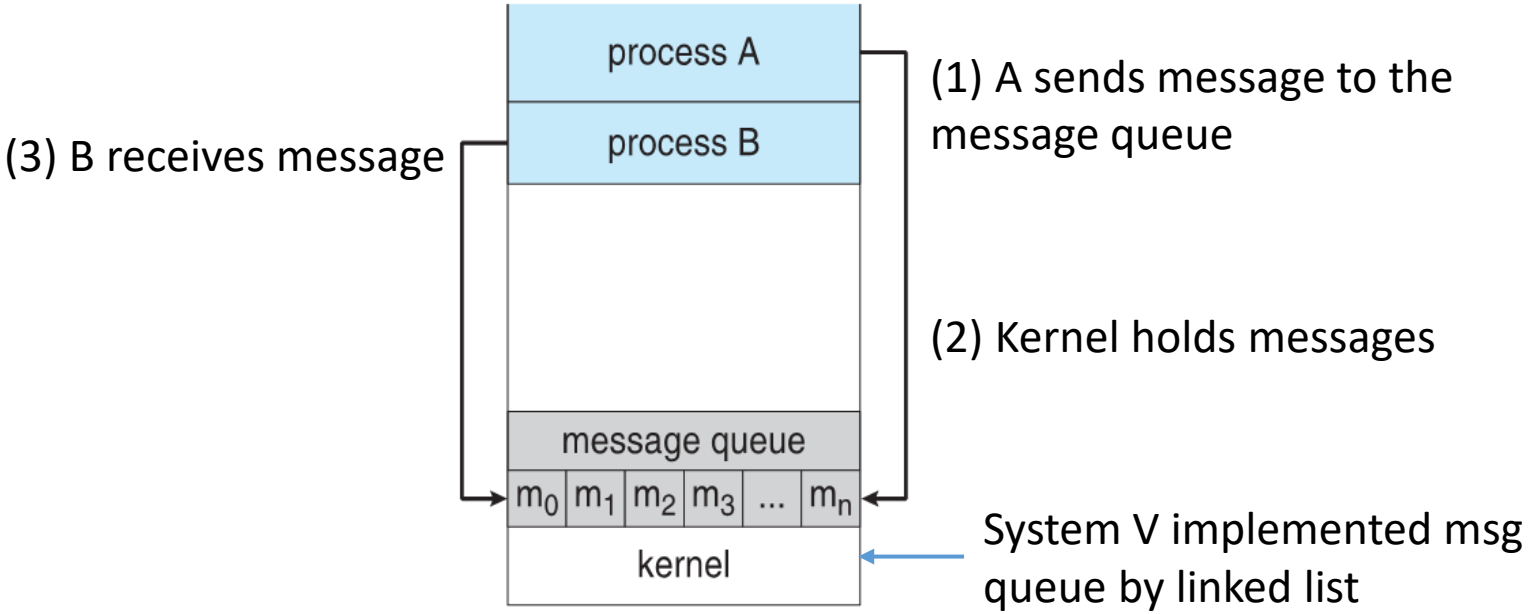
- **ipc_perm**
  - Data structure designed to handle permission
  - Component: user and group ID, mode bit, and sequence number

```
/* defined <sys/ipc.h> */

struct ipc_perm {
        key_t           key;    /* user specified msg/sem/shm key */
        uid_t           cuid;   /* creator user id */
        gid_t           cgid;   /* creator group id */
        uid_t           uid;    /* user id */
        gid_t           gid;    /* group id       */
        mode_t          mode;   /* r/w permission */
        unsigned short  seq;    /* sequence #     */
};
```

Message Queue

- Mail-box like communication method
  - Pros: easy to use
  - Cons: overhead if data is large

- Typical message passing model



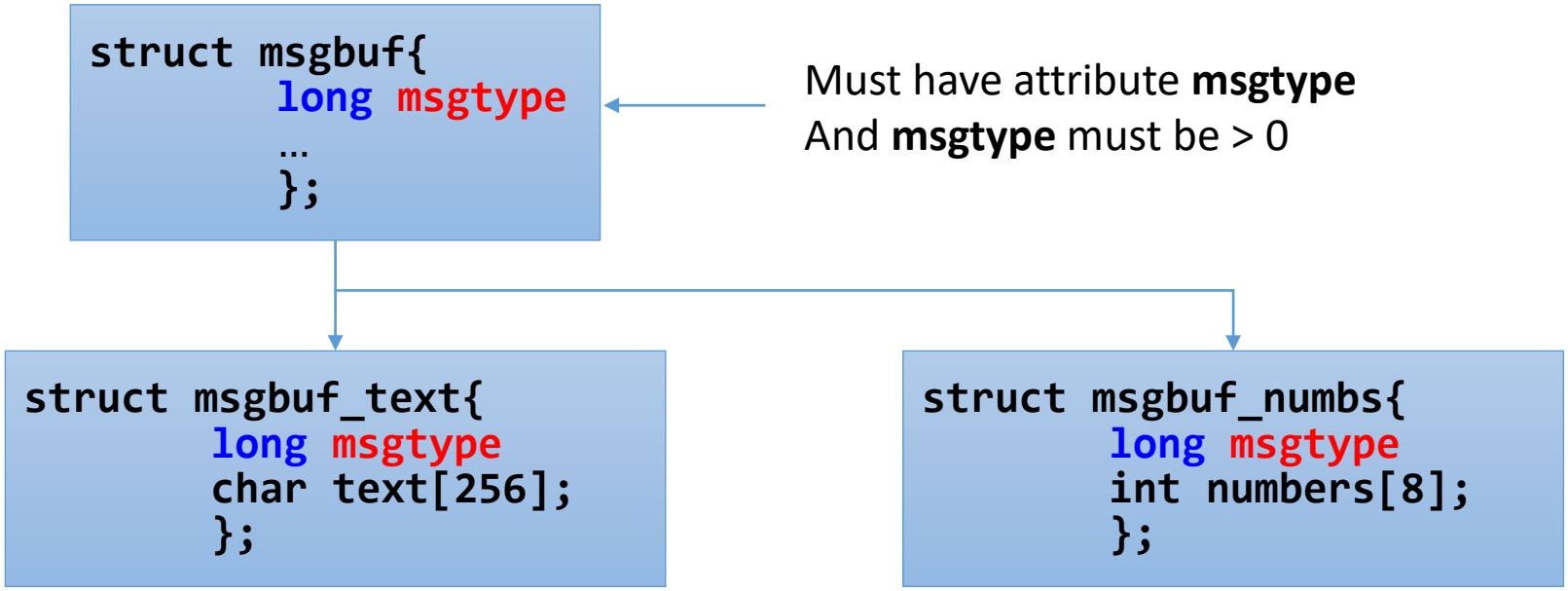
Message Queue Data Structure

- Defined in <sys/msg.h>

```
struct msqid_ds {
        struct ipc_perm msg_perm;       /* permission    */
as a list
        struct msg      *msg_first;     /* first message */
        struct msg      *msg_last;      /* last message  */
        unsigned long   msg_cbytes;     /* number of bytes in use */
resource
        unsigned long   msg_qnum;       /* number of msgs */
        unsigned long   msg_qbytes;     /* max # of bytes */
        pid_t           msg_lspid;      /* pid of last sender() */
        pid_t           msg_lrpid;      /* pid of last receiver() */
access
        time_t          msg_stime;      /* time of last send() */
        time_t          msg_rtime;      /* time of last receive() */
        time_t          msg_ctime;      /* time of last msgctl() */
        };
```

Message Structure

- **msgbuf** should be implemented
  - The first attribute of **msgbuf** is **msgtype**
  - Sender and receiver should share the same **msgbuf** implementation



Message Queue API

- **int msgget (key_t key, int msgflg)**
  - e.g., msgget (0x1234, IPC_CREAT)

- **int msgsnd (int msgid, const void *msgp, size_t msgsz, int msgflg)**
  - e.g., msgsnd (0x1234, (void *)buf, sizeof(struct msgbuf), 0)

- **ssize_t msgrcv (int msgid, void *msgp,**
  size_t msgsz, long msgtyp, int msgflg)
  - e.g., msgrcv (0x1234, (void *)buf, sizeof(struct msgbuf), 1, 0)

- **int msgctl (int msgid, int cmd, struct msqid_ds *buf)**
  - e.g., msgctl (0x1234, IPC_RMID, NULL) /* remove msgq */

System V IPC Flags

- IPC_CREAT
  - Like O_CREAT
  - Create object if there is no object created by given key
- IPC_EXCL
  - Like O_EXCL
  - Return error if object is already exists
- IPC_SET
  - Set data of the object with given ID
  - Parameter of **ctl()** function
    - e.g., **msgctl(0x1234, IPC_SET, ptr)**

System V IPC Flags

- IPC_RMID
  - To remove object with given ID
  - Parameter of **ctl()** function
    - e.g., **msgctl(0x1234, IPC_RMID, NULL)**
- IPC_STAT
  - Get data of the object with given ID
  - Return object specific data structure
    - e.g., **msgqid_ds**, **shmid_ds**
  - Parameter of **ctl()** function
    - e.g., **msgctl(0x1234, IPC_STAT, ptr)**
- IPC_NOWAIT
  - Return immediately if no message of the requested type is in the queue. The system call fails with errno set to ENOMSG.

Message Queue Sender Example

```
int main(void)
{
        key_t k = ftok(".", 'a');              // get key
        int qid = msgget(k, IPC_CREAT | 0666); // get msg queue object
        if (qid == -1)  {
                perror("msgget error : ");
                exit(1);
        }

        struct msgbuf *msg = malloc(sizeof(struct msgbuf));
        msg->msgtype = 1;                      // set msgtype 1
        strcpy(msg->text, "Hello world?\n");   // write message

        /* send */
        if (msgsnd(qid, (void *)msg, sizeof(struct msgbuf), 0) == -1) {
                perror("msgsnd error : ");
                exit(1);
        }

        free(msg);
        return 0;
}
```

```
struct msgbuf {
        long msgtype;
        char text[256];
};
```

Message Queue Receiver Example

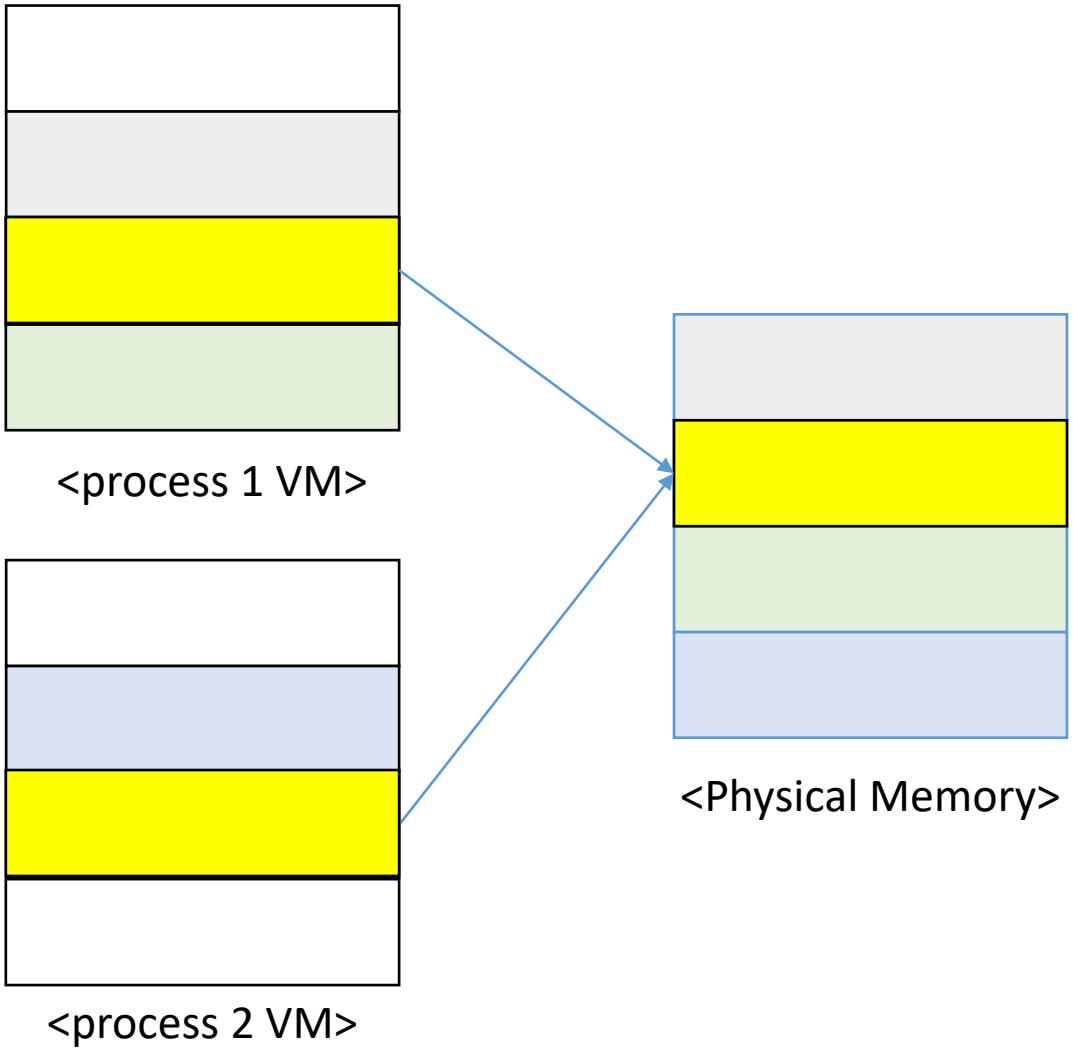
```
int main(void)
{
        key_t k = ftok(".", 'a');               // get key
        int qid = msgget(k, IPC_EXCL | 0666);  // get msg queue object
        if (qid == -1)  {
                perror("msgget error : ");
                exit(1);
        }

        struct msgbuf *msg = malloc(sizeof(struct msgbuf));

        /* receive */
        if (msgrcv(qid, (void *)msg,
                   sizeof(struct msgbuf), 1, 0) == -1) {
                perror("msgrsv error : ");
                exit(1);
        }

        printf("%s", msg->text);
        free(msg);
        return 0;
}
```

```
struct msgbuf {
        long msgtype;
        char text[256];
};
```

Shared Memory Model

- Mapping segment of memory that will be shared by more than one process
- Fastest form of IPC
  - There is no intermediation
- Shared memory can be written to and read from by any number of process
  - Pros: fast
  - Cons: synchronization

Shared Memory Abstract



Shared Memory API

- **int shmget (key_t key, size_t size, int shmflag)**
  - e.g., **shmget (0x1234, sizeof(int) * 100, IPC_CREAT)**
  - Use flags start with SHM_HUGE to allocate large size memory

- **void *shmat (int shmid,**
  const void *shmaddr, int shmflg)
  - e.g., **shmat(1, NULL, 0)**
  - **shmaddr**: usually NULL
  - **shmflag**: set 0 for READ WRITE permission

Shared Memory API (Cont.)

- **int shmdt (int shmid,**

  const void * shmaddr, int shmflag)
  - e.g., **shmdt (1, shmaddr, 0)**
  - **shmaddr**: address where shared memory segment is attached

- **void shmctl (int shmid,**

  int cmd, struct shmid_ds *buf)
  - e.g., **shmctl(1, IPC_RMID, 0)** /* remove ID 1 */
  - See common IPC flags

Shared Memory Data Structure

- Defined in <sys/shm.h>

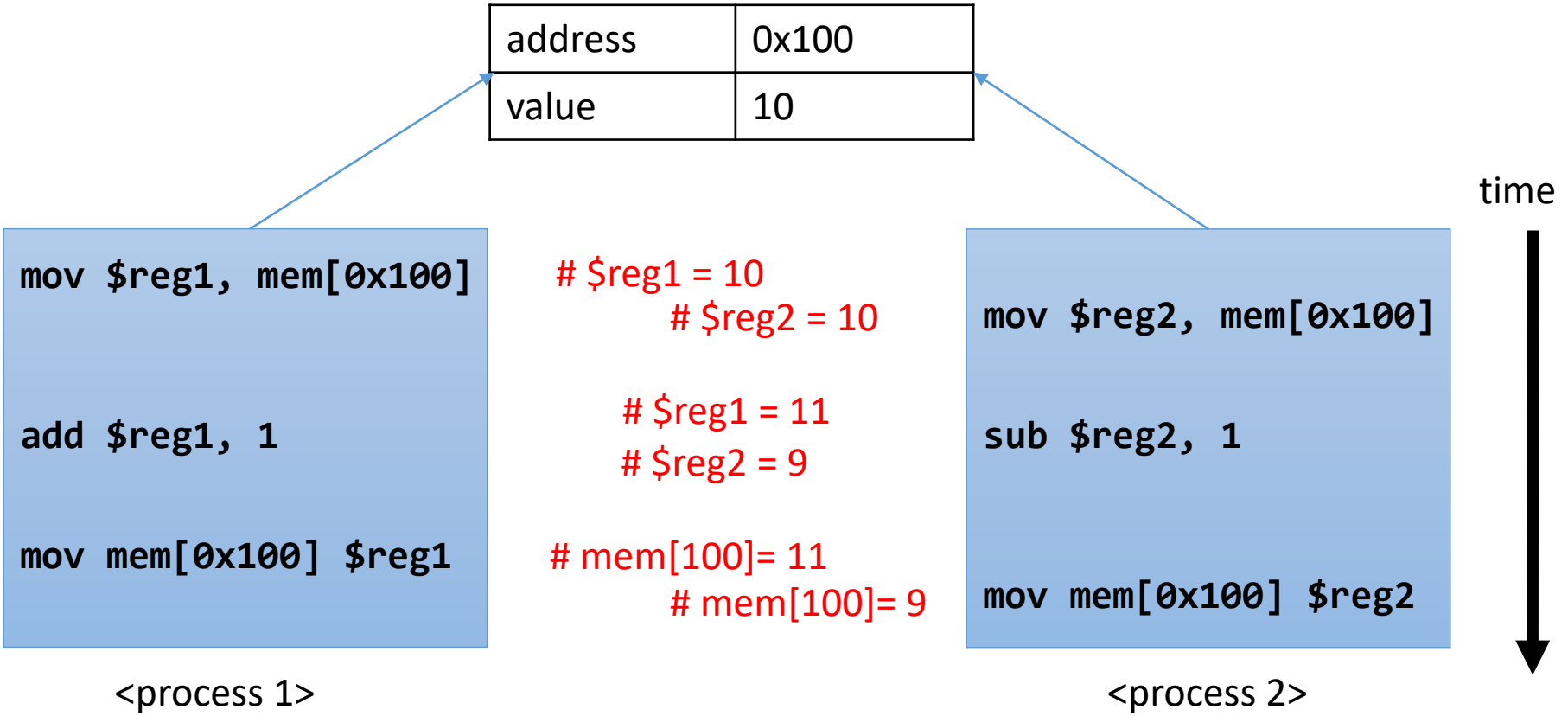
```
struct shmid_ds {
        struct ipc_perm         shm_perm;       /* operation perms */
        int                     shm_segsz;      /* size of seg (bytes) */
        time_t                  shm_atime;      /* last attach time */
        time_t                  shm_dtime;      /* last detach time */
        time_t                  shm_ctime;      /* last change time */
        pid_t                   shm_cpid;       /* pid of creator */
        pid_t                   shm_lpid;       /* pid of last operator */
        unsigned short          shm_nattch;     /* no. of curr attaches */

/* ------------------------- unused ------------------------- */
        unsigned short          shm_unused;
        void                    *shm_unused2;
        void                    *shm_unused3;
};
```

Shared Memory Example

```
int main(void)
{
        key_t k = ftok(".", 'b');
        int shm_id = shmget(k, sizeof(int), IPC_CREAT | 0666);
        if (shm_id < 0)  {
                perror("shmget fail");
                exit(0);
        }

        int *shmaddr = shmat(shm_id, NULL, 0);
        *shmaddr = 1;
        printf("value before fork: %d\n", *shmaddr);

        if (fork() == 0) {
                (*shmaddr) += 1;
                printf("value in the child: %d\n", *shmaddr);
                exit(0);
        } else {
                wait(NULL);
                printf("value in the fork: %d\n", *shmaddr);
        }

        return 0;
}
```

```
jaehyun@csl:~/SSE/week9$ ./shm
value before fork: 1
value in the child: 2
value in the fork: 2
```

Race Condition



Race Condition Example

```c
int main(void)
{
        key_t k = ftok(".", 'b');
        int shm_id = shmget(k, sizeof(int), IPC_CREAT | 0666);
        if (shm_id < 0)  {
                perror("shmget fail");
                exit(0);
        }

        int *shmaddr = shmat(shm_id, NULL, 0);
        *shmaddr = 1;
        printf("value before fork: %d\n", *shmaddr);

        if (fork() == 0) {
                for (int i = 0; i < 1000000; i++) /* Child add 1M */
                        (*shmaddr) += 1;
                exit(0);
        } else {
                for (int i = 0; i < 1000000; i++) /* Parent sub 1M */
                        (*shmaddr) -= 1;
        }
        printf("value after operation: %d\n", *shmaddr); /* Must 1 */
        return 0;
}
```

```
jaehyun@csl:~/SSE/week9$ ./race
value before fork: 1
value after operation: 519688
jaehyun@csl:~/SSE/week9$ ./race
value before fork: 1
value after operation: -813183
jaehyun@csl:~/SSE/week9$ ./race
value before fork: 1
value after operation: -999818
jaehyun@csl:~/SSE/week9$ ./race
value before fork: 1
value after operation: -814614
```

Race Condition Solution Example

```c
int main(void)
{
        key_t k = ftok(".", 'b');
        int shm_id = shmget(k, sizeof(int), IPC_CREAT | 0666);
        if (shm_id < 0)  {
                perror("shmget fail");
                exit(0);
        }

        int *shmaddr = shmat(shm_id, NULL, 0);
        *shmaddr = 1;
        printf("value before fork: %d\n", *shmaddr);

        if (fork() == 0) {
                for (int i = 0; i < 1000000; i++) /* Child add 1M */
                        (*shmaddr) += 1;
                exit(0);
        } else {
                wait(NULL);
                for (int i = 0; i < 1000000; i++) /* Parent sub 1M */
                        (*shmaddr) -= 1;
        }
        printf("value after operation: %d\n", *shmaddr); /* Must 1 */
        return 0;
}
```

```
jaehyun@csl:~/SSE/week9$ ./race_
value before fork: 1
value after operation: 1
jaehyun@csl:~/SSE/week9$ ./race_
value before fork: 1
value after operation: 1
jaehyun@csl:~/SSE/week9$ ./race_
value before fork: 1
value after operation: 1
jaehyun@csl:~/SSE/week9$ ./race_
value before fork: 1
value after operation: 1
```

Shell Commands

- **ipcs (-i, -m, -q, -a)**
  - List IPC objects
  - **-i <ID>** : show details about given ID
  - **-m** : show shared memory
  - **-q** : show message queue
  - **-a** : show all

```
❯ ipcs

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x6120b818 0          jaepark    666        0            0

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status

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

- **ipcrm (-M, -m, -Q, -q, -a)**
  - Remove IPC objects
  - **-m <ID>** : remove shared memory given ID (-M <key>)
  - **-q <ID>** : remove message queue given ID (-Q <key>)
  - **-a** : remove all IPC objects

Exercise

- Make chatting program
  - When program starts, get ID and receiver's ID by stdin
  - User can receive message while entering message
  - When the user get the message from the other user (normal message), print out the message
  - When the receiver (the other user) check the message, send back ack message with read time
  - There should be no wait between normal message and ack message (add IPC_NOWAIT flags to the **msgrcv()**)
  - To finish program, enter "quit"

Exercise

- Print format
  - RECEIVED $USER_MSG /* normal message */
  - $ID read message at:\t$READ_TIME /* ack message */
  - Use **ctime()** function to match $READ_TIME format

```
❯ ./p9
My id is: 10
Enter the receiver of your message: 20
How was your exam last week?
20 read message at:     Mon Apr 24 16:07:12 2023

RECEIVED The exams were easy. Expecting straight A's.

Good for you...
20 read message at:     Mon Apr 24 16:08:15 2023

quit
```

```
❯ ./p9
My id is: 20
Enter the receiver of your message: 10
RECEIVED How was your exam last week?

The exams were easy. Expecting straight A's.
10 read message at:     Mon Apr 24 16:08:04 2023

RECEIVED Good for you...

quit
```

  - Copy the skeleton code to your directory

$ cp ~swe2024-41_23s/2023s/p9_skeleton.c ./

Skeleton code of p9.c

- copy the skeleton code to your directory

```
$ cp ~swe2024-41_23s/2023s/p9_skeleton.c ./
```

```c
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/msg.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>

#define T_MSG 20

typedef struct {
    long msgtype;
    int sender;
    char text[512];
} msgbuf;

typedef struct {
    long msgtype;
    char timestamp[512];
} msgbuf_ack;

int main() {
    key_t key = ftok(".", 'a');
    int user_id, receiver_id, qid, pid;
    printf("My id is: ");
    scanf("%d", &user_id);
    getchar();
    printf("Enter the receiver of your message: ");
    scanf("%d", &receiver_id);
    getchar();
    if ((qid = msgget(key, IPC_CREAT | 0666) < 0)) {
        perror("msgget failed\n");
        exit(1);
    }
    if ((pid = fork()) == 0) {
        while (1) {
            /* child = receives messages
                1. receive normal message using buf
                1-1. print "RECEIVED [normal message]"
                1-2. send ack message using ack (current timestamp)
                    --> use T_MSG + receiver_id for ack.msgtype
                2. receive ack message using read_time
                2-1. print "[receiver_id] read message at:\t[timestamp]" */
            msgbuf buf;
            msgbuf_ack read_time, ack;
            /* your code */
        }

    } else {
        while (1) {
            /* parent = sends messages
                1. get normal message from stdin
                2. if "quit", exit both parent and child (use SIGINT)
                3. send normal message */
            msgbuf buf;
            buf.msgtype = receiver_id;
        }
    }
}
```

Exercise

- Submit your exercise source code
  - To InUiYeJi Cluster
  - Put your Makefile and *.c files in p9 folder
  - Submit using

  $ ~swe2024-41_23s/bin/submit p9 p9
  - We will compile by using command *make*
    - When compilation fails, you get zero points
    - Compiled binary name should be *"p9"*

- Due 2023/4/26 23:59