Lab 2
Interprocess Communication

Guideline

- Read text materials and practice IPC processing
- Use your notebook PC to examine context and produce the source code when you finish the successful practice.

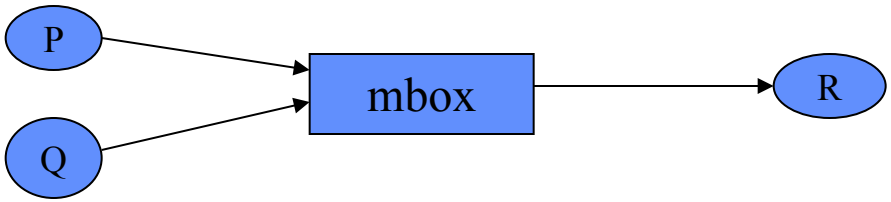
Two types of IPC in LAB

1. Message Queue

2. Shared Memory

Mailboxes (Message Queue)

- Also known as message queues, ports
- The explicit and symmetric naming of processes in direct naming
- Þ Limited modularity since changing the name of a process requires changes elsewhere, i.e., in definitions of other processes



P or Q call

send(mbox-id, message)

R calls

receive(mbox-id, message)

System Call: msgget()

- The msgget() function returns the message queue identifier associated with key. A message queue identifier and an associated message queue and data structure are created if key is equal to IPC_PRIVATE, or key does not already have a message queue identifier associated with it and (msgflg & IPC_CREAT) is non-zero.

Function	**int msgget (key_t key, int msgflg);**
Function arguments	
key	Specifies the message queue key for which to retrieve the msqid.
msgflg	Is a flag that indicates specific message queue conditions and options to implement.
Return values	
If successful, msgget() returns a message queue identifier. On failure, msgget() returns a value of -1 and sets errno to indicate the error.	

System Call: msgsnd()

- The msgsnd() function sends a message to the queue associated with message queue identifier msqid.

Function	**int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);**
Function arguments	
msqid	Is a unique positive integer, created by msgget(), that identifies a message queue and its associated data structure.
msgp	Points to a user-defined buffer.
msgsz	Is the length of the message to be sent.
msgflg	Specifies the action to be taken if one or more of the following are true
Return values	
If successful, msgsnd() returns a message queue identifier. On failure, msgsnd() returns a value of -1 and sets errno to indicate the error	

System Call: msgrcv()

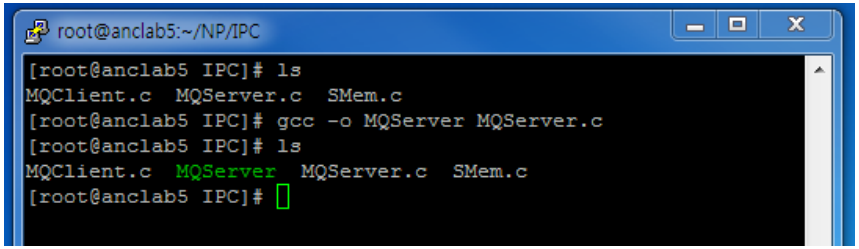
- The msgrcv() function reads a message from the queue associated with the message queue identifier that msqid specifies and places it in the user-defined structure that msgp points to.

Function	**int msgrcv(int msqid, void *msgp, int msgsz, long msgtyp, int msgflg);**
Function arguments	
msqid	Is a unique positive integer, created by a msgget() call, that identifies a message queue
msgflg	Points to a user-defined buffer
msgsz	Specifies the size, in bytes, of mtext
msgtyp	Specifies the type of message requested (3 types: -, 0, +)
Return values	
If successful, msgrcv() returns the number of bytes actually placed into mtext. On failure, msgrcv() returns a value of -1, receives no message, and sets errno.	

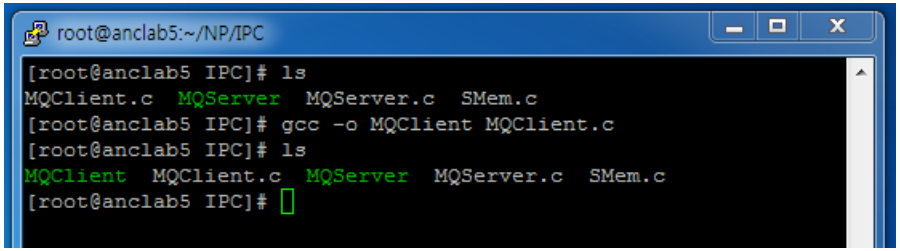
Practice1: Message Queue in Linux

- Compile

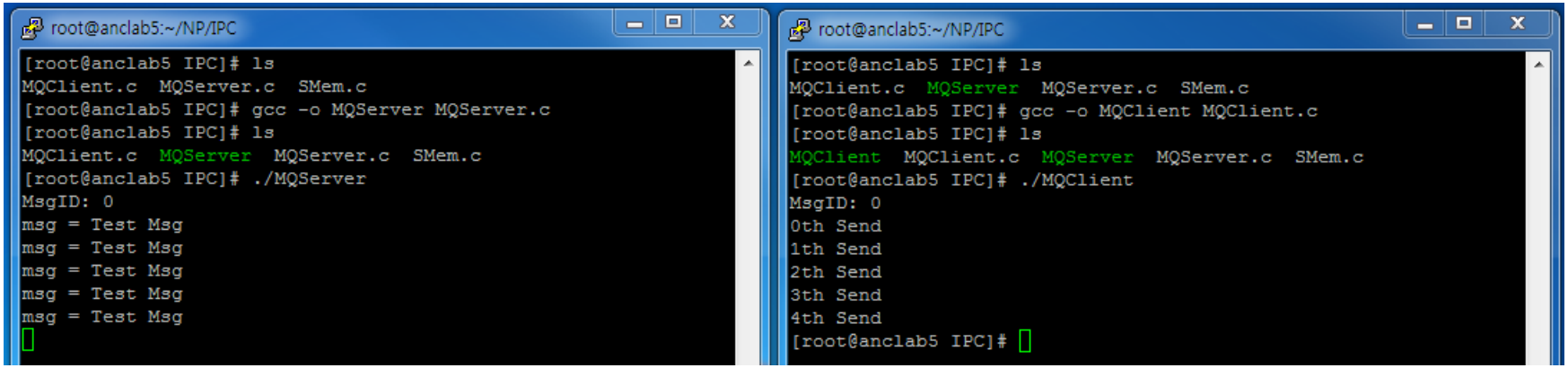
< MQServer >

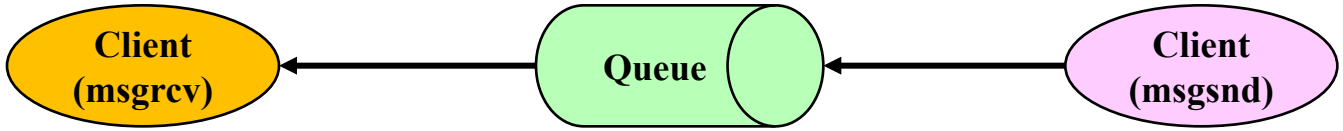


< MQClient >



- MQServer and MQClient





Server Process

```
int MsgQInit(int key){
    int msgID;
    if(msgID = msgget(key, IPC_CREAT|0666)==-1){
        perror("msgget error");
        return -1;
    }
return msgID;
}

int MsgQRcv(int msgID, void* buf, int size, int type){
    Msgbuf rcvbuf;
    int len;
    len = msgrcv(msgID, &rcvbuf, MAX_LEN, type, MSG_NOERROR |
        IPC_NOWAIT);
        if(len == -1){
            perror("msgrcv");
            return -1;
        }
    memcpy(buf, rcvbuf.msgtxt, size);
    return len;
}

int main(){// Server
    Msgbuf buf;
    int MsgID;
    int MsgType;
    int ret;
    MsgID = MsgQInt(788);
    while(1){
        ret = MsgQRcv(MsgID, buf.msgtxt, MAX_LEN, buf.type);
        if(buf.type == 3){
            printf("msg = %s \n", buf.msgtxt);
        }
    return 0;
}
```

Client Process

```
int MsgQRcv(int msgID, void* buf, int size, int type){
    Msgbuf rcvbuf;
    int len;
    len = msgrcv(msgID, &rcvbuf, MAX_LEN, type,
        MSG_NOERROR | IPC_NOWAIT);
        if(len == -1){
            perror("msgrcv");
            return -1;
        }
    memcpy(buf, rcvbuf.msgtxt, size);
    return len;
}

int MsgQSnd(int msgID, void* buf, int size, int type){
    Msgbuf  sndbuf;
    sndbuf.type = type;
    memcpy(sndbuf.msgtxt, buf, size);
    if(msgsnd(msgID, &sndbuf, size, 0){
        perror("MsgASnd");
        return -1;
    }
    return 0;
}

int main(){ // Client
    int MsgID;
    int MsgType;
    int ret;
    MsgID = MsgQInit(788);
    for(int i =0; i<5; i++){
        MsgQSnd(MsgID, "Test Msg", 9, 3);
    }
}
```

Type 1: Message Queue

root@anclab5:~/NP/IPC

Server
(Message Queue Receiver)

```c
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/wait.h>
#define MAX_LEN 255

typedef struct
{
long type;
char msgtxt[MAX_LEN];
}Msgbuf;

int MsgQInit(int key){
  int msgID;
  if(msgID = msgget(key, IPC_CREAT|0666)== -1){
    perror("msgget error");
    return -1;
  }
  return msgID;
}

int MsgQRcv(int msgID, void* buf, int size, int type){
  Msgbuf rcvbuf;
  int len;
  len = msgrcv(msgID, &rcvbuf, MAX_LEN, type, MSG_NOERROR | IPC_NOWAIT);
  if(len == -1) return -1;
  memcpy(buf, rcvbuf.msgtxt, size);
  return len;
}


int main(){
  Msgbuf buf;
  int MsgID;
  int MsgType;
  int ret;

  MsgID = MsgQInit(788);
  printf("MsgID: %d\n", MsgID);
  buf.type = 3;
  while(1){
    ret = MsgQRcv(MsgID, buf.msgtxt, MAX_LEN, buf.type);
    if(ret != -1){
      if(buf.type == 3){
        printf("msg = %s \n", buf.msgtxt);
      }
    }
  }
  return 0;
}
```

5,0-1 Top

root@anclab5:~/NP/IPC

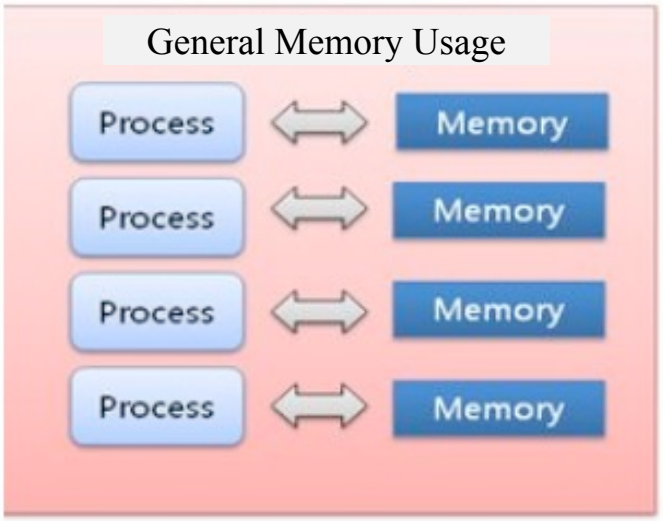
Client
(Message Queue Sender)

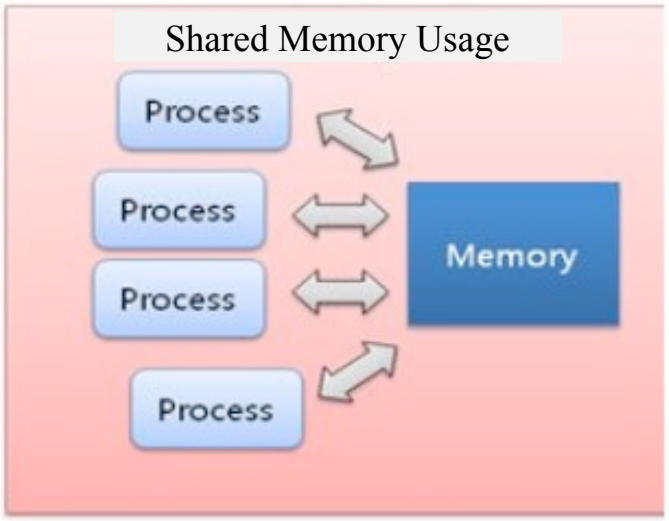
```c
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/wait.h>
#define MAX_LEN 255

typedef struct
{
  long type;
  char msgtxt[MAX_LEN];
}Msgbuf;

int MsgQInit(int key){
  int msgID;
  if(msgID = msgget(key, IPC_CREAT|0666)==-1){
    perror("msgget error");
    return -1;
  }
  return msgID;
}


int MsgQSnd(int msgID, void* buf, int size, int type){
  Msgbuf  sndbuf;
  sndbuf.type = type;
  memcpy(sndbuf.msgtxt, buf, size);

  if(msgsnd(msgID, &sndbuf, size, 0)){
    perror("MsgASnd");
    return -1;
  }
  return 0;
}


int main(){
  int MsgID;
  int MsgType;
  int ret;
  MsgID = MsgQInit(788);
  printf("MsgID: %d\n", MsgID);
  int i;
  for(i=0; i<5; i++){
    printf("%dth Send\n", i);
    MsgQSnd(MsgID, "Test Msg",9,3);
  }
}
~
~
~
```

46,1 All

Shared Memory

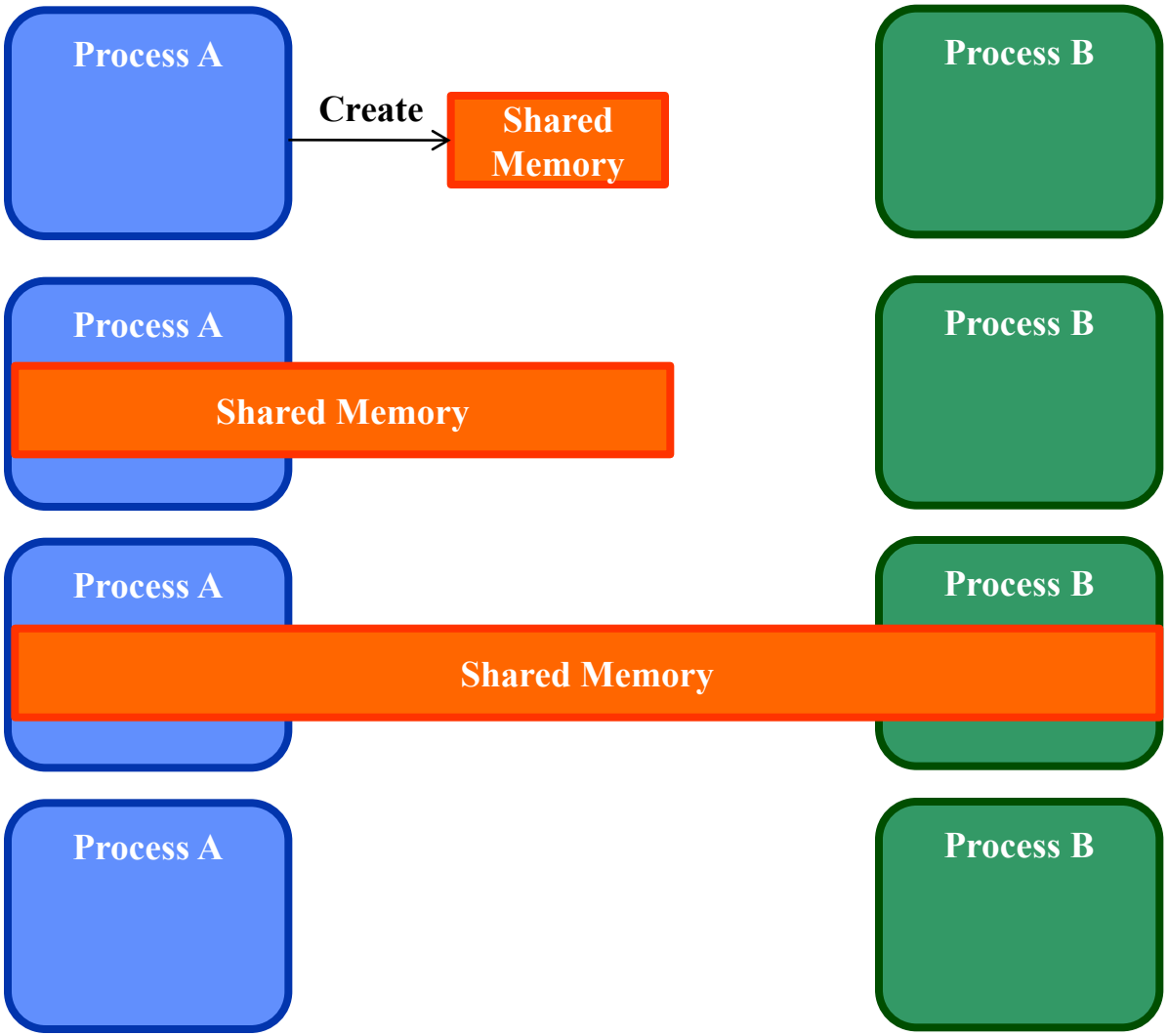
- Processes can share the same segment of memory directly when it is mapped into the address space of each sharing process
- Faster communication
- Mutual exclusion must be provided by processes using the shared memory





Shared Memory

- Generation of the shared memory: shmget()
- Access to the shared memory: shmat()
- Finishing access of the shared memory: shmdt()
- Controlling the shared memory: shmctl()



Struct for the shared memory

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

Parameters	Description
shm_perm	This is an instance of the ipc_perm structure, which is defined for us in linux/ipc.h. This holds the permission information for the segment, including the access permissions, and information about the creator of the segment (uid, etc).
shm_segsz	Size of the segment (measured in bytes).
shm_atime	Time the last process attached the segment.
shm_dtime	Time the last process detached the segment.
shm_ctime	Time of the last change to this structure (mode change, etc).
shm_cpid	The PID of the creating process.
shm_lpid	The PID of the last process to operate on the segment.
shm_nattch	Number of processes currently attached to the segment

System calls for Shared Memory

- *int shmget(key_t key, size_t size, int shmflg) :* creates a new region of shared memory or returns an existing one
- *void *shmat(int shmid, const void *shmaddr, int shmflg) :* attaches a shared memory region to the virtual address space of the process
- *int shmdt(char *shmaddr):*detaches a shared region

System Call: shmget()

- The shmget() function returns the shared memory identifier associated with key. A shared memory identifier and shared memory segment of at least size bytes is created if key is equal to IPC_PRIVATE, or if key does not already have a shared memory identifier associated with it and if (shmflg & IPC_CREAT) is non-zero.

Function	**int shmget (key_t key, int size, int shmflg);**
Function arguments	
key	Specifies either IPC_PRIVATE or a unique key.
size	Is the size in bytes of the shared memory segment.
shmflg	Specifies both the creation and permission bits (for example, IPC_CREAT | 0666).
Return values	
If successful, shmget() returns a shared memory identifier. On failure, it returns -1 and sets errno.	

System Call: shmat()

- The shmat() function attaches the shared memory segment associated with the shared memory identifier *shmid* to the data segment of the calling process.

Function	**void *shmat(int *shmid*, const void **shmaddr*, int *shmflg*);**
Function arguments	
shmid	Is a unique positive integer created by a shmget() system call and associated with a segment of shared memory.
shmaddr	Points to the desired address of the shared memory segment.
shmflg	Specifies a set of flags that indicate the specific shared memory conditions and options to implement.
Return values	
If successful, shmat() returns the data segment start address of the attached shared memory segment. On failure, it returns -1, does not attach the shared memory segment, and sets errno	

System Call: shmdt()

- The shmdt() function detaches from the calling process's data segment the shared memory segment located at the address specified by shmaddr.

Function	**int shmdt(const void *_shmaddr_);**
Function arguments	
shmaddr	Is the data segment start address of a shared memory segment.
Return values	
If successful, shmdt() decrements the shm_nattach associated with the shared memory segment and returns zero. On failure, it returns -1 and set errno.	

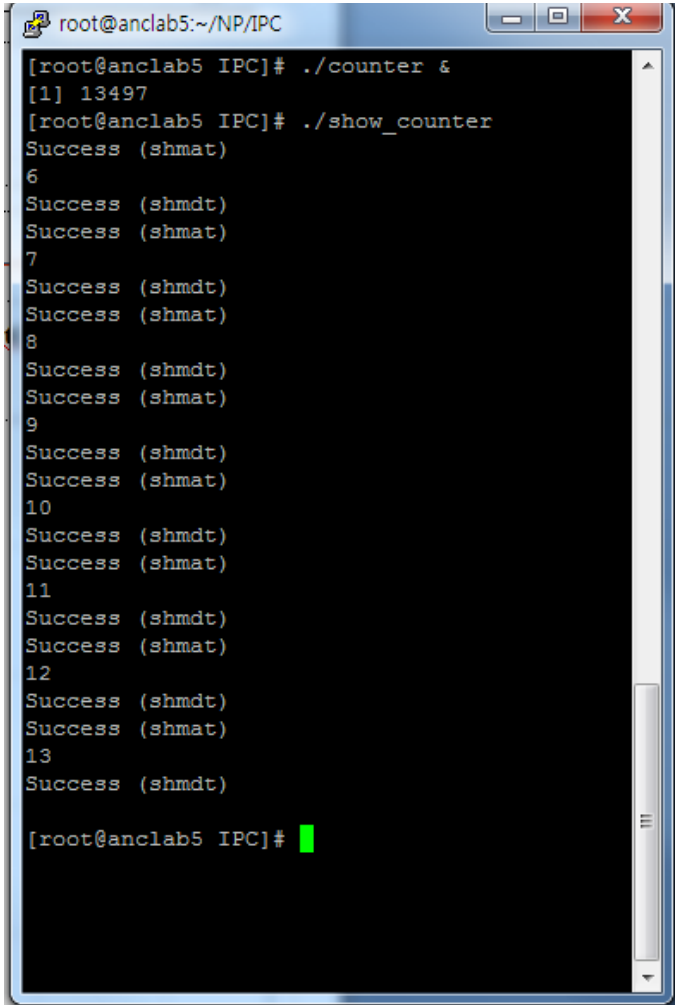
System Call: shmat()

- The shmctl() function provides a variety of shared memory control operations as specified by cmd.

Function	**int shmctl(int *shmid*, int *cmd*, struct shmid_ds **buf*);**
Function arguments	
shmid	Is a unique positive integer returned by the shmget() function and associated with a segment of memory and a data structure.
cmd	Specifies one of IPC_STAT, IPC_SET, or IPC_RMID.
buf	Points to the data structure used for sending or receiving data during execution of shared memory control operations.
Return values	
If successful, shmctl() returns zero. On failure, it returns -1 and sets errno.	

Practice2: Shared Memory in Linux

- Two processes
- One shared memory
- counter.c
  - Add 1 every one minutes to the shared data in the shared memory
- Show_counter.c
  - Show the valued stored in the shared data in the shared memory



counter.c

```
#include <stdio.h>      // printf()
#include <unistd.h>     // sleep()
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#define  KEY_NUM     9527
#define  MEM_SIZE    1024

int main( void){
  int   shm_id;
  void *shm_addr;

  if ( -1 == ( shm_id = shmget( (key_t)KEY_NUM,
       MEM_SIZE, IPC_CREAT|0666)))   {
    printf( "shmget Error\n");
    return -1;
  }
```

```
while( 1 )   {
    if ( ( void *)-1 == ( shm_addr = shmat( shm_id, ( void *)0,
      0)))  {
      printf( "shmat Error\n");
      return -1;
    }
    else  {
      printf( "Success (shmat)\n");
    }
    printf( "%s\n", (char *)shm_addr);
    if ( -1 == shmdt( shm_addr))  {
      printf( "shmdt Error\n");
      return -1;
    }
    else  {
      printf( "Success (shmdt)\n");
    }
    sleep( 1);
  }
  return 0;
}
```

show_counter.c

```
#include <stdio.h>      // printf()
#include <unistd.h>     // sleep()
#include <sys/ipc.h>
#include <sys/shm.h>
#define  KEY_NUM     9527
#define  MEM_SIZE    1024

int main( void)
{
  int   shm_id;
  void *shm_addr;
  int   count;

  if ( -1 == ( shm_id = shmget( (key_t)KEY_NUM,
       MEM_SIZE, IPC_CREAT|0666))) {
    printf( "shmget Error\n");
    return -1;
  }

  if ( ( void *)-1 == ( shm_addr = shmat( shm_id, (
       void *)0, 0))) {
    printf( "shmat Error\n");
    return -1;
  }

  count = 0;
  while( 1 ) {
    sprintf( (char *)shm_addr, "%d", count++);
    sleep( 1);
  }
  return 0;
}
```

root@anclab5:~/NP/IPC

show_counter.c

```c
// show_counter.c
#include <stdio.h>      // printf()
#include <unistd.h>     // sleep()
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#define  KEY_NUM     9527
#define  MEM_SIZE    1024

int main( void){
   int   shm_id;
   void *shm_addr;

   if ( -1 == ( shm_id = shmget( (key_t)KEY_NUM, MEM_SIZE, IPC_CREAT|0666)))
   {
      printf( "shmget Error\n");
      return -1;
   }

   while( 1 ) {
      if ( ( void *)-1 == ( shm_addr = shmat( shm_id, ( void *)0, 0))) {
         printf( "shmat Error\n");
         return -1;
      }
      else {
         printf( "Success (shmat)\n");
      }

      printf( "%s\n", (char *)shm_addr);
      if ( -1 == shmdt( shm_addr))  {
         printf( "shmdt Error\n");
         return -1;
      }
      else  {
         printf( "Success (shmdt)\n");
      }
      sleep( 1);
   }
   return 0;
}
```

1,7

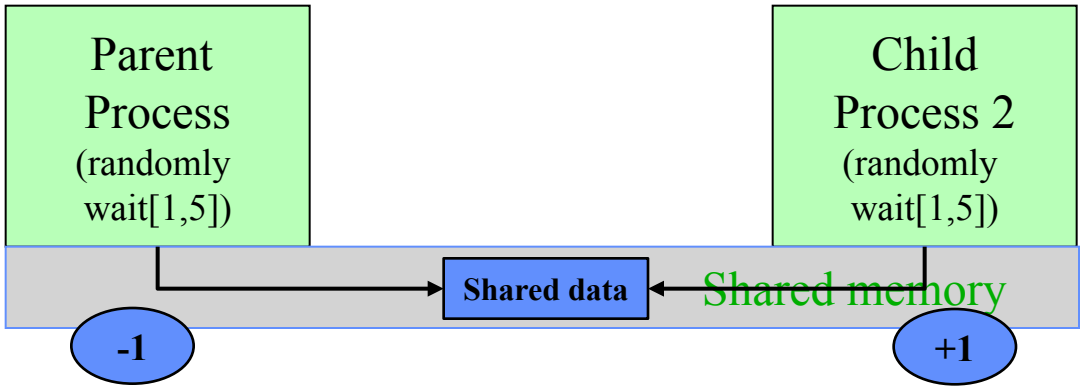
root@anclab5:~/NP/IPC

counter.c

```c
// counter.c
#include <stdio.h>      // printf()
#include <unistd.h>     // sleep()
#include <sys/ipc.h>
#include <sys/shm.h>

#define  KEY_NUM     9527
#define  MEM_SIZE    1024

int main( void)
{
   int   shm_id;
   void *shm_addr;
   int   count;

   if ( -1 == ( shm_id = shmget( (key_t)KEY_NUM, MEM_SIZE, IPC_CREAT|0666)))
   {
      printf( "shmget Error\n");
      return -1;
   }

   if ( ( void *)-1 == ( shm_addr = shmat( shm_id, ( void *)0, 0)))
   {
      printf( "shmat Error\n");
      return -1;
   }

   count = 0;
   while( 1 )
   {
      sprintf( (char *)shm_addr, "%d", count++);
      sleep( 1);
   }
   return 0;
}
```

1,1 All

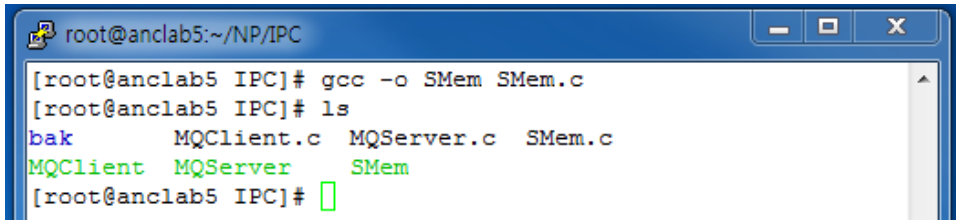
Problem: A game with the shared memory



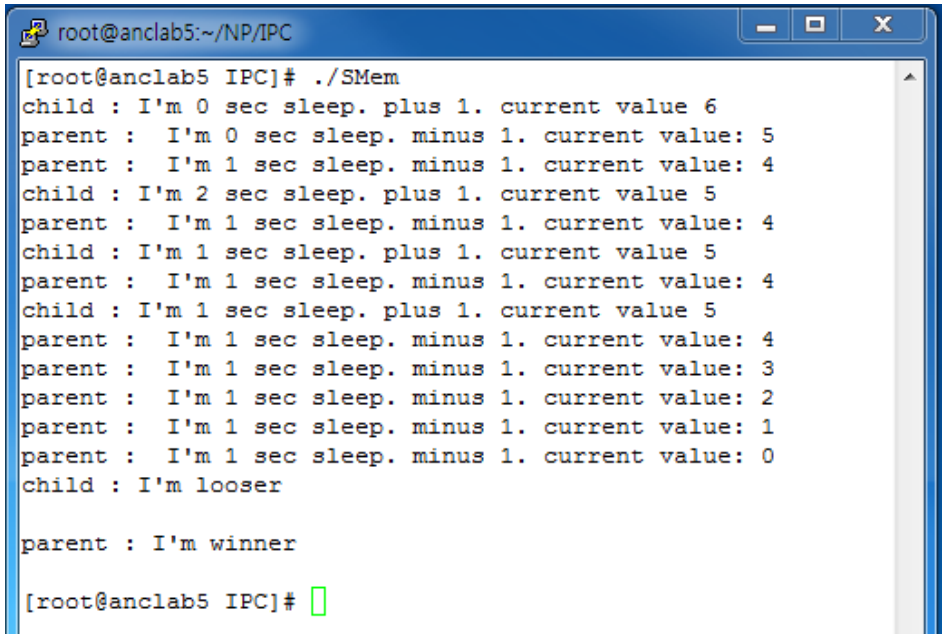
- Rule
  - Initial value of the shared data: 5
  - Parent process adds -1 to the shared data
  - Child process adds +1 to the shared data
  - Randomly wait
  - If the shared data reach 0, parent process wins the game.
  - If the shared data reach 10, child process wins the game.

Shared Memory

- Compile



- Execution



Type 2: Shared memory


```
#include <stdio.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <string.h>
#include <unistd.h>
#include <time.h>

#define SHARED_MEMORY_KEY 1234
#define INIT_VALUE 5
#define MAX_VALUE 10
#define MIN_VALUE 0
#define RAND_DISTANCE 5

int main(){
   int shmid;
   int pid;
   int *cal_num;
   int sleep_time=0;
   int status;
   int seed1, seed2;

   srand(time(NULL));
   seed1 = rand();
   seed2 = rand();

   shmid = shmget((key_t)SHARED_MEMORY_KEY, sizeof(int), 0666 |
         IPC_CREAT);
   if(shmid == -1){
      perror("shmget failed : ");
      exit(0);
   }

   cal_num = (int *)shmat(shmid, NULL, 0);
   if(cal_num ==(int *)-1){
      perror("shmat failed : ");
      exit(0);
   }
   *cal_num = INIT_VALUE;
 pid = fork();

if(pid == 0){
      srand(seed2);
      while(1){
         if((*cal_num) >= MAX_VALUE || (*cal_num)<= MIN_VALUE){
            break;
         }
         *cal_num = *cal_num + 1;
         printf("child : I'm %d sec sleep. plus 1. current value %d\n", sleep_time, *cal_num);
         sleep_time = (rand()%RAND_DISTANCE) +1;
         sleep(sleep_time);
      }
      if((*cal_num) >= MAX_VALUE){
         puts("child : I'm winner\n");
      }
      else{
         puts("child : I'm looser\n");
      }
      shmdt(cal_num);
   }
   else{
      while(1){
         srand(seed1);
         if((*cal_num) >= MAX_VALUE || (*cal_num)<= MIN_VALUE){
            break;
         }
         *cal_num = *cal_num - 1;
         printf("parent :  I'm %d sec sleep. minus 1. current value: %d\n", sleep_time,
          *cal_num);
         sleep_time = (rand()%RAND_DISTANCE) +1;
         sleep(sleep_time);
      }
      if((*cal_num) >= MAX_VALUE){
         puts("parent : I'm looser\n");
      }
      else{
         puts("parent : I'm winner\n");
      }
      shmdt(cal_num);
      shmctl(shmid, IPC_RMID, NULL);
      waitpid(pid, &status, 0);
   }
   return 0;
}
```

root@anclab5:~/NP/IPC

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

#define SHARED_MEMORY_KEY 1234
#define INIT_VALUE 5
#define MAX_VALUE 10
#define MIN_VALUE 0
#define RAND_DISTANCE 5

int main()
{
    int shmid;
    int pid;
    int *cal_num;
    int sleep_time=0;
    int status;
    int seed1, seed2;

    srand(time(NULL));
    seed1 = rand();
    seed2 = rand();

    // make space that shared-memory
    shmid = shmget((key_t)SHARED_MEMORY_KEY, sizeof(int), 0666 | IPC_CREAT);
    if(shmid == -1){
        perror("shmget failed : ");
        exit(0);
    }

    cal_num = (int *)shmat(shmid, NULL, 0);
    if(cal_num ==(int *)-1){
        perror("shmat failed : ");
        exit(0);
    }
    *cal_num = INIT_VALUE;

    // make child process
    pid = fork();

    //child process
    if(pid == 0){
        srand(seed2);
        while(1){
            if((*cal_num) >= MAX_VALUE || (*cal_num)<= MIN_VALUE){
                break;
            }
            *cal_num = *cal_num + 1;
            printf("child : I'm %d sec sleep. plus 1. current value %d\n", sleep_time, *cal_num);
            sleep_time = (rand()%RAND_DISTANCE) +1;
            sleep(sleep_time);
        }
        if((*cal_num) >= MAX_VALUE){
```

1,1 Top

Source code for Shared Memory

Source code for Shared Memory (Cont'd)

root@anclab5:~/NP/IPC

```
        exit(0);
    }

    cal_num = (int *)shmat(shmid, NULL, 0);
    if(cal_num ==(int *)-1){
        perror("shmat failed : ");
        exit(0);
    }
    *cal_num = INIT_VALUE;

    // make child process
    pid = fork();

    //child process
    if(pid == 0){
        srand(seed2);
        while(1){
            if((*cal_num) >= MAX_VALUE || (*cal_num)<= MIN_VALUE){
                break;
            }
            *cal_num = *cal_num + 1;
            printf("child : I'm %d sec sleep. plus 1. current value %d\n", sleep_time, *cal_num);
            sleep_time = (rand()%RAND_DISTANCE) +1;
            sleep(sleep_time);
        }
        if((*cal_num) >= MAX_VALUE){
            puts("child : I'm winner\n");
        }
        else{
            puts("child : I'm looser\n");
        }
        shmdt(cal_num);
    }
    // parent process
    else{
        while(1){
            srand(seed1);
            if((*cal_num) >= MAX_VALUE || (*cal_num)<= MIN_VALUE){
                break;
            }
            *cal_num = *cal_num - 1;
            printf("parent :  I'm %d sec sleep. minus 1. current value: %d\n", sleep_time, *cal_num);
            sleep_time = (rand()%RAND_DISTANCE) +1;
            sleep(sleep_time);
        }
        if((*cal_num) >= MAX_VALUE){
            puts("parent : I'm looser\n");
        }
        else{
            puts("parent : I'm winner\n");
        }
        shmdt(cal_num);
        shmctl(shmid, IPC_RMID, NULL);
        waitpid(pid, &status, 0);
    }
    return 0;
}
```

88,1 Bot

Lab Practice

- **Objectives: generate echo programs by using shared memory with two independent processes**
- **Requirements**
  - Generation of two processes (server, client)
  - Sample message type different from server-driven strings

    → "[Client-sent message]" + " by server"
  - Based on the Shared memory scheme
- **Execution example**