National University of Computer and Emerging Sciences

Lab Manual 8
(Operating Systems)

Department of Computer Science
FAST-NU, Lahore

IPC: Shared Memory

Ftok():

key_t key = ftok(char *filename, 0);

- Returns a key associated with the filename.

Shmget():

int id = shmget(key_t key, int size, int flags);

- Allocates a shared memory segment.
- Key is the key associated with the shared memory segment you want.
- Size is the size in bytes of the shared memory segment you want allocated.
- Memory is allocated in pages, so chances are you will probably get a little more memory than you wanted.
- Flags indicate how you want the segment created and its access permissions.
- The general rule is just to use 0666 | IPC_CREAT | IPC_EXCL if the caller is making a new segment.
- If the caller wants to use an existing share region, simply pass 0 in the flag.

RETURN VALUES

- Upon successful completion, **shmget**() returns the positive integer identifier of a shared memory segment.
- Otherwise, -1 is returned

Shmget will fail if:

1. Size specified is greater than the size of the previously existing segment. Size specified is less than the system imposed minimum, or greater than the system imposed maximum.
2. No shared memory segment was found matching *key*, and IPC_CREAT was not specified.
3. The kernel was unable to allocate enough memory to satisfy the request.
4. IPC_CREAT and IPC_EXCL were specified, and a shared memory segment corresponding to *key* already exists.

Shmat():

void *shmat(int shmid, const void *shmaddr, int shmflg);

- Maps a shared memory segment onto your process's address space.
- shmid is the id as returned by shmget() of the shared memory segment you wish to attach.
- Addr is the address where you want to attach the shared memory. For simplicity, we will pass **NULL**.
- NULL means that kernel itself will decide where to attach it to address space of the process.

RETURN VALUES

- Upon success, **shmat**() returns the address where the segment is attached;
- Otherwise, -1 is returned and *errno* is set to indicate the error.
- Upon success, **shmdt**() returns 0; otherwise, -1 is returned and *errno* is set to indicate the error.

Shmat() will fail if:

1. No shared memory segment was found corresponding to the given id.

Shmdt():

int shmdt(void *addr);

- This system call is used to detach a shared memory region from the process's address space.
- Addr is the address of the shared memory

RETURN VALUES

- On success, shmdt() returns 0; on error -1 is returned

shmdt will fail if:

1. The address passed to it does not correspond to a shared region.

Delete Shared Memory Region:

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

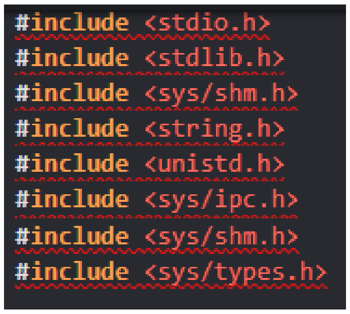
- shmctl(shmid, IPC_RMID, NULL);
- **shmctl**() performs the control operation specified by *cmd* on the System V shared memory segment whose identifier is given in *shmid*.
- For Deletion, we will use **IPC_RMID** flag.
- **IPC_RMID** marks the segment to be destroyed.
- The segment will actually be destroyed only after the last process detaches it (**The caller must be the owner or creator of the segment, or be privileged**).
- The *buf* argument is ignored.

Return Value:

- For IPC_RMID operation, 0 is returned on success; else -1 is returned.

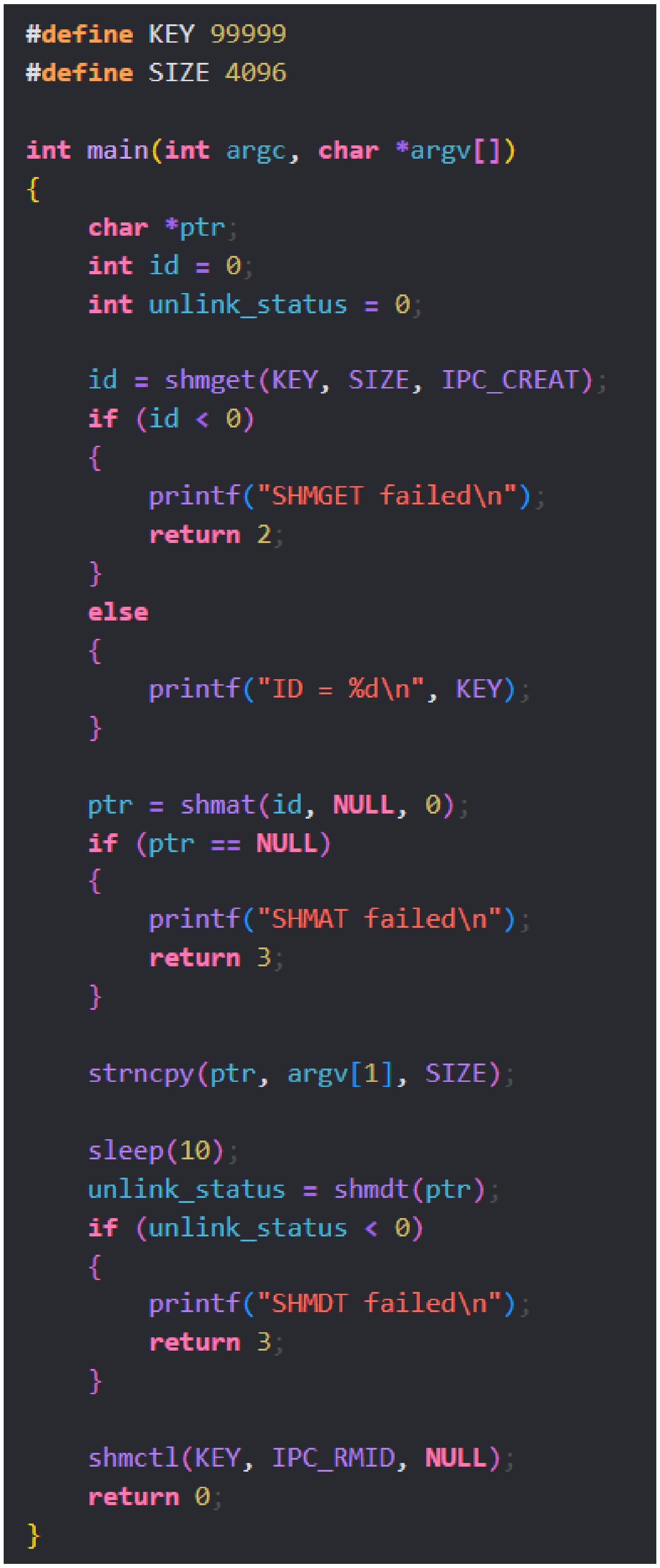
Example

Process 1 sends a text, passed to it via command line arguments, to the process 2. It first creates a shared memory area and writes the text to it. It also waits for 10 seconds before unlinking and deleting that memory area.
Process 2 accesses that shared memory area, reads the text, and prints it on the screen. Finally, it unlinks itself from it and exits.
Header Files:



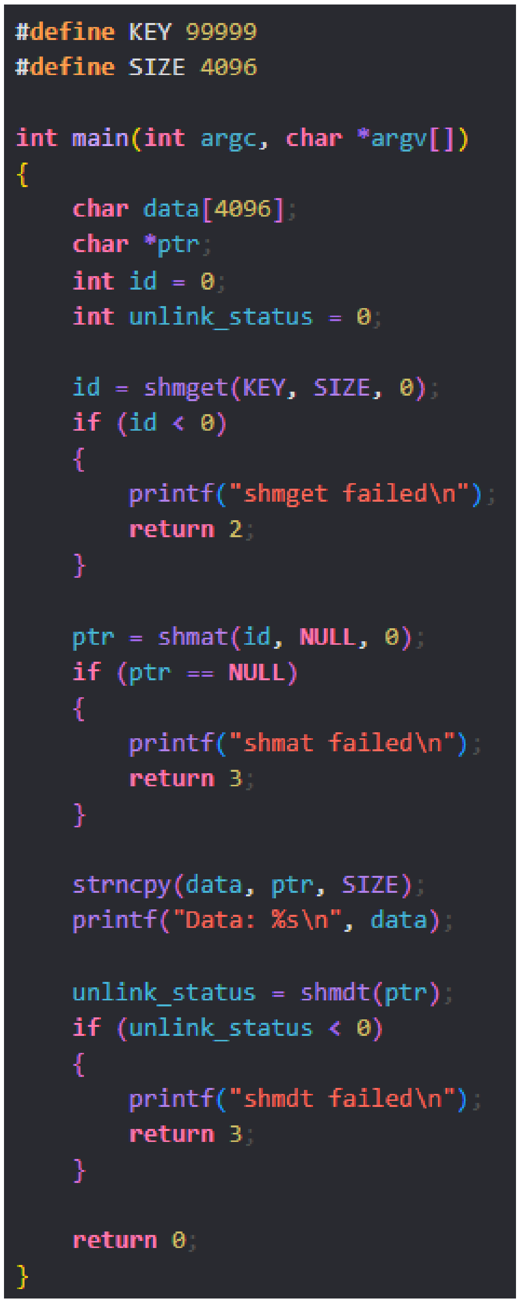
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <string.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
```

Process 1:



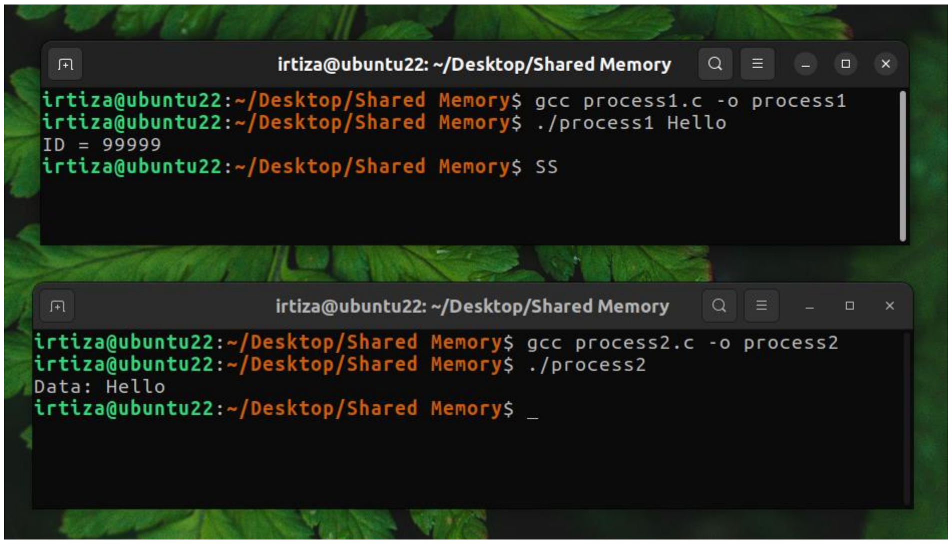
```c
#define KEY 99999
#define SIZE 4096

int main(int argc, char *argv[])
{
    char *ptr;
    int id = 0;
    int unlink_status = 0;

    id = shmget(KEY, SIZE, IPC_CREAT);
    if (id < 0)
    {
        printf("SHMGET failed\n");
        return 2;
    }
    else
    {
        printf("ID = %d\n", KEY);
    }

    ptr = shmat(id, NULL, 0);
    if (ptr == NULL)
    {
        printf("SHMAT failed\n");
        return 3;
    }

    strncpy(ptr, argv[1], SIZE);

    sleep(10);
    unlink_status = shmdt(ptr);
    if (unlink_status < 0)
    {
        printf("SHMDT failed\n");
        return 3;
    }

    shmctl(KEY, IPC_RMID, NULL);
    return 0;
}
```

Process 2:



```c
#define KEY 99999
#define SIZE 4096

int main(int argc, char *argv[])
{
    char data[4096];
    char *ptr;
    int id = 0;
    int unlink_status = 0;

    id = shmget(KEY, SIZE, 0);
    if (id < 0)
    {
        printf("shmget failed\n");
        return 2;
    }

    ptr = shmat(id, NULL, 0);
    if (ptr == NULL)
    {
        printf("shmat failed\n");
        return 3;
    }

    strncpy(data, ptr, SIZE);
    printf("Data: %s\n", data);

    unlink_status = shmdt(ptr);
    if (unlink_status < 0)
    {
        printf("shmdt failed\n");
        return 3;
    }

    return 0;
}
```

Output:



irtiza@ubuntu22: ~/Desktop/Shared Memory

```
irtiza@ubuntu22:~/Desktop/Shared Memory$ gcc process1.c -o process1
irtiza@ubuntu22:~/Desktop/Shared Memory$ ./process1 Hello
ID = 99999
irtiza@ubuntu22:~/Desktop/Shared Memory$ SS
```

irtiza@ubuntu22: ~/Desktop/Shared Memory

```
irtiza@ubuntu22:~/Desktop/Shared Memory$ gcc process2.c -o process2
irtiza@ubuntu22:~/Desktop/Shared Memory$ ./process2
Data: Hello
irtiza@ubuntu22:~/Desktop/Shared Memory$ _
```

In Lab Tasks: **(10 Marks)**

- Create 2 processes, **client** and **server**
- A client process read data from a file named “number.txt” (passed as a command-line argument) and sends the data to a server process (unrelated process) via shared memory.
- The server process reads the data from the shared memory and display the **sum** and **average** of the integers.

Note 1: Use open, read, and write system calls for file handling.
Note 2: Use **strncpy** for writing data to shared memory.

Sample Data for File:
1 2 8 8 6

Output should be:
Sum = 25
Average = 5

Submit File with name as: YOUR_ROLLNUMBER_Q.c and also submit output with name YOUR_ROLLNUMBER_Q_OUTPUT.jpg