

C Programming Basics

Lab Session: Inter-Process Communication (IPC) using Shared Memory in Linux

Course: Operating Systems

Date: 02.06.2025

Program 1: Shared Memory Communication – Writer Process

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

#define SHM_SIZE 1024 // Size of shared memory segment

int main() {
    key_t key = ftok("nuzha", 100);
    if (key == -1) {
        perror("ftok");
        exit(1);
    }

    int shmid = shmget(key, SHM_SIZE, IPC_CREAT | 0666);
    if (shmid == -1) {
        perror("shmget");
        exit(1);
    }
```

```

char *shmaddr = (char*) shmat(shmid, NULL, 0);

if (shmaddr == (char*) -1) {
    perror("shmat");
    exit(1);
}

printf("Write Data: ");
fgets(shmaddr, SHM_SIZE, stdin);

printf("Data written in memory: %s\n", shmaddr);
shmdt(shmaddr);

return 0;
}

```

Program 2: Shared Memory Communication – Reader Process

```

#include <sys/shm.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#define SHM_SIZE 1024

int main() {
    key_t key = ftok("nuzha", 100);
    if (key == -1) {

```

```
perror("ftok");
exit(1);
}

int shmid = shmget(key, SHM_SIZE, 0666);
if (shmid == -1) {
    perror("shmget");
    exit(1);
}

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

printf("Data read from shared memory: %s\n", shmaddr);

shmdt(shmaddr);
shmctl(shmid, IPC_RMID, NULL);

return 0;
}
```

Fedora Output

```
[2021ict108@fedora ~]$ vi writer.c
```

```
[2021ict108@fedora ~]$ gcc writer.c -o writer
```

```
[2021ict108@fedora ~]$ vi reader.c
```

```
[2021ict108@fedora ~]$ gcc reader.c -o reader
```

```
[2021ict108@fedora ~]$ ./writer
```

Write Data: hi nuzha

Data written in memory: hi nuzha

```
[2021ict108@fedora ~]$ ./reader
```

Data read from shared memory: hi nuzha

Explanation

- `ftok()` generates a unique key.
 - `shmget()` creates or accesses a shared memory segment.
 - `shmat()` attaches the shared memory to the process.
 - `shmdt()` detaches it.
 - `shmctl()` is used to remove the memory segment in the reader.
-

Exercise 02: Parent-Child IPC via Shared Memory

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

int main() {
```

```
size_t size = 4096;

char *shared_mem = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_SHARED |
MAP_ANONYMOUS, -1, 0);

if (shared_mem == MAP_FAILED) {
    perror("mmap failed");
    exit(1);
}

pid_t pid = fork();

if (pid < 0) {
    perror("fork failed");
    exit(1);
} else if (pid == 0) {
    printf("Child Process (PID: %d)\n", getpid());
    printf("Enter a message: ");
    fgets(shared_mem, size, stdin);
    printf("Child wrote: %s\n", shared_mem);
    exit(0);
} else {
    wait(NULL);
    printf("Parent Process (PID: %d)\n", getpid());
    printf("Parent reads: %s\n", shared_mem);
    munmap(shared_mem, size);
}
```

```
    return 0;  
}  


---


```

Fedora Output

```
[2021ict108@fedora ~]$ gcc ICP3.c -o ICP3
```

```
[2021ict108@fedora ~]$ ./ICP3
```

```
Child Process (PID: 15991)
```

```
Enter a message: NUZHA
```

```
Child wrote: NUZHA
```

```
Parent Process (PID: 15990)
```

```
Parent reads: NUZHA
```

Explanation

- Uses mmap() to create anonymous shared memory between parent and child.
 - The child writes a message; the parent reads after child exits.
 - getpid() prints respective process IDs.
-

Exercise 03: IPC – Factorial, NCR, NPR Calculation

```
#include <stdio.h>  
  
#include <stdlib.h>  
  
#include <unistd.h>  
  
#include <sys/mman.h>  
  
#include <sys/wait.h>  
  
  
int main() {
```

```
size_t size = 3 * sizeof(long);

long *shared_mem = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_SHARED |
MAP_ANONYMOUS, -1, 0);

if (shared_mem == MAP_FAILED) {
    perror("mmap failed");
    exit(1);
}

int n, r;

printf("Enter value for n: ");
scanf("%d", &n);

printf("Enter value for r: ");
scanf("%d", &r);

pid_t pid = fork();

if (pid < 0) {
    perror("fork failed");
    exit(1);
} else if (pid == 0) {
    printf("Child process started (PID: %d)\n", getpid());

long factorial(int num) {
    long fact = 1;
    for (int i = 1; i <= num; i++)
        fact *= i;
```

```
    return fact;
}

shared_mem[0] = factorial(n);

shared_mem[1] = factorial(r);

shared_mem[2] = factorial(n - r);

printf("Child calculated factorials and stored them in shared memory.\n");

exit(0);

} else {

    wait(NULL);

long fact_n = shared_mem[0];

long fact_r = shared_mem[1];

long fact_n_r = shared_mem[2];

long ncr = fact_n / (fact_r * fact_n_r);

long npr = fact_n / fact_n_r;

printf("Parent process started (PID: %d)\n", getpid());

printf("Factorial of %d: %ld\n", n, fact_n);

printf("Factorial of %d: %ld\n", r, fact_r);

printf("NCR (%dC%d): %ld\n", n, r, ncr);

printf("NPR (%dP%d): %ld\n", n, r, npr);

munmap(shared_mem, size);
```

```
    }  
  
    return 0;  
}
```

Sample Output

Enter value for n: 5

Enter value for r: 2

Child process started (PID: 16000)

Child calculated factorials and stored them in shared memory.

Parent process started (PID: 15999)

Factorial of 5: 120

Factorial of 2: 2

NCR (5C2): 10

NPR (5P2): 20

Explanation

- The child calculates factorials of n, r, and n-r.
- Parent reads from shared memory to compute nCr and nPr .
- Shared memory allows efficient data passing between processes.