# Experiment no: 6

Name: Saloni Vishwakarma
Batch-Roll no: C1-13
Semester-Section: 4th-C
Subject: OS(Operating System) Lab
Date of execution: 21 June 2023

**Aim:**
**A)** Write 2 C programs A.c and B.c. The program A.c reads in a set of integers (maximum 100) from the file named inpfile and writes it to a shared array. The program B.c reads the set of integers from the shared array (how will it know how many integers are there?), sorts it and prints the sorted output in a file named outfile.
Make sure that B.c deletes all shared memory created before exiting.
There is an obvious synchronization problem here, B.c should not start until A.c has finished writing the integers in the array. For the first part, ignore it, and start the program for B.c a few seconds after A.c starts.

**B)** In this part, we will try to synchronize A.c and B.c by a simple method. Create a shared integer variable called done and initialize it to 0. done = 0 indicates that A.c has not finished writing the integers into the array. The program in A.c sets done to 1 after it finishes. The program in B.c periodically checks done and loops until it is 1. Modify A.c and B.c to implement this.

**Theory:**
Interprocess communication (IPC) in operating systems refers to the mechanisms and techniques used by processes running on an operating system to communicate and exchange data with each other. IPC is crucial for enabling collaboration, coordination, and synchronization among processes.

There are several methods of IPC that can be used, including:
Shared Memory: Processes can access a shared portion of memory to exchange data. This method allows for fast and efficient communication but requires synchronization mechanisms to manage access to shared memory.

Message Passing: Processes communicate by sending and receiving messages. Messages can be sent through various mechanisms such as queues, mailboxes, or channels. This method provides a more structured approach to communication and ensures data integrity.

Pipes: A pipe is a unidirectional communication channel between two related processes. It allows one process to write data to the pipe, which can then be read by the other process. Pipes are commonly used for IPC in scenarios where a producer-consumer relationship exists between processes.
Sockets: Sockets are communication endpoints used for network-based IPC. They enable processes running on different machines to establish connections and exchange data over a network.

Signals: Signals are asynchronous notifications sent by the operating system to processes. They can be used to inform processes about specific events or handle exceptional conditions like interrupts or errors.

Semaphores: Semaphores are synchronization objects used for managing access to shared resources. They help in preventing race conditions and ensuring mutual exclusion when multiple processes are accessing shared resources concurrently.

Remote Procedure Call (RPC): RPC allows a process to execute a procedure or function on a remote process as if it were a local procedure call. It provides a high-level abstraction for IPC and is commonly used in distributed systems.
Each method of IPC has its advantages and considerations. The choice of IPC method depends on factors such as the nature of the application, performance requirements, and the desired level of complexity.

Overall, IPC plays a vital role in facilitating communication and data exchange among processes, enabling the creation of sophisticated and collaborative applications in operating systems.

**Viva Questions:**

1) What is interprocess communication (IPC)?

→ Interprocess communication (IPC) refers to the mechanisms and techniques used by processes running on an operating system to communicate and exchange data with each other.

2) Why is IPC important in operating systems?
→ IPC is important in operating systems because it enables processes to collaborate, share information, and synchronize their activities. It allows for the coordination of tasks, resource sharing, and the implementation of complex system functionalities.

3) What are the different methods of IPC?
→ The different methods of IPC include shared memory, message passing, pipes, sockets, signals, semaphores, and remote procedure call (RPC).

4) Explain the differences between synchronous and asynchronous IPC.
→Synchronous IPC involves blocking the sending process until the receiving process acknowledges receipt of the message or completes the requested operation. Asynchronous IPC allows the sending process to continue execution without waiting for a response from the receiver.

5) What is shared memory and how does it facilitate IPC?
→ Shared memory is a method of IPC where multiple processes can access the same region of memory. It allows processes to exchange data quickly and efficiently by reading and writing to shared memory locations.

6) Describe the message passing method of IPC.
→ Message passing is a method of IPC where processes communicate by sending and receiving messages. The sender process explicitly sends a message to the receiver process, which then processes the message accordingly.

7) What are the advantages and disadvantages of using shared memory for IPC?
→ Advantages of shared memory for IPC include fast data exchange, as it avoids the need for data copying, and the ability to share complex data structures. Disadvantages include the need for explicit synchronization mechanisms to manage access to shared memory and the potential for data inconsistencies or race conditions.

## Code and Output (A):

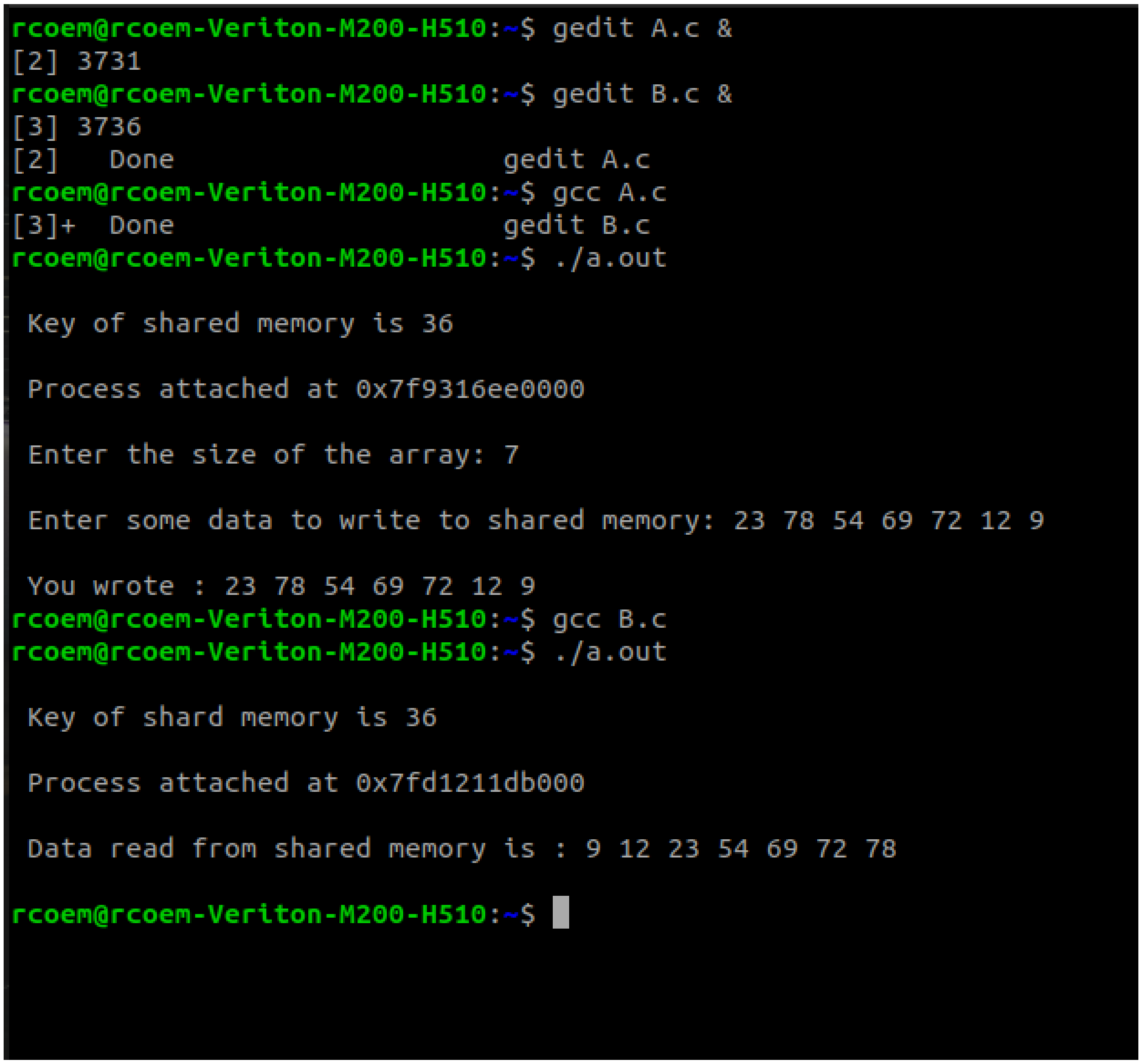
**File name: A.c**

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
  int i,n;
  int *shared_memory;
  int buff[100];
  int shmid;
  shmid=shmget((key_t)2345, 1024, 0666|IPC_CREAT);
  printf("\n Key of shared memory is %d\n",shmid);
  shared_memory=(int*)shmat(shmid,NULL,0);
  printf("\n Process attached at %p\n",shared_memory);
  printf("\n Enter the size of the array: ");
  scanf("%d",&n);
  printf("\n Enter some data to write to shared memory: ");
  for(i=0;i<n;i++){
        scanf("%d",&buff[i]);
        shared_memory[i]=buff[i];//data written to shared memory
  }
  printf("\n You wrote : ");
  for(i=0;i<n;i++){
        printf("%d ",buff[i]);
  }
  printf("\n");
  shmdt((void *) shared_memory);
}
```

**File name: B.c**

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
```

```
{
  int i,j,a;
  int *shared_memory; int buff[100]; int shmid;
  shmid=shmget((key_t)2345,1024,0666); printf("\n Key of shard memory is %d\n",shmid);
  shared_memory=shmat(shmid,NULL,0);
  printf("\n Process attached at %p\n",shared_memory);
  for (i = 0; i < 100; ++i)
  {
        for (j = i + 1; j < 100; ++j)
        {
        if (shared_memory[i] > shared_memory[j])
        {
        a = shared_memory[i];
        shared_memory[i] = shared_memory[j];
        shared_memory[j] = a;
        }
        }
  }
  printf("\n Data read from shared memory is : ");
  for(i=0; i<100; i++){
        if(shared_memory[i] != 0)
        {
        printf("%d ",shared_memory[i] );
        }
  }
  shmdt((void *)shared_memory);
  shmctl(shmid, IPC_RMID, NULL);
  printf("\n\n");
  return 0;
}
```



```
rcoem@rcoem-Veriton-M200-H510:~$ gedit A.c &
[2] 3731
rcoem@rcoem-Veriton-M200-H510:~$ gedit B.c &
[3] 3736
[2]   Done                    gedit A.c
rcoem@rcoem-Veriton-M200-H510:~$ gcc A.c
[3]+  Done                    gedit B.c
rcoem@rcoem-Veriton-M200-H510:~$ ./a.out

 Key of shared memory is 36

 Process attached at 0x7f9316ee0000

 Enter the size of the array: 7

 Enter some data to write to shared memory: 23 78 54 69 72 12 9

 You wrote : 23 78 54 69 72 12 9
rcoem@rcoem-Veriton-M200-H510:~$ gcc B.c
rcoem@rcoem-Veriton-M200-H510:~$ ./a.out

 Key of shard memory is 36

 Process attached at 0x7fd1211db000

 Data read from shared memory is : 9 12 23 54 69 72 78

rcoem@rcoem-Veriton-M200-H510:~$ 
```

**Code and Output (B):**

**File name: A.c**

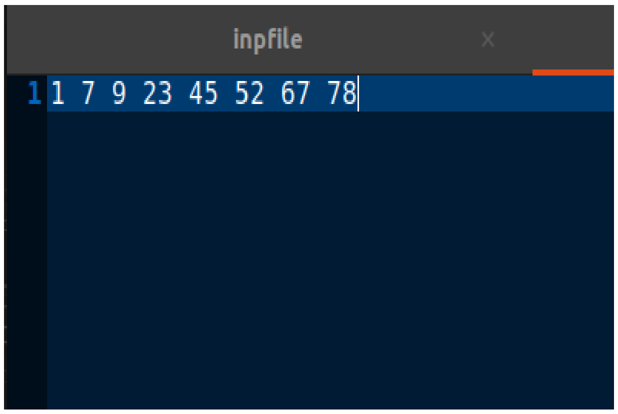
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>
int main()
{
        key_t key = 1020;
        int shmid = shmget(key, sizeof(int) *100, IPC_CREAT | 0666);
        if (shmid == -1)
        {
        perror("shmget");
        exit(1);
        }
        int *sharedArray = (int *)shmat(shmid, NULL, 0);
        if (sharedArray == (int *)(-1))
        {
        perror("shmat");
        exit(1);
        }
        int shmid2 = shmget(key, sizeof(int), IPC_CREAT | 0666);
        if (shmid2 == -1)
        {
        perror("shmget");
        exit(1);
        }
        int *done = (int *)shmat(shmid2, NULL, 0);
        if (done == (int *)(-1))
        {
        perror("shmat");
        exit(1);
        }
        *done = 0;
        FILE *inputFile = fopen("inpfile", "r");
        if (inputFile == NULL)
        {
        perror("fopen"); exit(1);
        }
        int numInts = 0;
        while (fscanf(inputFile, "%d",&sharedArray[numInts]) == 1)
        {
        numInts++;
        if (numInts >=100)
        break;
        }
        fclose(inputFile);
        shmdt(sharedArray);
        *done = 1;
        return 0;
}
```

**File name: B.c**

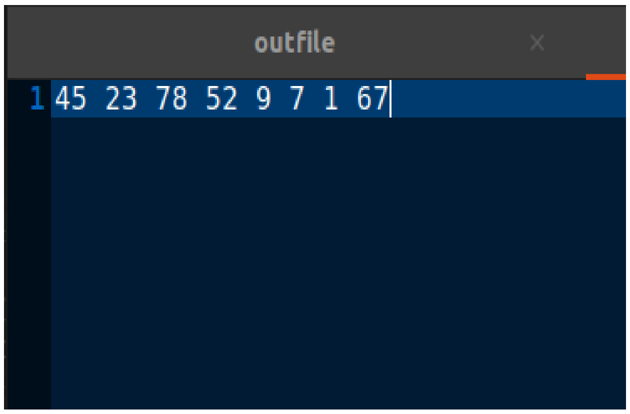
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int compare(const void *a, const void *b)
{
        return (*(int *)a - *(int *)b);
}
int main()
{
        key_t key = 1020;
        int shmid = shmget(key, sizeof(int) *100, IPC_CREAT | 0666);
        if (shmid == -1)
        {
        perror("shmget");
        exit(1);
        }
        int *sharedArray = (int *)shmat(shmid, NULL, 0);
        if (sharedArray == (int *)(-1))
        {
        perror("shmat");
        exit(1);
        }
        int shmid2 = shmget(key, sizeof(int), IPC_CREAT | 0666);
        if (shmid2 == -1)
        {
        perror("shmget");
        exit(1);
        }

        int *done = (int *)shmat(shmid2, NULL, 0);
        if (done == (int *)(-1))
        {
        perror("shmat");
        exit(1);
        }
        while(*done==0)
        {
```

```
        ;
        }
        int numInts = 0;
        while(sharedArray[numInts] != 0) {
        numInts++;
        if (numInts >= 100)
        break;
        }
        qsort(sharedArray, numInts, sizeof(int), compare);
        FILE *outputFile = fopen("outputfile", "w");
        if(outputFile == NULL)
        {
        perror("fopen");
        exit(1);
        }
        for (int i = 0; i < numInts; i++)
        {
        fprintf(outputFile, "%d ", sharedArray[i]);
        }
        fclose(outputFile);
        shmdt(sharedArray);
        shmctl(shmid, IPC_RMID, NULL);
        return 0;
}
```



```
1 7 9 23 45 52 67 78
```



```
45 23 78 52 9 7 1 67
```

**Conclusion:** In conclusion, interprocess communication (IPC) is a fundamental aspect of operating systems that allows processes to communicate and exchange data with each other. IPC enables processes to collaborate, share information, and synchronize their activities, leading to the implementation of complex system functionalities.