OPERATING SYSTEM LAB
PRACTICAL : 06

NAME: Arya Narendra Narlawar
BRANCH: CSE
Roll Number: 30

AIM:
1) Use of shared memory(with and without synchronization)

Part 1
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.

Part 2
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.

Note: Unix requires a key of type key_t defined in file sys/types. h for requesting resources such as shared memory segments, message queues and semaphores. A key is simply an integer of type key_t.

CODE:

A.c file

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

#define MAX_SIZE 100

typedef struct {
   int numbers[MAX_SIZE];
   int count;
} SharedData;

int main() {
```

```
    key_t key = ftok("inpfile", 'R'); // Generate a unique key based on "inpfile" file

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

    SharedData* sharedData = (SharedData*)shmat(shmid, NULL, 0);
    if (sharedData == (void*)-1) {
        perror("shmat");
        exit(1);
    }

    FILE* file = fopen("inpfile", "r");
    if (file == NULL) {
        perror("fopen");
        exit(1);
    }

    int number;
    int index = 0;

    while (fscanf(file, "%d", &number) != EOF && index < MAX_SIZE) {
        sharedData->numbers[index] = number;
        index++;
    }

    sharedData->count = index;

    fclose(file);
    shmdt(sharedData);

    sleep(5); // Delay to allow B.c to start after A.c has finished writing

    return 0;
}
```

B.c file

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

#define MAX_SIZE 100

typedef struct {
    int numbers[MAX_SIZE];
    int count;
```

```
} SharedData;

int main() {
    key_t key = ftok("inpfile", 'R'); // Use the same key as A.c to access the shared memory

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

    int* done = (int*)shmat(shmid, NULL, 0);

    // Synchronization: Wait until done becomes 1
    while (*done != 1) {
        sleep(1);
    }

    shmdt(done);

    SharedData* sharedData = (SharedData*)shmat(shmid, NULL, 0);
    if (sharedData == (void*)-1) {
        perror("shmat");
        exit(1);
    }

    // Sorting the numbers
    int i, j, temp;
    for (i = 0; i < sharedData->count - 1; i++) {
        for (j = 0; j < sharedData->count - i - 1; j++) {
            if (sharedData->numbers[j] > sharedData->numbers[j + 1]) {
                temp = sharedData->numbers[j];
                sharedData->numbers[j] = sharedData->numbers[j + 1];
                sharedData->numbers[j + 1] = temp;
            }
        }
    }

    FILE* file = fopen("outfile", "w");
    if (file == NULL) {
        perror("fopen");
        exit(1);
    }

    for (int i = 0; i < sharedData->count; i++) {
        fprintf(file, "%d\n", sharedData->numbers[i]);
    }

    fclose(file);
    shmdt(sharedData);
    shmctl(shmid, IPC_RMID, NULL); // Delete shared memory segment
```

```
    return 0;
}
```

A1.c file

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

#define MAX_SIZE 100

typedef struct {
    int numbers[MAX_SIZE];
    int count;
} SharedData;

int main() {
    key_t key = ftok("inpfile", 'R'); // Generate a unique key based on "inpfile" file

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

    SharedData* sharedData = (SharedData*)shmat(shmid, NULL, 0);
    if (sharedData == (void*)-1) {
        perror("shmat");
        exit(1);
    }

    FILE* file = fopen("inpfile", "r");
    if (file == NULL) {
        perror("fopen");
        exit(1);
    }

    int number;
    int index = 0;

    while (fscanf(file, "%d", &number) != EOF && index < MAX_SIZE) {
        sharedData->numbers[index] = number;
        index++;
    }

    sharedData->count = index;

    fclose(file);
```

```
    // Synchronization: Set done to 1 indicating A.c has finished writing
    int* done = (int*)shmat(shmid, NULL, 0);
    *done = 1;
    shmdt(done);

    return 0;
}
```

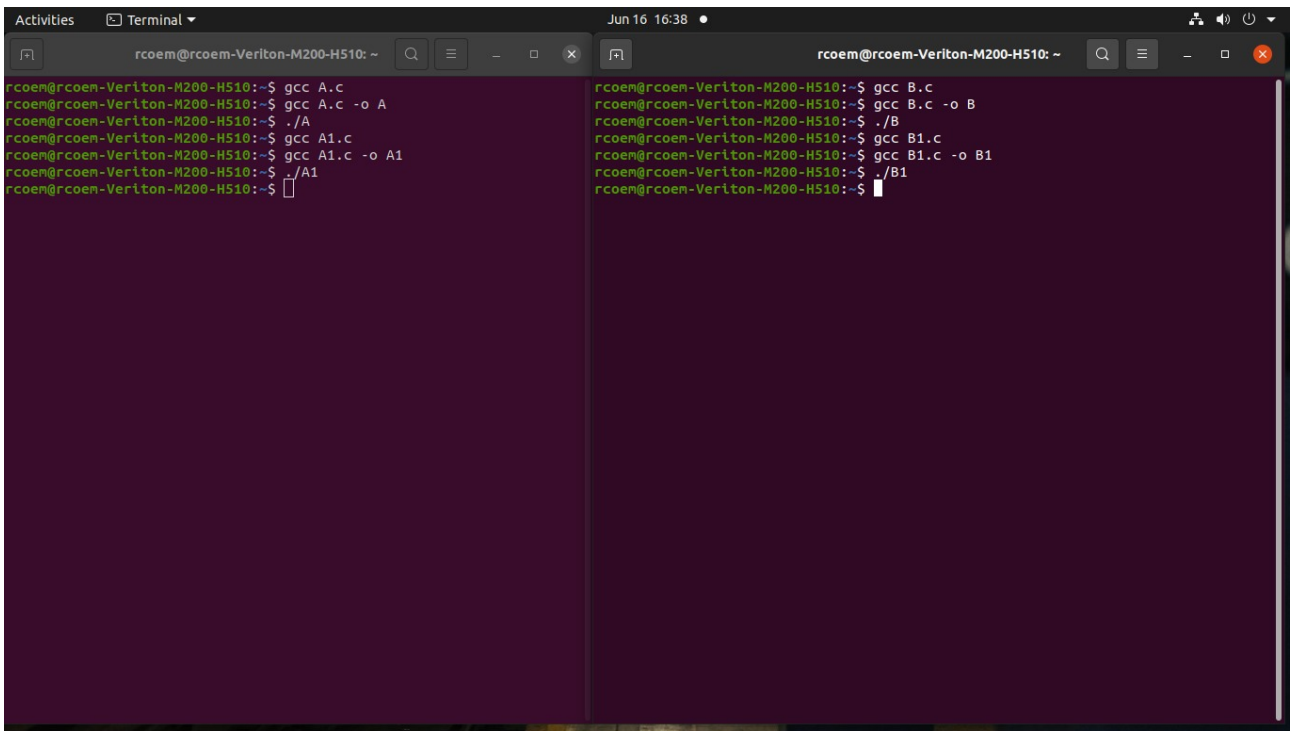
B1.c file

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

#define MAX_SIZE 100

typedef struct {
    int numbers[MAX_SIZE];
    int count;
} SharedData;

int main() {
    key_t key = ftok("inpfile", 'R'); // Use the same key as A.c to access the shared memory

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

    int* done = (int*)shmat(shmid, NULL, 0);

    // Synchronization: Wait until done becomes 1
    while (*done != 1) {
        sleep(1);
    }

    shmdt(done);

    SharedData* sharedData = (SharedData*)shmat(shmid, NULL, 0);
    if (sharedData == (void*)-1) {
        perror("shmat");
        exit(1);
    }

    // Sorting the numbers
    int i, j, temp;
    for (i = 0; i < sharedData->count - 1; i++) {
```

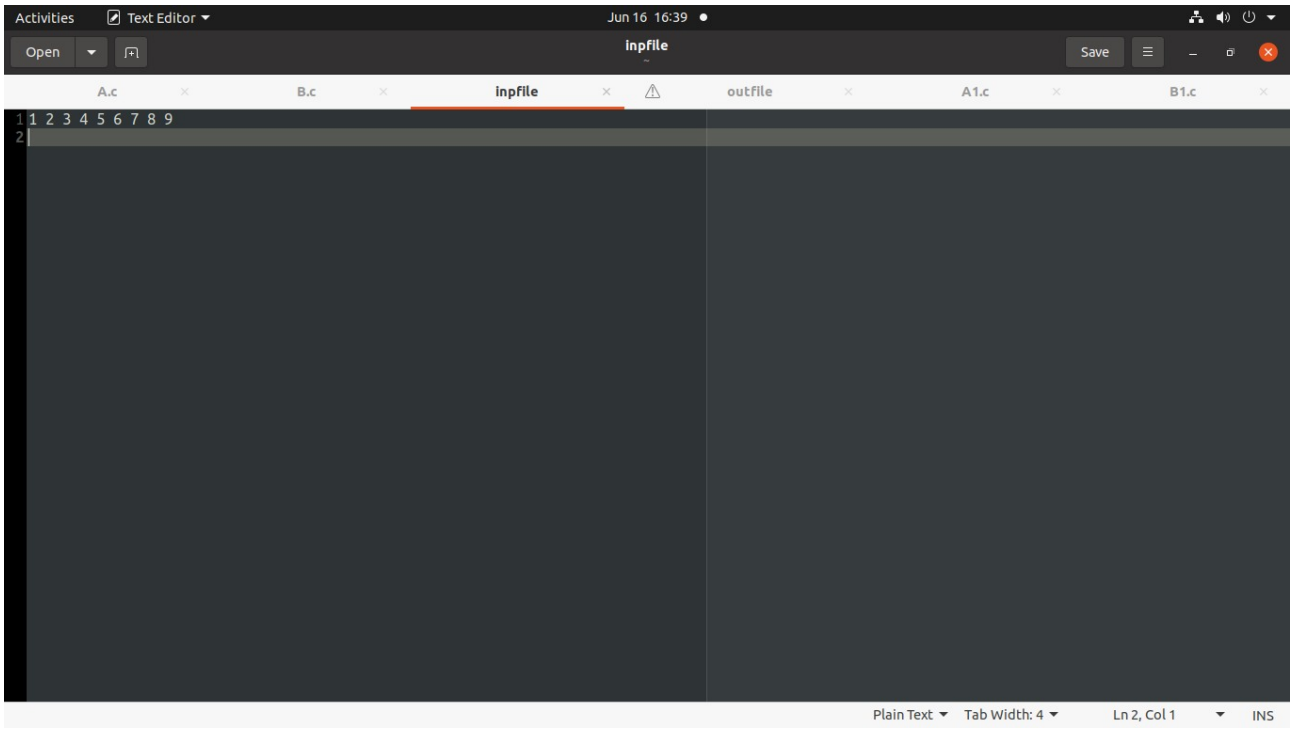
```
        for (j = 0; j < sharedData->count - i - 1; j++) {
            if (sharedData->numbers[j] > sharedData->numbers[j + 1]) {
                temp = sharedData->numbers[j];
                sharedData->numbers[j] = sharedData->numbers[j + 1];
                sharedData->numbers[j + 1] = temp;
            }
        }
    }

    FILE* file = fopen("outfile", "w");
    if (file == NULL) {
        perror("fopen");
        exit(1);
    }

    for (int i = 0; i < sharedData->count; i++) {
        fprintf(file, "%d\n", sharedData->numbers[i]);
    }

    fclose(file);
    shmdt(sharedData);
    shmctl(shmid, IPC_RMID, NULL); // Delete shared memory segment

    return 0;
}
```

OUTPUT:

TERMINAL 1 and 2:



inpfile



outfile