OS - Practical – 6 Part 1

Name: Rishabh Jain

Branch & Sem: CSE - IV

Sec: A

Roll no.:54

AIM : Implement a C program to demonstrate the concept of Shared Memory.  
  
1) Shared memory basic program to find the total of n numbers.  
  
#include <stdio.h>

#include <string.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/shm.h>

#include <sys/wait.h>

#include <unistd.h>

#define buf\_size 100

int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,

           20};

int main(void)

{

    pid\_t pid;

    int i;

    int \*total;

    char b[buf\_size];

    // Get the segment

    int segment\_id = shmget(IPC\_PRIVATE, sizeof(int), S\_IRUSR | S\_IWUSR);

    // Attach the segment with variable to be used by process

    total = (int \*)shmat(segment\_id, NULL, 0);

    \*total = 0;

    // Create new child

    pid = fork();

    if (pid == 0)

    {

        // Child Process

        for (i = 10; i < 20; i++)

        {

            \*total += a[i];

        }

        sprintf(b, "\nChild Total = %d\n\n", \*total);

        write(1, b, strlen(b));

    }

    else

    {

        // Parent Process

        for (i = 0; i < 10; i++)

        {

            \*total += a[i];

        }

        sprintf(b, "\nParent Total = %d\n\n", \*total);

        write(1, b, strlen(b));

        pid = wait(NULL);

        if (pid != -1)

        {

            printf("\nTotal of all Numbers = %d\n\n", \*total);

        }

        shmdt(total);

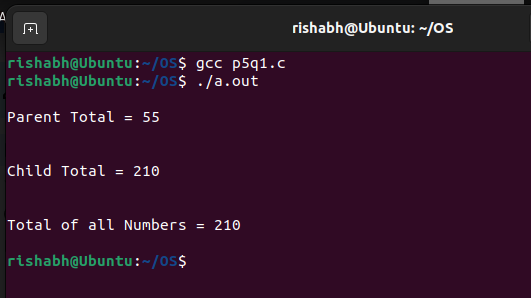
        // shmctl(segment\_id, IPC\_RMID, NULL); // Remove the shared memory

        segment

    }

    return 0;

}

OUTPUT:  


2) To find the maximum and minimum element in an array using shared memory.

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/shm.h>

int A[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,

           18, 19,

           20};

typedef struct

{

    int min;

    int max;

} find;

int main()

{

    int N;

    find \*ans;

    pid\_t pid;

    int segment\_id = shmget(IPC\_PRIVATE, sizeof(find), IPC\_CREAT | 0666);

    ans = (find \*)shmat(segment\_id, NULL, 0);

    ans->min = A[0];

    ans->max = A[0];

    pid = fork();

    if (pid == 0)

    {

        // Child Process which finds minimum

        for (int i = 1; i < 20; i++)

        {

            if (A[i] < ans->min)

                ans->min = A[i];

        }

        exit(0);

    }

    else

    {

        for (int i = 1; i < 20; i++)

        {

            if (A[i] > ans->max)

                ans->max = A[i];

        }

        wait(NULL);

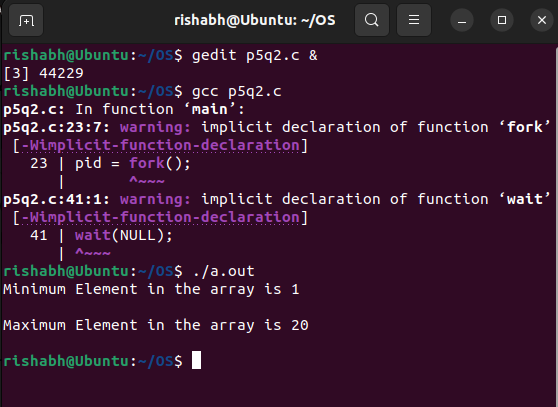
    }

    printf("Minimum Element in the array is %d\n\n", ans->min);

    printf("Maximum Element in the array is %d\n\n", ans->max);

    return 0;

}

OUTPUT:  


A structure is defined with integer data members called "min" and "max". When the program forks, it creates two processes: the Parent Process and the Child Process. The Child Process is responsible for finding the minimum element in an array, while the Parent Process finds the maximum element in the same array.  
  
3) Two processes communicating via shared memory: shm\_server.c, shm\_client.c  
  
Server.c

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <fcntl.h>

#define SHMSZ 27

int main()

{

    char c;

    int shmid;

    key\_t key;

    char \*shm, \*s;

    /\* We'll name our shared memory segment "5678". \*/

    key = 5678;

    /\* Create the segment. \*/

    if ((shmid = shmget(key, SHMSZ, IPC\_CREAT | 0666)) < 0)

    {

        perror("shmget");

        exit(1);

    }

    /\* Now we attach the segment to our data space. \*/

    if ((shm = shmat(shmid, NULL, 0)) == (char \*)-1)

    {

        perror("shmat");

        exit(1);

    }

    /\* Now put some things into the memory for the other process to

    read. \*/

    s = shm;

    for (c = 'a'; c <= 'z'; c++)

    {

        \*s++ = c;

    }

    \*s = '\0';

    /\* Finally, we wait until the other process changes the first

    character of

    our memory

    \* to '\*', indicating that it has read what we put there. \*/

    while (\*shm != '\*')

    {

        sleep(1);

    }

    /\* Detach the shared memory segment. \*/

    if (shmdt(shm) == -1)

    {

        perror("shmdt");

        exit(1);

    }

    /\* Delete the shared memory segment. \*/

    if (shmctl(shmid, IPC\_RMID, 0) == -1)

    {

        perror("shmctl");

        exit(1);

    }

    exit(0);

}

Client.c  
#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <fcntl.h>

#define SHMSZ 27

int main()

{

    int shmid;

    key\_t key;

    char \*shm, \*s;

    /\* We need to get the segment named "5678", created by the server.

     \*/

    key = 5678;

    /\* Locate the segment. \*/

    if ((shmid = shmget(key, SHMSZ, 0666)) < 0)

    {

        perror("shmget");

        exit(1);

    }

    /\* Now we attach the segment to our data space. \*/

    if ((shm = shmat(shmid, NULL, 0)) == (char \*)-1)

    {

        perror("shmat");

        exit(1);

    }

    /\* Now read what the server put in the memory. \*/

    for (s = shm; \*s != '\0'; s++)

    {

        putchar(\*s);

    }

    putchar('\n');

    /\* Finally, change the first character of the segment to '\*',

    indicating

    we have read the segment. \*/

    \*shm = '\*';

    for (s = shm; \*s != '\0'; s++)

    {

        putchar(\*s);

    }

    putchar('\n');

    /\* Detach the shared memory segment. \*/

    if (shmdt(shm) == -1)

    {

        perror("shmdt");

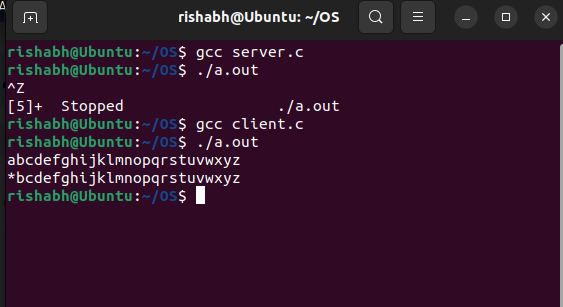
        exit(1);

    }

    exit(0);

}

OUTPUT:

  
  
4) By using system calls in Linux we will write the c program to implement to provide the communication using shared memory.

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <unistd.h>

#include <string.h>

#include <stdio.h>

#include <errno.h>

int main(void)

{

    pid\_t pid;

    int \*shared;

    int shmid;

    shmid = shmget(IPC\_PRIVATE, sizeof(int), IPC\_CREAT | 0666);

    printf("Shared Memory ID=%d\n", shmid);

    if ((pid = fork()) == 0)

    {

        shared = shmat(shmid, (void \*)0, 0);

        printf("Child pointer %p\n", shared);

        \*shared = 1;

        printf("Child value=%d\n", \*shared);

        sleep(2);

        printf("Child value=%d\n", \*shared);

    }

    else if (pid > 0)

    {

        shared = shmat(shmid, (void \*)0, 0);

        printf("Parent pointer %p\n", shared);

        printf("Parent value=%d\n", \*shared);

        sleep(1);

        \*shared = 42;

        printf("Parent value=%d\n", \*shared);

        sleep(5);

        shmctl(shmid, IPC\_RMID, 0);

    }

    else

    {

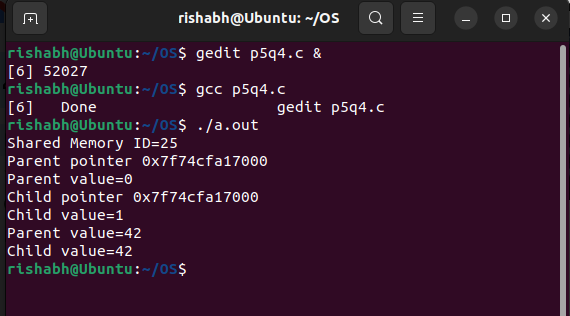
        perror("fork");

        return 1;

    }

    return 0;

}

OUTPUT:  
  
  
5) SHARING MEMORY BETWEEN PROCESSES   
I) Simple execution

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/shm.h>

int main()

{

    int shmid, status;

    int \*a, \*b;

    int i;

    shmid = shmget(IPC\_PRIVATE, 2 \* sizeof(int), 0777 | IPC\_CREAT);

    if (fork() == 0)

    {

        // Child Process

        printf("\nChild Process\n");

        b = (int \*)shmat(shmid, 0, 0);

        for (i = 0; i < 10; i++)

        {

            sleep(1);

            printf("\t\tChild reads %d,%d\n", b[0], b[1]);

        }

        shmdt(b);

    }

    else

    {

        // Parent Process

        a = (int \*)shmat(shmid, 0, 0);

        a[0] = 0;

        a[1] = 1;

        for (i = 0; i < 10; i++)

        {

            sleep(1);

            a[0] = a[0] + a[1];

            a[1] = a[0] + a[1];

            printf("\t\tParent writes %d,%d\n", a[0], a[1]);

        }

        wait(&status);

        shmdt(a);

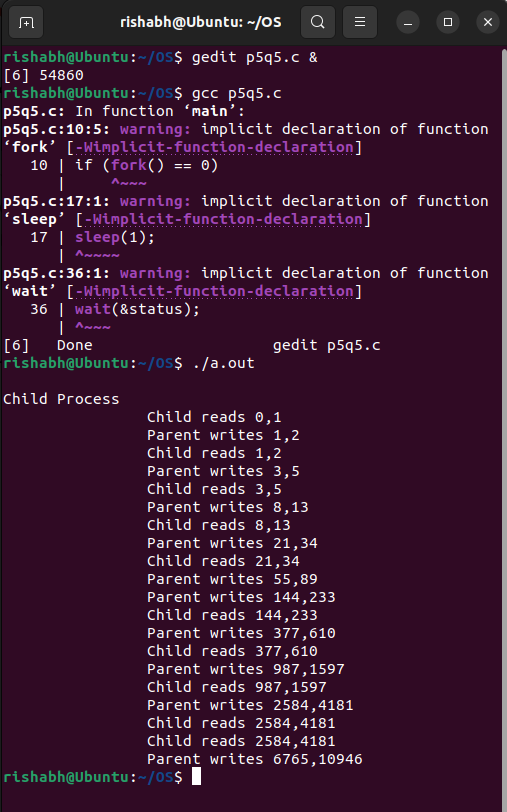
        shmctl(shmid, IPC\_RMID, 0);

    }

    return 0;

}

OUTPUT:

  
II)  
#include <stdio.h>

#include <sys/ipc.h>

#include <sys/shm.h>

int main()

{

    int shmid, status;

    int \*a, \*b;

    int i;

    shmid = shmget(IPC\_PRIVATE, 2 \* sizeof(int), 0777 | IPC\_CREAT);

    if (fork() == 0)

    {

        // Child Process

        printf("\nChild Process\n");

        b = (int \*)shmat(shmid, 0, 0);

        for (i = 0; i < 10; i++)

        {

            sleep(1);

            printf("\t\tChild reads %d,%d\n", b[0], b[1]);

        }

        shmdt(b);

    }

    else

    {

        // Parent Process

        a = (int \*)shmat(shmid, 0, 0);

        a[0] = 0;

        a[1] = 1;

        for (i = 0; i < 10; i++)

        {

            printf("\nParent Process\n");

            sleep(1);

            printf("\nAfter sleep\n");

            a[0] = a[0] + a[1];

            a[1] = a[0] + a[1];

            printf("\t\tParent writes %d,%d\n", a[0], a[1]);

        }

        wait(&status);

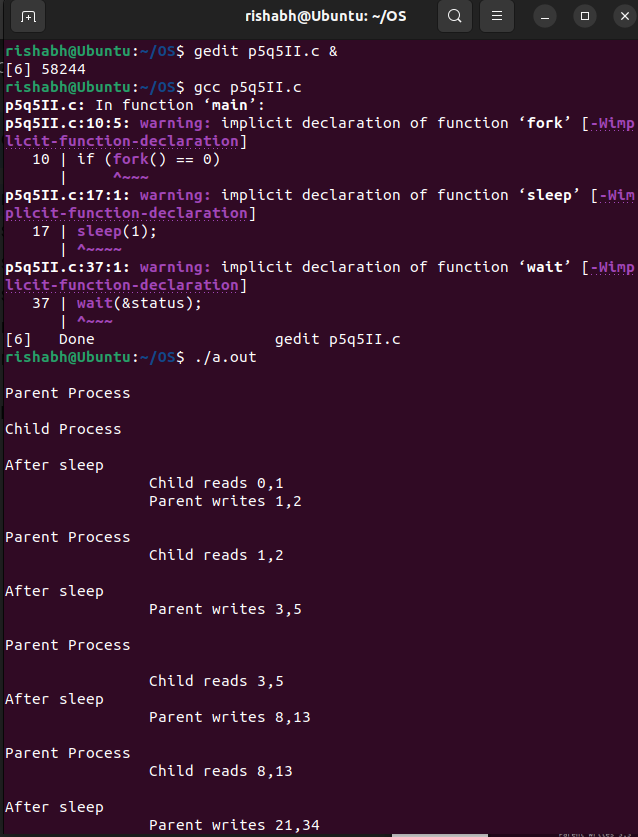
        shmdt(a);

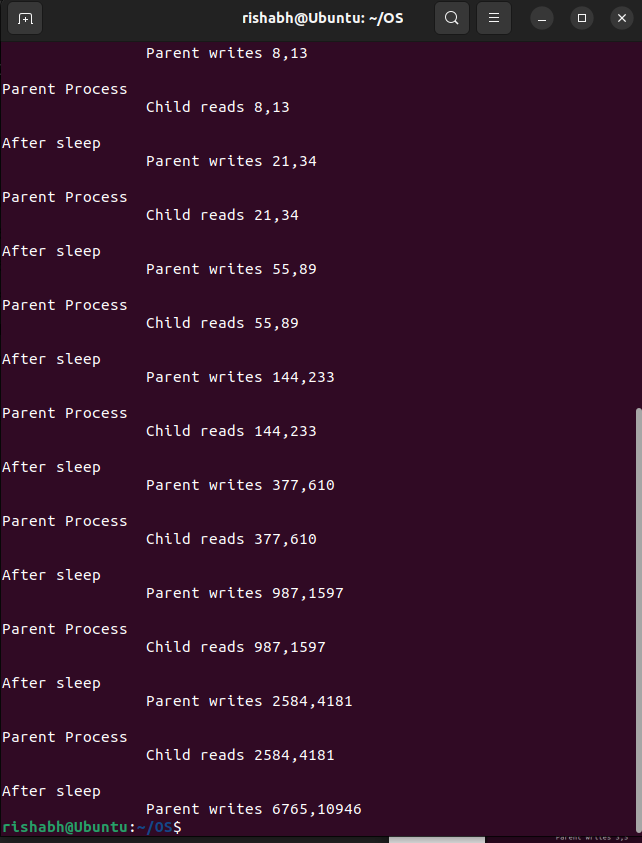
        shmctl(shmid, IPC\_RMID, 0);

    }

    return 0;

}

OUTPUT:  


  
  
III) Modify the sleep in the child process to sleep(2).

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/shm.h>

int main()

{

    int shmid, status;

    int \*a, \*b;

    int i

        shmid = shmget(IPC\_PRIVATE, 2 \* sizeof(int), 0777 | IPC\_CREAT);

    if (fork() == 0)

    {

        // Child Process

        b = (int \*)shmat(shmid, 0, 0);

        for (i = 0; i < 10; i++)

        {

            printf("\nChild Process\n");

            sleep(2);

            printf("\t\tChild reads %d,%d\n", b[0], b[1]);

        }

        shmdt(b);

    }

    else

    {

        // Parent Process

        a = (int \*)shmat(shmid, 0, 0);

        a[0] = 0;

        a[1] = 1;

        for (i = 0; i < 10; i++)

        {

            printf("\nParent Process\n");

            sleep(1);

            a[0] = a[0] + a[1];

            a[1] = a[0] + a[1];

            printf("\t\tParent writes %d,%d\n", a[0], a[1]);

        }

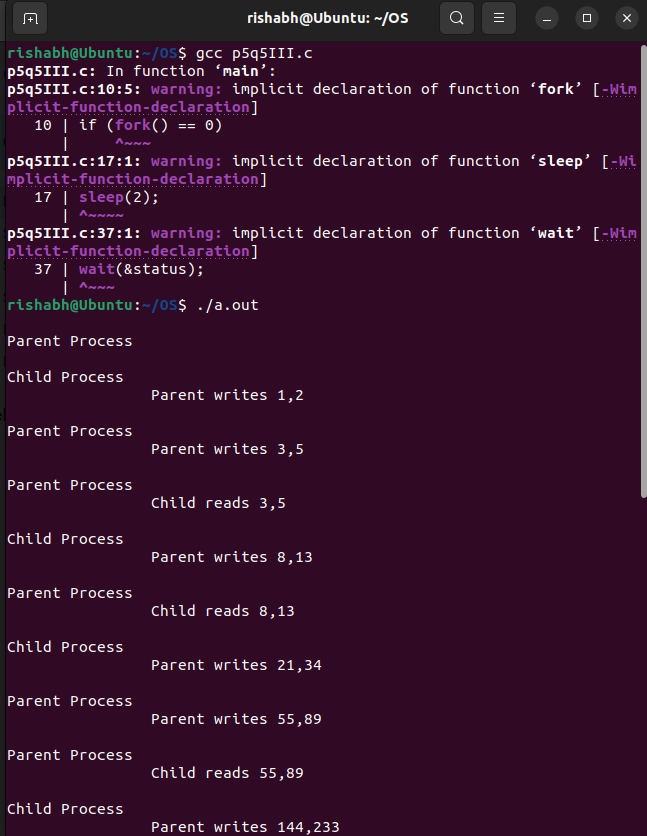
        wait(&status);

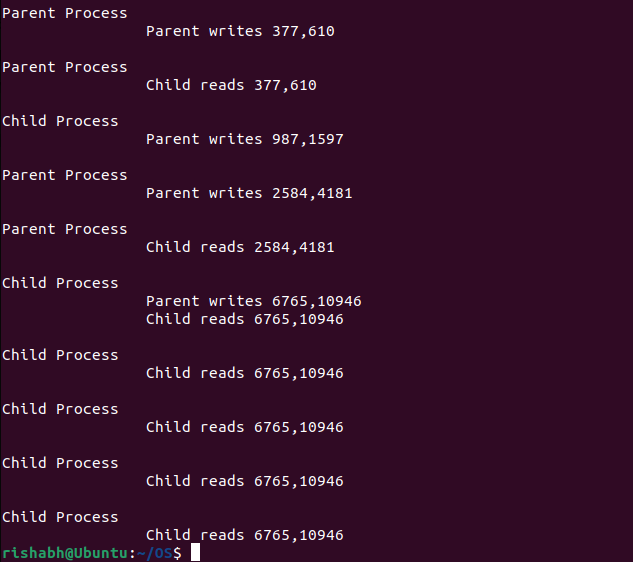
        shmdt(a);

        shmctl(shmid, IPC\_RMID, 0);

    }

}

OUTPUT:  




IV) Restore the sleep in the child process to sleep(1) and modify the sleep in the parent process to sleep(2).

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/shm.h>

int main()

{

    int shmid, status;

    int \*a, \*b;

    int i;

    shmid = shmget(IPC\_PRIVATE, 2 \* sizeof(int), 0777 | IPC\_CREAT);

    if (fork() == 0)

    {

        // Child Process

        b = (int \*)shmat(shmid, 0, 0);

        for (i = 0; i < 10; i++)

        {

            printf("\nChild Process\n");

            sleep(1);

            printf("\t\tChild reads %d,%d\n", b[0], b[1]);

        }

        shmdt(b);

    }

    else

    {

        // Parent Process

        a = (int \*)shmat(shmid, 0, 0);

        a[0] = 0;

        a[1] = 1;

        for (i = 0; i < 10; i++)

        {

            printf("\nParent Process\n");

            sleep(2);

            a[0] = a[0] + a[1];

            a[1] = a[0] + a[1];

            printf("\t\tParent writes %d,%d\n", a[0], a[1]);

        }

        wait(&status);

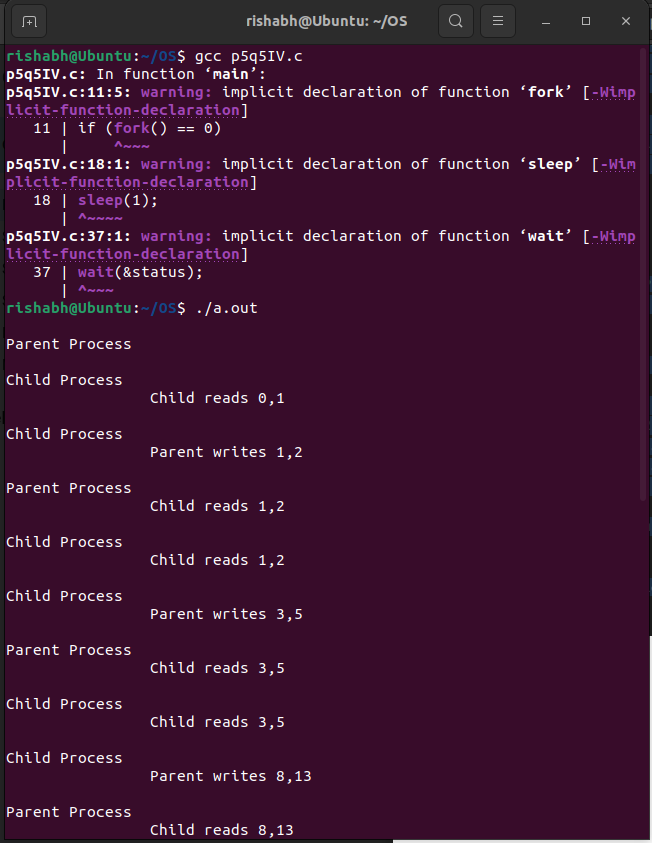
        shmdt(a);

        shmctl(shmid, IPC\_RMID, 0);

    }

}

OUTPUT:

   
  
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 Modify A.c and B.c to implement this.   
  
Submission : Name your programs A\_1.c and B\_1.c for the first part of the assignment, and A\_2.c and B\_2.c for the second part.  
  
Part 1 and 2   
A.c

#include <stdio.h>

#include <stdlib.h>

#include <sys/shm.h>

#include <unistd.h>

int main()

{

    int count = 0, n;

    key\_t key = ftok("inpfile", 1);

    int shmid = shmget(key, 10 \* sizeof(int), IPC\_CREAT | 0666);

    int \*array = (int \*)shmat(shmid, NULL, 0);

    FILE \*f = fopen("inpfile.txt", "r");

    if (f == NULL)

    {

        printf("File cannot be opened.\n");

        exit(1);

    }

    while (fscanf(f, "%d ", &n) == 1 && count < 10)

    {

        array[count] = n;

        count++;

    }

    fclose(f);

    printf("Read %d integers from A.c\n", count);

    sleep(10);

    shmdt(array);

    return 0;

}

B.c  
#include <stdio.h>

#include <stdlib.h>

#include <sys/shm.h>

#include <sys/ipc.h>

#include <unistd.h>

void sort(int A[], int m)

{

    int temp;

    for (int i = 0; i < m; i++)

    {

        for (int j = i + 1; j < m; j++)

        {

            if (A[j] < A[i])

            {

                temp = A[i];

                A[i] = A[j];

                A[j] = temp;

            }

        }

    }

}

int main()

{

    sleep(10);

    int count = 0;

    key\_t key = ftok("inpfile", 1);

    int shmid = shmget(key, 10 \* sizeof(int), IPC\_CREAT | 0666);

    int \*array = (int \*)shmat(shmid, NULL, 0);

    int \*done = array + 1;

    while (\*done == 0)

        usleep(1000);

    while (array[count] != 0 && count < 10)

    {

        count++;

    }

    sort(array, count);

    FILE \*f = fopen("outfile.txt", "w");

    if (f == NULL)

    {

        printf("File cannot be opened.\n");

        exit(1);

    }

}

OUTPUT:  
  
Here we can see that the input file is present and the output file with the desired output is created by the program.   
  
Result : Linux C programs to demonstrate the concept of Shared Memory has been implemented