Problem:- Server and worker interaction.

Code:-

Server.c

```
#include <stdio.h>      /* for printf(),..*/
#include <stdlib.h>     /* for rand(), exit(), .. */
#include <time.h>       /* for random time seed*/
#include <sys/types.h> /* for predefined structs like pid_t ...*/
#include <sys/ipc.h>    /* for shmget(), shmat() ,...*/
#include <sys/shm.h>    /* for shmget(), shmat() ,....*/
#include <unistd.h>     /* for usleep(), ...*/
#include <signal.h>     /* for signal hanlder to respond to ctrl-C*/

#define RANGE 500 // Range of array elements is from [0,499]
#define SIZE 20   // Size of array is 20

typedef struct
{
   int data[100]; /* The array to hold the numbers put by the server
process and sorted by one worker process */

   int size; /* The number of elements of the array data to be sorted. The
server process updates this field when putting the numbers in the array
data. */

   pid_t worker_pid; /* The pid of the worker process that is sorting (or
has sorted)  the size number of elements of the array data. */

   int status; /* status = 0 means that at present there is nothing in the
array data[ ] that needs to be sorted.
                  status = 1 means that the server process has put some
numbers in the array data[ ] that need to be sorted.
                  status = 2 means that the worker process having pid =
worker_pid has started sorting size number of elements of the array data[
].
                  status = 3 means that the worker process having pid =
worker_pid has sorted size number of elements of the array data[ ].
```

```
                    status = 4 means that the server process is using the
sorted elements of the array data[ ].
                    status = -1 means that the server process has ended. */

   int sequenceNo; /* stores the sequence number of the array */

} task;

task *t;

int shmid;

/* following is a signal handler for the keypress ctrl-c*/
typedef void (*sighandler_t)(int);

void releaseSHM(int signum)
{
   int status;
   // int shmctl(int shmid, int cmd, struct shmid_ds *buf); /* Read the
manual for shmctl() */
   status = shmctl(shmid, IPC_RMID, NULL); /* IPC_RMID is the command for
destroying the shared memory*/
   t->status = -1;                          // Server and workers should
stop working
   if (status == 0)
   {
       fprintf(stderr, "Remove shared memory with id=%d\n", shmid);
   }
   else if (status == -1)
   {
       fprintf(stderr, "Cannot remove shared memory with id=%d\n", shmid);
   }
   else
   {
       fprintf(stderr, "shmctl() returned wrong value while removing
shared memory with id=%d\n", shmid);
   }

   // int kill(pid_t pid,int sig)
   status = kill(0, SIGKILL);
```

```
    if (status == 0)
    {
        fprintf(stderr, "kill successful\n");
    }
    else if (status == -1)
    {
        perror("kill failed\n");
        fprintf(stderr, "Cannot remove shared memory with id=%d\n", shmid);
    }
    else
    {
        fprintf(stderr, "kill(2) returned wrong value\n");
    }
}

int main(int argc, char *argv[])
{
    int start = 1;      // maintaining sequence number of array, to keep
track which worker is sorting which array
    srand(time(NULL)); // For random seed
    sighandler_t shandler;
    /* install signal handler */
    // sighandler_t signal(int signum, sighandler_t handler);
    shandler = signal(SIGINT, releaseSHM); /* should we call this
seperately in parent and child process */
    key_t key = ftok("/tmp/", 4);
    if (key == -1)
    {
        perror("ftok() error at server \n");
        exit(0);
    }
    shmid = shmget(key, sizeof(task), IPC_CREAT | 0777);
    if (shmid == -1)
    {
        perror("Server---> shmget(): error\n");
        exit(0);
    }
    t = (task *)shmat(shmid, NULL, 0);
    if (t == (void *)-1)
    {
```

```
        perror("shmat() error at server");
        exit(0);
    }
    /* Initialise task */
    t->status = 0;

    while (t->status != -1)
    {
        // Put values
        if (t->status == 0)
        {
            // int totalNumbers = rand() % 101;
            printf("The number of elements in the data array is %d\n",
SIZE);
            t->size = SIZE;
            t->sequenceNo = start++;
            for (int i = 0; i < SIZE; i++)
            {
                t->data[i] = rand() % RANGE;
            }
            t->status = 1;
            printf("The initial unsorted array [%d] is:\n", t->sequenceNo);
            for (int i = 0; i < SIZE; i++)
            {
                printf("%d ", t->data[i]);
            }
            printf("\n");
        }
        // Wait for the status, i.e from worker side, status should be put
to 3
        else if (t->status == 3)
        {
            t->status = 4;
            printf("The array has been sorted by worker with pid %d\n",
t->worker_pid);
            printf("The sorted array [%d] is :\n", t->sequenceNo);
            for (int i = 0; i < t->size; i++)
            {
                printf("%d ", t->data[i]);
            }
```

```
            printf("\n\n");
        }
        else if (t->status == 4)
        {
            t->status = 0;
        }
        usleep(10000); // 10 ms --> server feeds data fastly
    }
    return 0;
}
```

worker.c

```
#include <stdio.h>      /* for printf(),..*/
#include <stdlib.h>     /* for rand(), exit(), .. */
#include <time.h>       /* for random time seed*/
#include <sys/types.h> /* for predefined structs like pid_t ...*/
#include <sys/ipc.h>    /* for shmget(), shmat() ,...*/
#include <sys/shm.h>    /* for shmget(), shmat() ,....*/
#include <unistd.h>     /* for usleep(), ...*/

typedef struct
{
   int data[100]; /* The array to hold the numbers put by the server
process and sorted by one worker process */

   int size; /* The number of elements of the array data to be sorted. The
server process updates this field when putting the numbers in the array
data. */

   pid_t worker_pid; /* The pid of the worker process that is sorting (or
has sorted)  the size number of elements of the array data. */

   int status; /* status = 0 means that at present there is nothing in the
array data[ ] that needs to be sorted.
                  status = 1 means that the server process has put some
numbers in the array data[ ] that need to be sorted.
                  status = 2 means that the worker process having pid =
worker_pid has started sorting size number of elements of the array data[
].
```

```
                    status = 3 means that the worker process having pid =
worker_pid has sorted size number of elements of the array data[ ].
                    status = 4 means that the server process is using the
sorted elements of the array data[ ].
                    status = -1 means that the server process has ended. */

   int sequenceNo; /* stores the sequence number of the array */
} task;

task *t; // pointer to a task type structure

int shmid; // shmid used to attach and release shared memory

// Comparator function for sorting
int cmp(const void *a, const void *b)
{
   return (*(int *)a - *(int *)b);
}

int main(int argc, char *argv[])
{
   task *t;

   // Key generation using ftok() to be used to generate shmid using
shmget
   key_t key = ftok("/tmp/", 4);
   if (key == -1)
   {
       perror("ftok() error at worker\n");
       exit(0);
   }
   shmid = shmget(key, sizeof(task), 0777);
   if (shmid == -1)
   {
       perror("Worker---> shmget(): error\n");
       exit(0);
   }
   // Attaching shared memory
   t = (task *)shmat(shmid, NULL, 0);
   if (t == (void *)-1)
```

```
    {
        perror("shmat() error at client");
        exit(-1);
    }
    while (t->status != -1) // Monitoring status of the shared segment
    {
        if (t->status == 1) // Server has already put the data in the
segment, so worker can do its job
        {
            printf("Worker with pid %d is sorting the array [%d]\n",
getpid(), t->sequenceNo);
            t->status = 2; // before sort
            qsort(t->data, t->size, sizeof(int), cmp);
            t->status = 3; // done sorting
            t->worker_pid = getpid();
        }
        usleep(200000); // 200 ms --> worker responds slowly to give chance
to other workers
    }
    // t->status = -1 , i.e, server has released shared memory segment
    printf("Server has stopped responding\n");
    return 0;
}
```

Ouput:-

Server Side:-

```
The number of elements in the data array is 20
The initial unsorted array [31] is:
482 432 406 85 189 493 220 410 218 466 307 261 84 288 336 408 319 179 42 107
The array has been sorted by worker with pid 10336
The sorted array [31] is :
42 84 85 107 179 189 218 220 261 288 307 319 336 406 408 410 432 466 482 493

The number of elements in the data array is 20
The initial unsorted array [32] is:
162 134 190 129 430 221 297 475 476 261 373 459 45 131 396 235 124 116 145 194
The array has been sorted by worker with pid 10335
The sorted array [32] is :
45 116 124 129 131 134 145 162 190 194 221 235 261 297 373 396 430 459 475 476

The number of elements in the data array is 20
The initial unsorted array [33] is:
83 305 456 19 93 144 427 264 175 470 371 337 456 413 466 239 135 115 214 111
The array has been sorted by worker with pid 10336
The sorted array [33] is :
19 83 93 111 115 135 144 175 214 239 264 305 337 371 413 427 456 456 466 470

The number of elements in the data array is 20
The initial unsorted array [34] is:
228 87 422 273 218 318 8 194 287 6 241 222 311 197 241 256 341 21 20 16
The array has been sorted by worker with pid 10335
The sorted array [34] is :
6 8 16 20 21 87 194 197 218 222 228 241 241 256 273 287 311 318 341 422

The number of elements in the data array is 20
The initial unsorted array [35] is:
343 243 353 151 156 171 390 143 138 104 255 366 43 29 139 261 200 0 308 487
The array has been sorted by worker with pid 10336
The sorted array [35] is :
0 29 43 104 138 139 143 151 156 171 200 243 255 261 308 343 353 366 390 487

The number of elements in the data array is 20
The initial unsorted array [36] is:
358 49 61 169 246 154 277 439 27 149 307 370 392 12 22 400 183 412 44 173
The array has been sorted by worker with pid 10335
The sorted array [36] is :
12 22 27 44 49 61 149 154 169 173 183 246 277 307 358 370 392 400 412 439

The number of elements in the data array is 20
The initial unsorted array [37] is:
369 151 39 412 180 30 26 380 30 334 219 388 383 280 409 481 435 38 272 462
^CRemove shared memory with id=62
Killed
```

Worker1 side:-

```
→  Assignment05 |main) ✗ ./worker
Worker with pid 10335 is sorting the array [1]
Worker with pid 10335 is sorting the array [2]
Worker with pid 10335 is sorting the array [3]
Worker with pid 10335 is sorting the array [4]
Worker with pid 10335 is sorting the array [6]
Worker with pid 10335 is sorting the array [8]
Worker with pid 10335 is sorting the array [10]
Worker with pid 10335 is sorting the array [12]
Worker with pid 10335 is sorting the array [14]
Worker with pid 10335 is sorting the array [16]
Worker with pid 10335 is sorting the array [18]
Worker with pid 10335 is sorting the array [20]
Worker with pid 10335 is sorting the array [22]
Worker with pid 10335 is sorting the array [24]
Worker with pid 10335 is sorting the array [26]
Worker with pid 10335 is sorting the array [28]
Worker with pid 10335 is sorting the array [30]
Worker with pid 10335 is sorting the array [32]
Worker with pid 10335 is sorting the array [34]
Worker with pid 10335 is sorting the array [36]
Server has stopped responding
```

Worker2 side:-

```
→  Assignment05 |main) ✗ ./worker
Worker with pid 10336 is sorting the array [5]
Worker with pid 10336 is sorting the array [7]
Worker with pid 10336 is sorting the array [9]
Worker with pid 10336 is sorting the array [11]
Worker with pid 10336 is sorting the array [13]
Worker with pid 10336 is sorting the array [15]
Worker with pid 10336 is sorting the array [17]
Worker with pid 10336 is sorting the array [19]
Worker with pid 10336 is sorting the array [21]
Worker with pid 10336 is sorting the array [23]
Worker with pid 10336 is sorting the array [25]
Worker with pid 10336 is sorting the array [27]
Worker with pid 10336 is sorting the array [29]
Worker with pid 10336 is sorting the array [31]
Worker with pid 10336 is sorting the array [33]
Worker with pid 10336 is sorting the array [35]
Server has stopped responding
```

Explanation:-

Here , on the server side we are feeding data into the shared memory segment after 10ms (very fast), whereas the worker side is responding to the segment slowly 200ms, giving a chance to the other workers to enter the critical section resulting in division of work among the workers.