Operating Systems and Kernel Design

Lab Assignment -3 Report

Pavan Kumar Revanuru

01578776

Due Date: 11/03/2016

Objective:

The objective of the assignment is to implement Memory Management System that handles all requests of allocation of memory space by different threads.

Functions and methods used:

pthread_mutex_lock():

The mutex object referenced by mutex shall be locked by calling pthread_mutex_lock(). If the mutex is already locked, the calling thread shall block until the mutex becomes available. This operation shall return with the mutex object referenced by mutex in the locked state with the calling thread as its owner.

Pthread_mutex_unlock():

The pthread_mutex_unlock() function shall release the mutex object referenced by mutex. The manner in which a mutex is released is dependent upon the mutex's type attribute. If there are threads blocked on the mutex object referenced by mutex when pthread_mutex_unlock() is called, resulting in the mutex becoming available.

atoi:

This function converts the string to an integer. This function is used in the Program 2 while taking the input from the command line.

MMS():

This function acts like a main memory in the code. It assigns the memory block in random for every thread that is created. It even calls which fit function to execute (First fit, Best fit and Worst fit). All the memory information of the block such as if the block memory is used or not is verified in the MMS function.

FirstFit():

This function executes the First Fit algorithm that was discussed.

BestFit():

This function executes the Best Fit algorithm that was discussed.

WorstFit():

This function executes the Worst Fit algorithm that was discussed.

Malloc():

Malloc function allocates the memory of space that was requested by the user.

Background of the assignment:

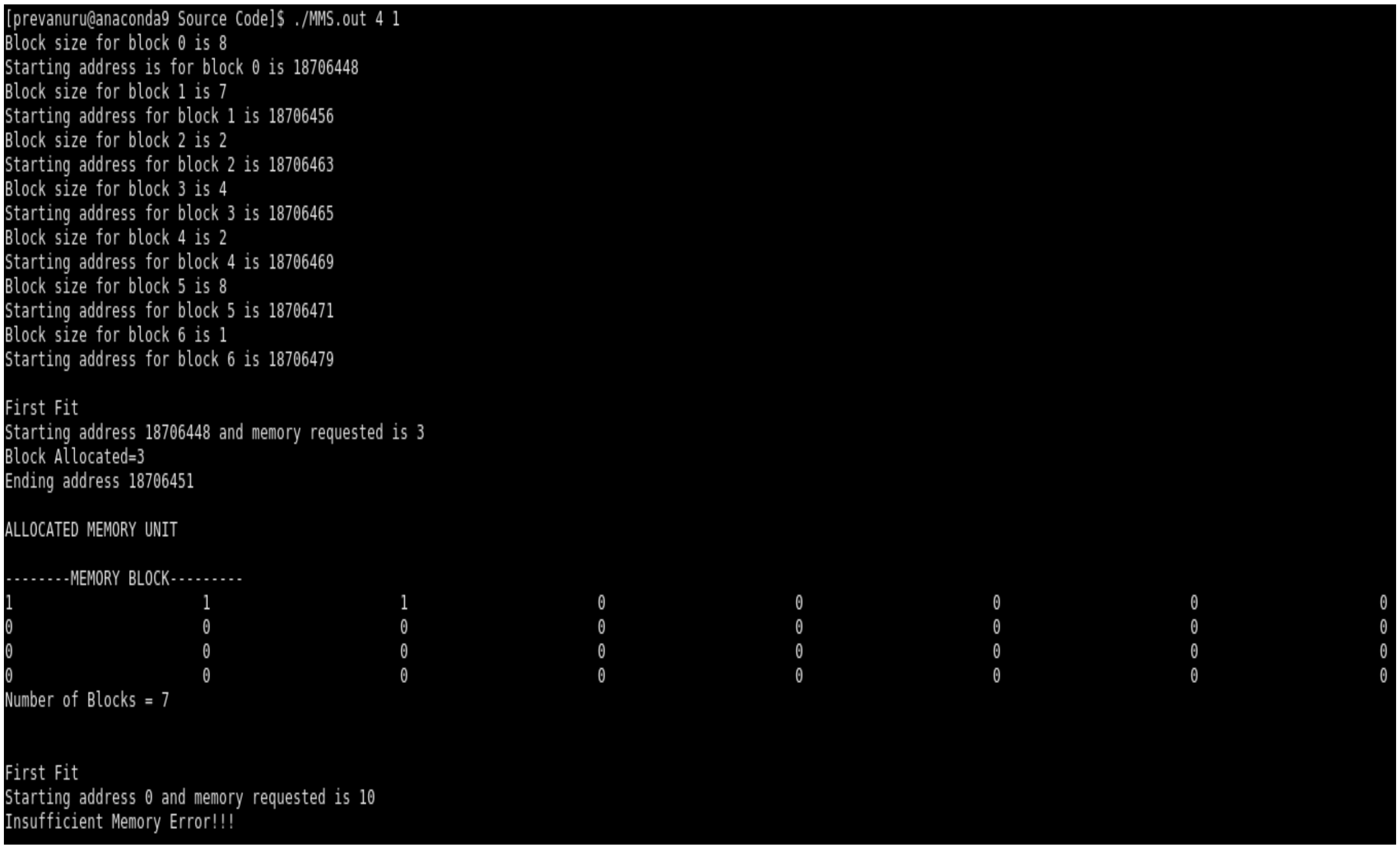
The assignment focuses on creating many threads using pthreads and executing them. Each thread will ask for a memory block in random and if there’s any available block according to the algorithm selected, the memory will be allocated. Else, it will go for other thread which has requested the memory.

Results:

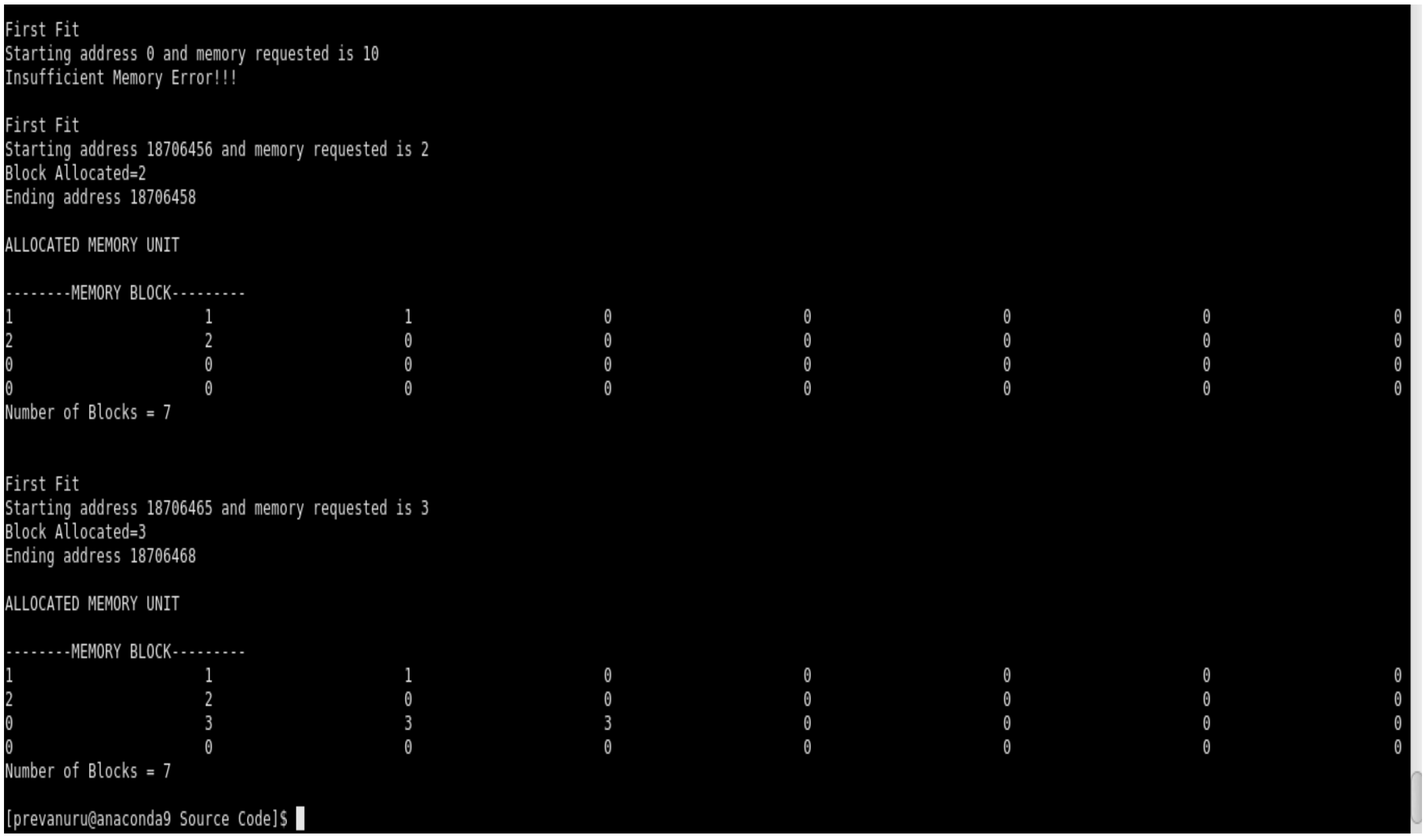
First Fit Alogorithm:

./MMS.out 4 1

4 here are the number of threads and 1 is the FirstFit Algorithm.



```
[prevanuru@anaconda9 Source Code]$ ./MMS.out 4 1
Block size for block 0 is 8
Starting address is for block 0 is 18706448
Block size for block 1 is 7
Starting address for block 1 is 18706456
Block size for block 2 is 2
Starting address for block 2 is 18706463
Block size for block 3 is 4
Starting address for block 3 is 18706465
Block size for block 4 is 2
Starting address for block 4 is 18706469
Block size for block 5 is 8
Starting address for block 5 is 18706471
Block size for block 6 is 1
Starting address for block 6 is 18706479

First Fit
Starting address 18706448 and memory requested is 3
Block Allocated=3
Ending address 18706451

ALLOCATED MEMORY UNIT

--------MEMORY BLOCK---------
1               1               1               0               0               0               0               0
0               0               0               0               0               0               0               0
0               0               0               0               0               0               0               0
0               0               0               0               0               0               0               0
Number of Blocks = 7


First Fit
Starting address 0 and memory requested is 10
Insufficient Memory Error!!!
```



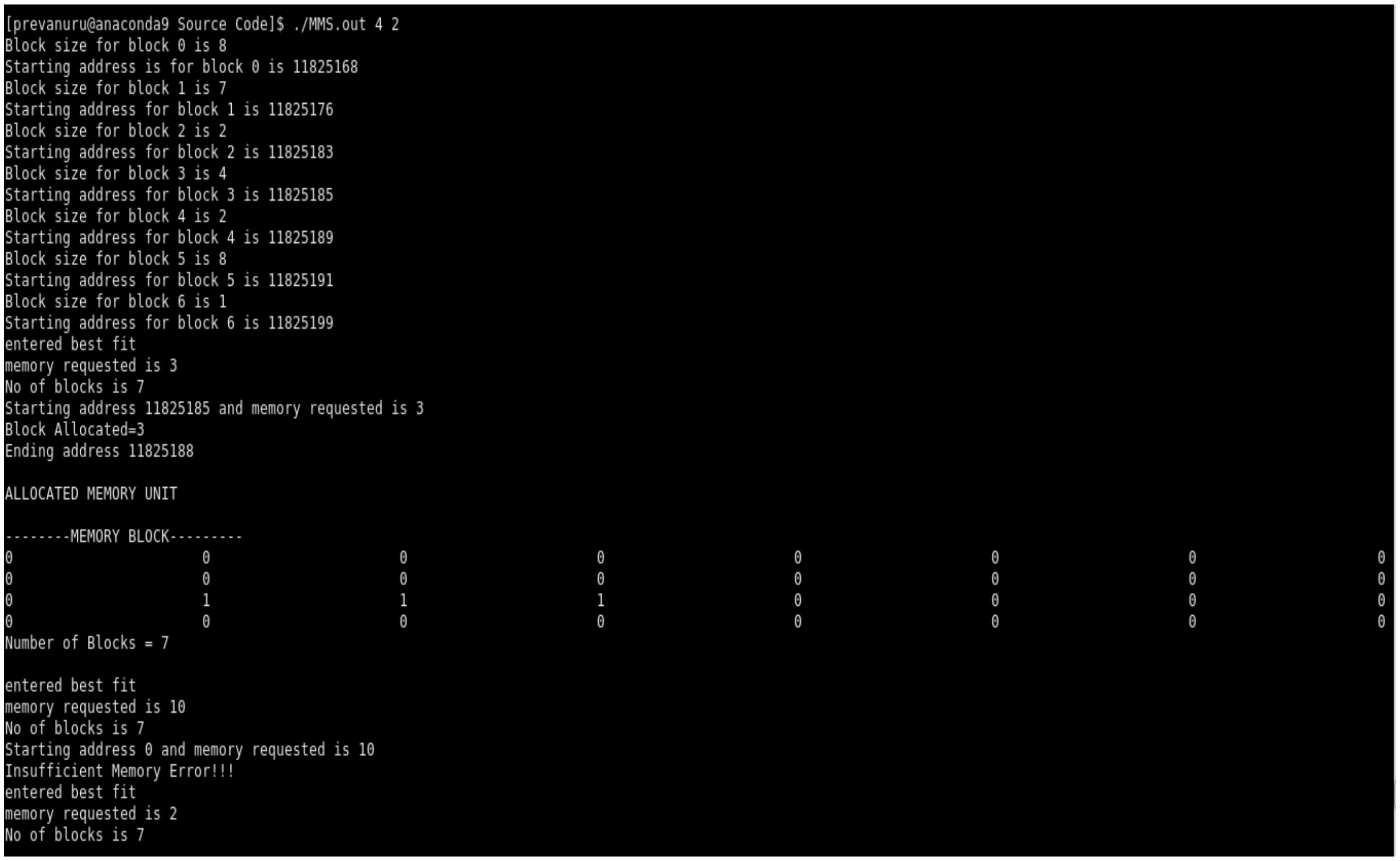
```
First Fit
Starting address 0 and memory requested is 10
Insufficient Memory Error!!!

First Fit
Starting address 18706456 and memory requested is 2
Block Allocated=2
Ending address 18706458

ALLOCATED MEMORY UNIT

--------MEMORY BLOCK---------
1                   1                   1                   0                   0                   0                   0                   0
2                   2                   0                   0                   0                   0                   0                   0
0                   0                   0                   0                   0                   0                   0                   0
0                   0                   0                   0                   0                   0                   0                   0
Number of Blocks = 7


First Fit
Starting address 18706465 and memory requested is 3
Block Allocated=3
Ending address 18706468

ALLOCATED MEMORY UNIT

--------MEMORY BLOCK---------
1                   1                   1                   0                   0                   0                   0                   0
2                   2                   0                   0                   0                   0                   0                   0
0                   3                   3                   3                   0                   0                   0                   0
0                   0                   0                   0                   0                   0                   0                   0
Number of Blocks = 7

[prevanuru@anaconda9 Source Code]$
```

<u>Best Fit Alogorithm:</u>

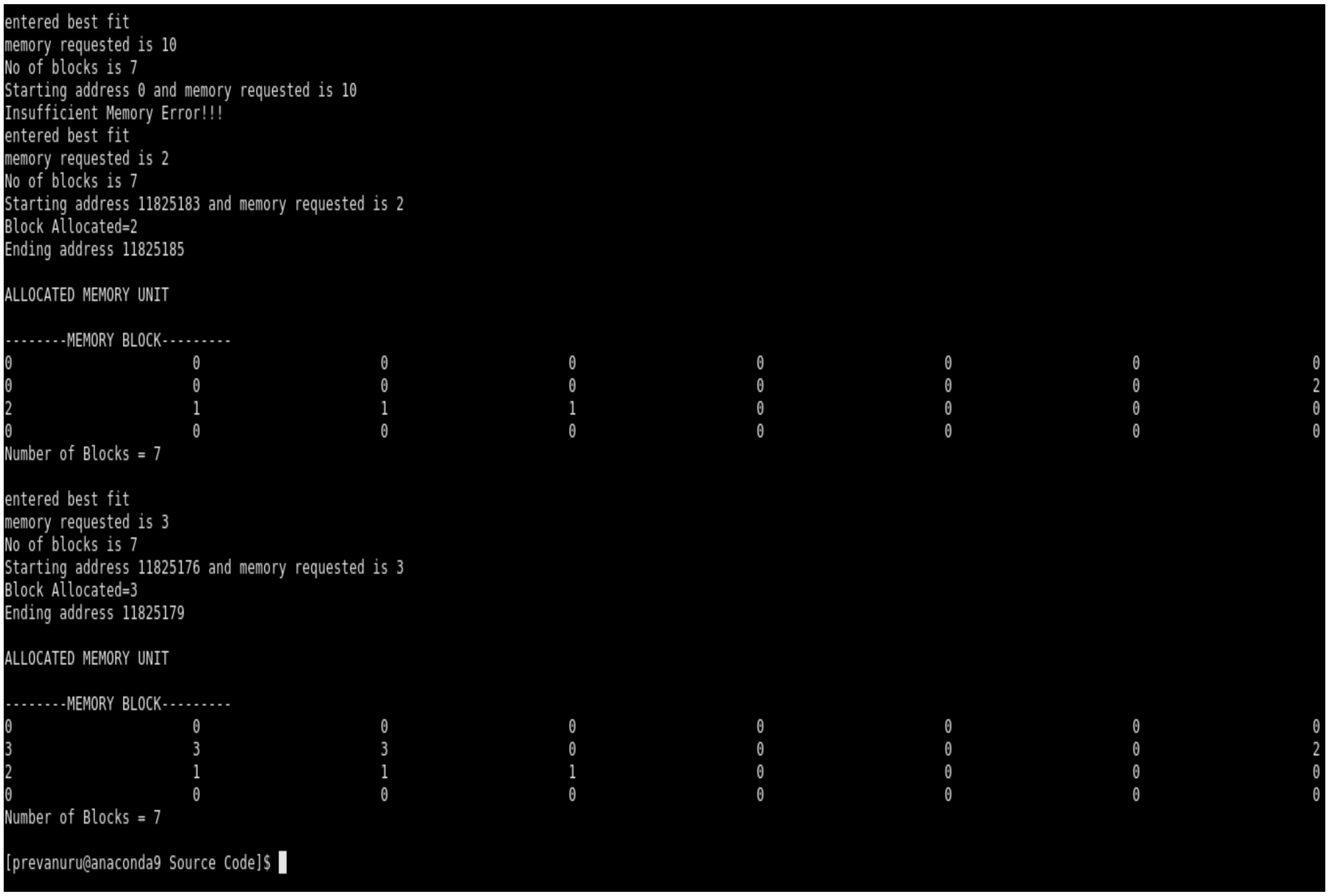
./MMS.out 4 2

4 here are the number of threads and 2 is the Best Fit Algorithm.

<u>(Contd.)</u>



```
[prevanuru@anaconda9 Source Code]$ ./MMS.out 4 2
Block size for block 0 is 8
Starting address is for block 0 is 11825168
Block size for block 1 is 7
Starting address for block 1 is 11825176
Block size for block 2 is 2
Starting address for block 2 is 11825183
Block size for block 3 is 4
Starting address for block 3 is 11825185
Block size for block 4 is 2
Starting address for block 4 is 11825189
Block size for block 5 is 8
Starting address for block 5 is 11825191
Block size for block 6 is 1
Starting address for block 6 is 11825199
entered best fit
memory requested is 3
No of blocks is 7
Starting address 11825185 and memory requested is 3
Block Allocated=3
Ending address 11825188

ALLOCATED MEMORY UNIT

--------MEMORY BLOCK---------
0               0               0               0               0               0               0               0
0               0               0               0               0               0               0               0
0               1               1               1               0               0               0               0
0               0               0               0               0               0               0               0
Number of Blocks = 7

entered best fit
memory requested is 10
No of blocks is 7
Starting address 0 and memory requested is 10
Insufficient Memory Error!!!
entered best fit
memory requested is 2
No of blocks is 7
```

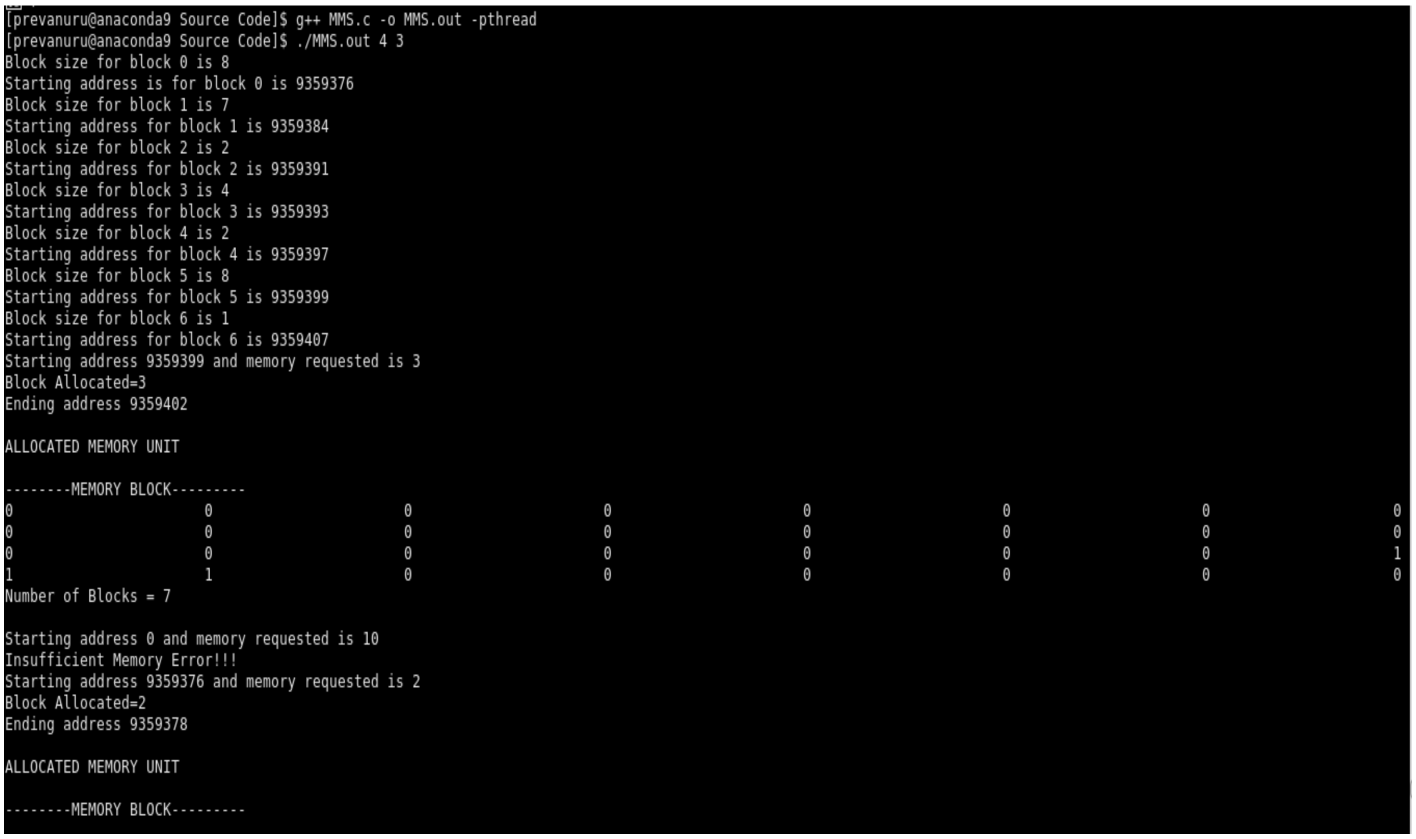


```
entered best fit
memory requested is 10
No of blocks is 7
Starting address 0 and memory requested is 10
Insufficient Memory Error!!!
entered best fit
memory requested is 2
No of blocks is 7
Starting address 11825183 and memory requested is 2
Block Allocated=2
Ending address 11825185

ALLOCATED MEMORY UNIT

--------MEMORY BLOCK---------
0               0               0               0               0               0               0               0
0               0               0               0               0               0               0               2
2               1               1               1               0               0               0               0
0               0               0               0               0               0               0               0
Number of Blocks = 7

entered best fit
memory requested is 3
No of blocks is 7
Starting address 11825176 and memory requested is 3
Block Allocated=3
Ending address 11825179

ALLOCATED MEMORY UNIT

--------MEMORY BLOCK---------
0               0               0               0               0               0               0               0
3               3               3               0               0               0               0               2
2               1               1               1               0               0               0               0
0               0               0               0               0               0               0               0
Number of Blocks = 7

[prevanuru@anaconda9 Source Code]$ 
```

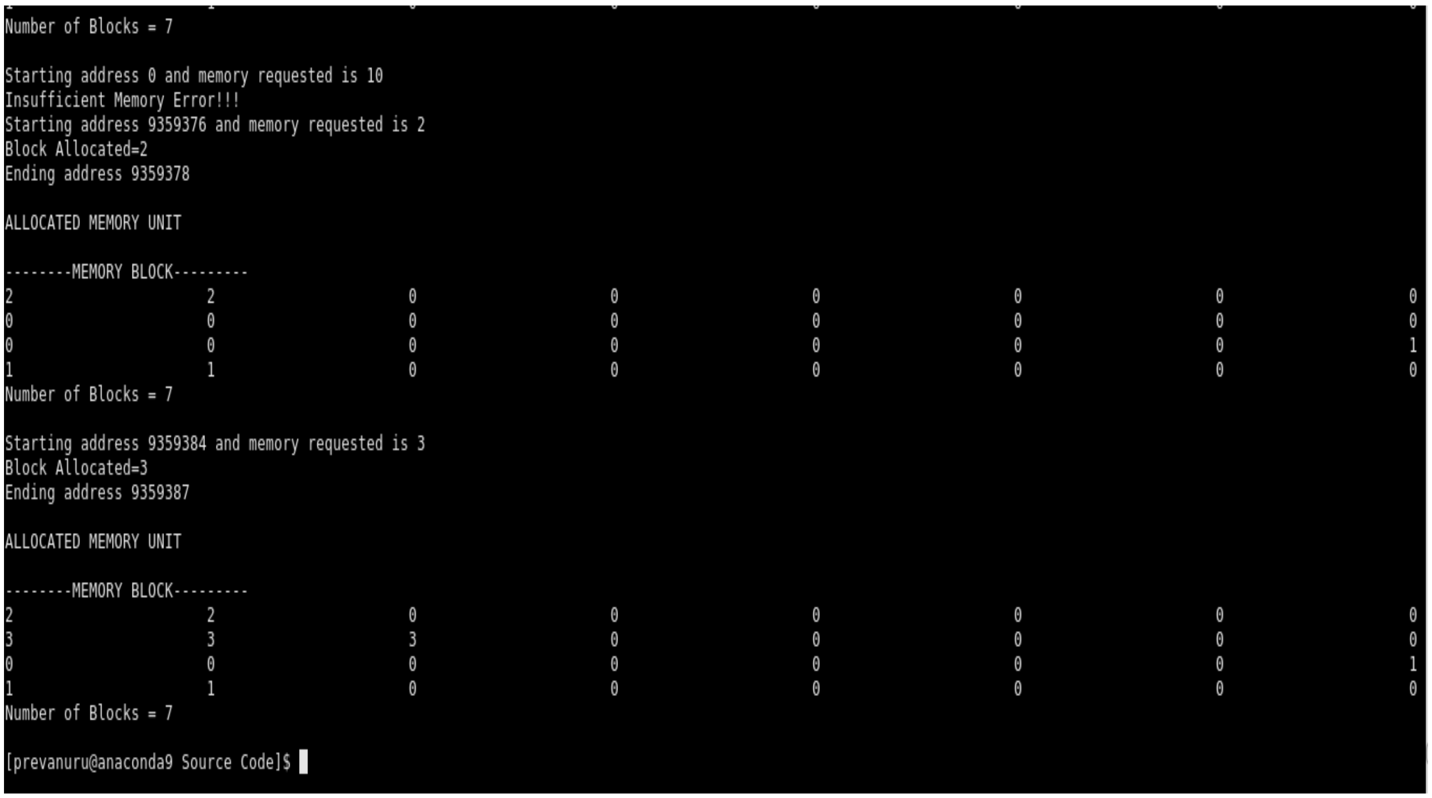
Worst Fit Alogorithm:

./MMS.out 4 3

4 here are the number of threads and 3 is the Worst Fit Algorithm.



```
[prevanuru@anaconda9 Source Code]$ g++ MMS.c -o MMS.out -pthread
[prevanuru@anaconda9 Source Code]$ ./MMS.out 4 3
Block size for block 0 is 8
Starting address is for block 0 is 9359376
Block size for block 1 is 7
Starting address for block 1 is 9359384
Block size for block 2 is 2
Starting address for block 2 is 9359391
Block size for block 3 is 4
Starting address for block 3 is 9359393
Block size for block 4 is 2
Starting address for block 4 is 9359397
Block size for block 5 is 8
Starting address for block 5 is 9359399
Block size for block 6 is 1
Starting address for block 6 is 9359407
Starting address 9359399 and memory requested is 3
Block Allocated=3
Ending address 9359402

ALLOCATED MEMORY UNIT

--------MEMORY BLOCK---------
0                  0                  0                  0                  0                  0                  0                  0
0                  0                  0                  0                  0                  0                  0                  0
0                  0                  0                  0                  0                  0                  0                  1
1                  1                  0                  0                  0                  0                  0                  0
Number of Blocks = 7

Starting address 0 and memory requested is 10
Insufficient Memory Error!!!
Starting address 9359376 and memory requested is 2
Block Allocated=2
Ending address 9359378

ALLOCATED MEMORY UNIT

--------MEMORY BLOCK---------
```



```
Number of Blocks = 7

Starting address 0 and memory requested is 10
Insufficient Memory Error!!!
Starting address 9359376 and memory requested is 2
Block Allocated=2
Ending address 9359378

ALLOCATED MEMORY UNIT

--------MEMORY BLOCK---------
2                  2                  0                  0                  0                  0                  0                  0
0                  0                  0                  0                  0                  0                  0                  0
0                  0                  0                  0                  0                  0                  0                  1
1                  1                  0                  0                  0                  0                  0                  0
Number of Blocks = 7

Starting address 9359384 and memory requested is 3
Block Allocated=3
Ending address 9359387

ALLOCATED MEMORY UNIT

--------MEMORY BLOCK---------
2                  2                  0                  0                  0                  0                  0                  0
3                  3                  3                  0                  0                  0                  0                  0
0                  0                  0                  0                  0                  0                  0                  1
1                  1                  0                  0                  0                  0                  0                  0
Number of Blocks = 7

[prevanuru@anaconda9 Source Code]$
```

Observations:

In this assignment, we create threads as dictated by the program and execute them accordingly by selecting one of the three fit algorithms available: First fit, Best fit and Worst fit. The program creates 5 blocks of memory selected in random and stores the chunk addresses. When the thread requests for the block, the MMS checks if the requested data is below the available block size. If yes, it will allocate the block as per the algorithm and if the requested block has a size more than the available size, the request is rejected and moved onto the next thread request. In the end if there is a defragmentation, the rejected block may get a chance to handle one of the available block.

Conclusion:

The program executed will create blocks of empty data and threads created request these empty block to put data. The data is set to the block according to the algorithm selected.

Note: The program is hard coded to divide the partition into 5 parts. It will run fine with 5 threads. If there are more than 5 threads executed then there would be a segmentation fault as the blocks are already allocated.

Source Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <pthread.h>

#include <unistd.h>

#include <sys/types.h>

#include <iostream>

#include <semaphore.h>

#define Max_No_of_Threads 10

#define MAX_MEM_SIZE 32

#define Max_Thread_Count 5


//Stucture for threads

struct thread_Data

{

  int id;

  int size;

};
```

```
//Structure for memory block
struct Memory_Block
{
  char *blockStartAddress;
  int blockSize;
  int dataInBlock;
};



// global variable, all threads can acess
void *allocatedAddress;
char *StartAddr;
char *EndAddr;
thread_Data Threads[Max_No_of_Threads];
Memory_Block Blocks[Max_Thread_Count];
pthread_mutex_t Mutex;
int sw,No_of_Threads,block_Count,ThreadIndex;
sem_t bin_sem; // semaphore
pthread_mutex_t mutx; // mutex
void InitializeBlocks();
void Print_Memory();
void *MMS(void *arg);
char *ReqMem(int size);
char *FirstFit(int blockAllocationSize, int threadIndex);
char *BestFit(int blockAllocationSize, int threadIndex);
char *WorstFit(int blockAllocationSize, int threadIndex);

int main(int argc, char *argv[])
{
  int i;
  allocatedAddress = malloc(MAX_MEM_SIZE);
  StartAddr = (char*)allocatedAddress;
  InitializeBlocks();
  No_of_Threads = atoi(argv[1]);
  sw = atoi(argv[2]);
  int state1, state2;
  state1 = pthread_mutex_init(&mutx, NULL);
  state2 = sem_init(&bin_sem, 0 ,0);
  if(state1 || state2 !=0)
    puts("Error mutex & semaphore initialization!!!");
  for (i = 1; i <= No_of_Threads; i++)
    {
      pthread_t id;
      pthread_create(&id, NULL, MMS, NULL);
    }
  sleep(2);
  free(allocatedAddress);
  return 0;

}

void *MMS(void *arg)
{
  char *allocAddr,*pointtoAddr,*blockAddress;
  int p,memReq, i, j,blkAlloc=2;
  blockAddress = StartAddr;
  Threads[ThreadIndex].id = (int)pthread_self();
  //printf("I am thread %d going to sleep\n",Threads[ThreadIndex]);
  p = 1 +(rand() % 10);
    Threads[ThreadIndex].size = p;

  pthread_mutex_lock(&mutx);
  //sleep(1);
  //printf("I am thread %d waking up\n",Threads[ThreadIndex]);
  //locking the mutex for the thread
```

```
{
 switch(sw)
  {
  case 1:{
        allocAddr = FirstFit(p, ThreadIndex);
        printf("Starting address %d and memory requested is %d\n",allocAddr,p);
        if(allocAddr == 0)
        printf("Insufficient Memory Error!!!\n");
        break;
  }
  case 2:
        {
         allocAddr = BestFit(p, ThreadIndex);
         printf("Starting address %d and memory requested is %d\n",allocAddr,p);
      if(allocAddr == 0)
         printf("Insufficient Memory Error!!!\n");
         break;
        }
  case 3:{
         allocAddr = WorstFit(p, ThreadIndex);
         printf("Starting address %d and memory requested is %d\n",allocAddr,p);
      if(allocAddr == 0)
      printf("Insufficient Memory Error!!!\n");
         break;
  }
  default:{
        printf("You selected invalid input");
        exit(-1);
  }
  }
 if(allocAddr == 0)
 goto UNLOCK;
 pointtoAddr = allocAddr;
```

```
   printf("Block Allocated=%d\n", p);
   for (i = 0;i < p;i++)
    {
          *pointtoAddr = (ThreadIndex + 1);
          pointtoAddr++;
    }
    printf("Ending address %d\n",pointtoAddr);

   //Memory Info
   for(i=0; i < block_Count; i++)
          {
                 for(j=0; j < Blocks[i].blockSize; j++){
              if (*Blocks[i].blockStartAddress != 0){
                               Blocks[i].dataInBlock = 1;
                               break;
                        }
                 }
          }
   printf("\nALLOCATED MEMORY UNIT\n");
   Print_Memory();
   ThreadIndex++;
   UNLOCK: pthread_mutex_unlock(&mutx);
   //unlocking the mutex
  }




}

// First Fit Algorithm
char *FirstFit(int blockAllocationSize, int threadIndex)
{
  char *pointer;
  int i, found = 0;
  printf("\nFirst Fit\n");
  for (i = 0;i < block_Count;i++)
    {
      if ((Blocks[i].dataInBlock == 0) && (Blocks[i].blockSize >= blockAllocationSize))
              {
                found = 1;
                break;
              }
    }
  if (found == 0)
    {
      pointer = ReqMem(blockAllocationSize);
    }
  else
    pointer = Blocks[i].blockStartAddress;
  return pointer;
}

//Best Fit Algorithm
char *BestFit(int blockAllocationSize, int threadIndex)
{
  printf("entered best fit\n");
  printf("memory requested is %d \n",blockAllocationSize);
  char *pointer;
  int i = 0, j=0, found = 0;
  int t;
  int required;
  required = blockAllocationSize;

  printf("No of blocks is %d\n",block_Count);
  while(i < block_Count)
  {
```

```
  //printf("entered while loop \n");
  for(j=0; j < block_Count; j++)
        {
              //printf("entered for loop \n");
              if((required == Blocks[j].blockSize) && (Blocks[j].dataInBlock == 0))
              goto EXIT;
        }
        required++;
        i++;
 }
 EXIT: pointer = Blocks[j].blockStartAddress;
 return pointer;
}

//Worst Fit Algorithm
char *WorstFit(int blockAllocationSize, int threadIndex)
{
 char *pointer;
 int i = 0, j, found = 0,max;
 int t;

 for(i=0;i < block_Count;i++)
 {
        if(Blocks[i].dataInBlock == 0)
        {
              max = Blocks[i].blockSize;
              break;
        }
 }
 if((Blocks[0].dataInBlock == 0) && (blockAllocationSize <= Blocks[0].blockSize))
 {
 pointer = Blocks[0].blockStartAddress;
 found = 1;
 }
 for(i=1; i < block_Count; i++)
 {
        if((max <= Blocks[i].blockSize) && (Blocks[i].dataInBlock == 0))
                {
                        if(blockAllocationSize <= Blocks[i].blockSize)
                        {
                        max = Blocks[i].blockSize;
                        pointer = Blocks[i].blockStartAddress;
                        found = 1;
                        }
                }
 }

 if(found == 1)
 return pointer;
 else
 return 0;
}

//Initialize Blocks
void InitializeBlocks()
{
 char *temp,*var;
 int i,n,j,k,p;
 p=0;
 k=0;
 var = StartAddr;
 temp = StartAddr;
 for (i = 0;i <Max_Thread_Count ;i++)
  {
   //Blocks[i].blockSize = 0;
   Blocks[i].dataInBlock = 0;
```

```
    //Blocks[i].blockStartAddress = StartAddr;
  }
for (i = 0;i < MAX_MEM_SIZE;i++)
  {
    *temp = 0;
    temp++;
  }
EndAddr = (temp);
i =0;
while(p < MAX_MEM_SIZE)
{
        n = 1+rand()%8;
        k = k+n;
        if(k > MAX_MEM_SIZE)
        {
                n = MAX_MEM_SIZE-p;
        }
        p = p + n;

        if(i == 0)
        {
                Blocks[i].blockSize = n;
                printf("Block size for block %d is %d\n",i,Blocks[i].blockSize);
                Blocks[i].blockStartAddress = StartAddr;
                printf("Starting address is for block %d is %d\n", i,Blocks[i].blockStartAddress);
                for(j=0; j < n; j++)
                {
                        var++;
                }
        }
        else
        {
                Blocks[i].blockSize = n;
                printf("Block size for block %d is %d\n",i,Blocks[i].blockSize);
                Blocks[i].blockStartAddress = var;
                printf("Starting address for block %d is %d\n", i,Blocks[i].blockStartAddress);
                for(j=0;j < n; j++)
                {
                        var++;
                }
        }
        i++;
  }
  block_Count = i;
}


//Request Memory from MMU
char *ReqMem(int size)
{
  int i;
  int found = 0;
  for (i = 0;i < block_Count;i++)
   {
     if ((Blocks[i].dataInBlock == 0) && (Blocks[i].blockSize >= size)){
        found =1;
        break;
        }
   }
  if (found == 1)
  return Blocks[i].blockStartAddress;
  else if(found == 0)
        {
                return 0;
        }
}
```

```
// Display Memory Index
void Print_Memory()
{
  int i;
  char *index;
  index = StartAddr;
  printf("\n--------MEMORY BLOCK---------\n");
  //printf("Starting address: %d\n", index);
  for (i = 0;i < MAX_MEM_SIZE;i++)
    {
      printf("%d\t\t\t", *index);
      index++;
    }
   //printf("Ending  address: %d\n", index);
  printf("\nNumber of Blocks = %d\n\n", block_Count);
}
```