CS 204

Operating Systems Assignment 2

Mohammed Rabeeh

Roll No: 35

S4 CS

Fixed Partition

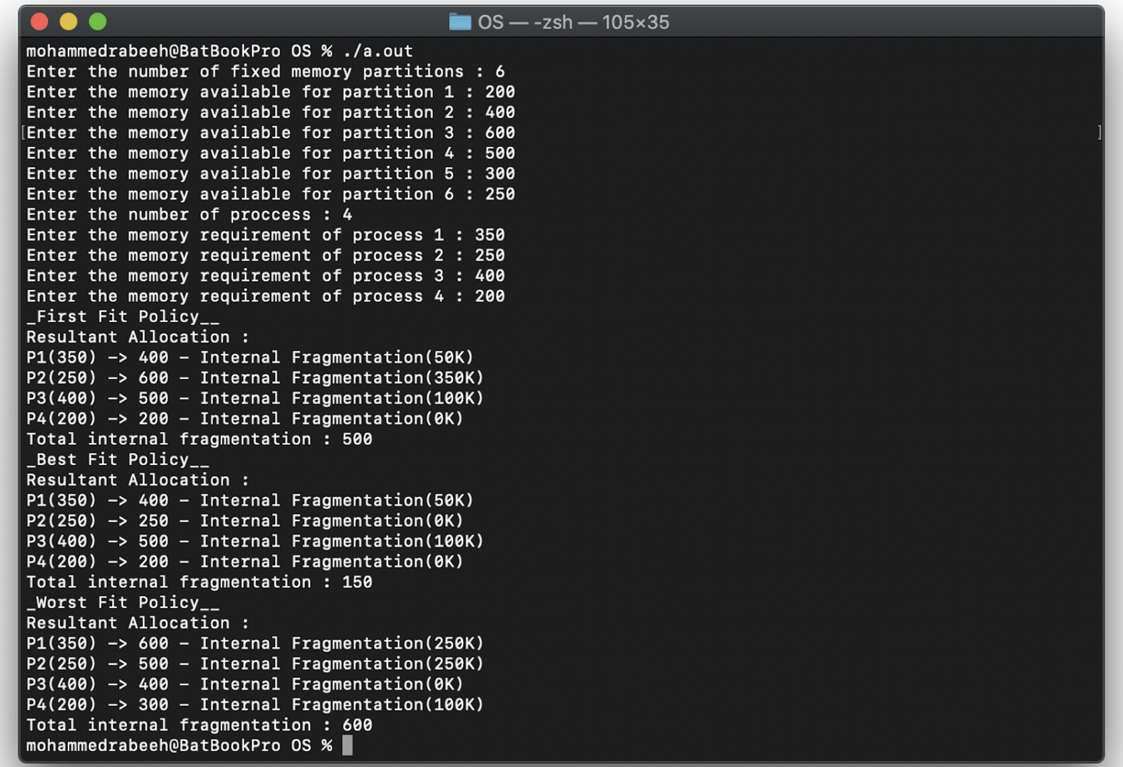
```
#include<stdio.h>
#include<stdlib.h>

void getAllocation(int partitionSize[], int partitionNo, int
processSize[], int processNo) {

   int internalFragmentation = 0;
   int externalFragmentation = 0;
   int remainingMemory = 0;
   int memoryAllocated, partitionAllocated[100];
   int externalFragmentationProcess[100];

   for(int i = 0; i < partitionNo; i++) {
       partitionAllocated[i] = 0; // Initially none of the
partitions are allocated
       remainingMemory += partitionSize[i];
   }

   for(int i = 0; i < processNo; i++) {
       externalFragmentationProcess[i] = 0; // Initially
```

```
assumes that external fragmentation will not occur
       memoryAllocated = 0;
       for(int j = 0; j < partitionNo; j++) {

           if(partitionAllocated[j]) {
               continue;
           }

           if(processSize[i] <= partitionSize[j]) {
               memoryAllocated = 1;
               partitionAllocated[j] = 1;
               remainingMemory -= partitionSize[j];

               int fragmentation = partitionSize[j] -
processSize[i];
               internalFragmentation += fragmentation;

               printf("P%d(%d) -> %d - Internal
Fragmentation(%dK)\n", i+1, processSize[i], partitionSize[j],
fragmentation);
               break;
           }
       }
       // Condition for external fragemntatio
       if(!memoryAllocated && remainingMemory >=
processSize[i]) {
           externalFragmentationProcess[i] = 1;
           externalFragmentation += processSize[i];
       }
```

```
   }

   printf("Total internal fragmentation : %d\n",
internalFragmentation);
   if(externalFragmentation) {
       printf("Following process caused external fragmentation
: \n");
       for(int i = 0; i < processNo; i++) {
           if(externalFragmentationProcess[i]) {
               printf("P%d\t", i+1);
           }
       }
       printf("\nTotal external fragmentation : %d\n",
externalFragmentation);
   }
}

void printFirstFitAllocation(int partitionSize[], int
partitionNo, int processSize[], int processNo) {

   printf("_First Fit Policy__\n");
   printf("Resultant Allocation : \n");

   getAllocation(partitionSize, partitionNo, processSize,
processNo);
}

int comparator1(const void *p, const void *q)
{
   // Get the values at given addresses
   int l = *(const int *)p;
   int r = *(const int *)q;

   return l-r;
}

void printBestFitAllocation(int partitionSize[], int
partitionNo, int processSize[], int processNo) {

   // Sort the partitions array in ascending order
   qsort((void *)partitionSize, partitionNo,
sizeof(partitionSize[0]), comparator1);

   printf("_Best Fit Policy__\n");
   printf("Resultant Allocation : \n");

   getAllocation(partitionSize, partitionNo, processSize,
processNo);
}

int comparator2(const void *p, const void *q)
{
   // Get the values at given addresses
   int l = *(const int *)p;
   int r = *(const int *)q;

   return r-l;
}
```

```c
void printWorstFitAllocation(int partitionSize[], int
partitionNo, int processSize[], int processNo) {

    // Sort the partitions array in descending order
    qsort((void *)partitionSize, partitionNo,
sizeof(partitionSize[0]), comparator2);

    printf("_Worst Fit Policy__\n");
    printf("Resultant Allocation : \n");

    getAllocation(partitionSize, partitionNo, processSize,
processNo);

}

void main() {
    int partitionNo, processNo;
    int partitionSize[100], processSize[100];

    printf("Enter the number of fixed memory partitions : ");
    scanf("%d", &partitionNo);

    for(int i = 0; i < partitionNo; i++) {
        printf("Enter the memory available for partition %d : ",
i+1);
        scanf("%d", &partitionSize[i]);
    }
```

```
   printf("Enter the number of proccess : ");
   scanf("%d", &processNo);

   for(int i = 0; i < processNo; i++) {
       printf("Enter the memory requirement of process %d : ",
i+1);
       scanf("%d", &processSize[i]);
   }

   printFirstFitAllocation(partitionSize, partitionNo,
processSize, processNo);
   printBestFitAllocation(partitionSize, partitionNo,
processSize, processNo);
   printWorstFitAllocation(partitionSize, partitionNo,
processSize, processNo);

}
```



```
mohammedrabeeh@BatBookPro OS % ./a.out
Enter the number of fixed memory partitions : 6
Enter the memory available for partition 1 : 200
Enter the memory available for partition 2 : 400
Enter the memory available for partition 3 : 600
Enter the memory available for partition 4 : 500
Enter the memory available for partition 5 : 300
Enter the memory available for partition 6 : 250
Enter the number of proccess : 4
Enter the memory requirement of process 1 : 350
Enter the memory requirement of process 2 : 250
Enter the memory requirement of process 3 : 400
Enter the memory requirement of process 4 : 200
_First Fit Policy__
Resultant Allocation :
P1(350) -> 400 - Internal Fragmentation(50K)
P2(250) -> 600 - Internal Fragmentation(350K)
P3(400) -> 500 - Internal Fragmentation(100K)
P4(200) -> 200 - Internal Fragmentation(0K)
Total internal fragmentation : 500
_Best Fit Policy__
Resultant Allocation :
P1(350) -> 400 - Internal Fragmentation(50K)
P2(250) -> 250 - Internal Fragmentation(0K)
P3(400) -> 500 - Internal Fragmentation(100K)
P4(200) -> 200 - Internal Fragmentation(0K)
Total internal fragmentation : 150
_Worst Fit Policy__
Resultant Allocation :
P1(350) -> 600 - Internal Fragmentation(250K)
P2(250) -> 500 - Internal Fragmentation(250K)
P3(400) -> 400 - Internal Fragmentation(0K)
P4(200) -> 300 - Internal Fragmentation(100K)
Total internal fragmentation : 600
mohammedrabeeh@BatBookPro OS %
```

Page Replacement

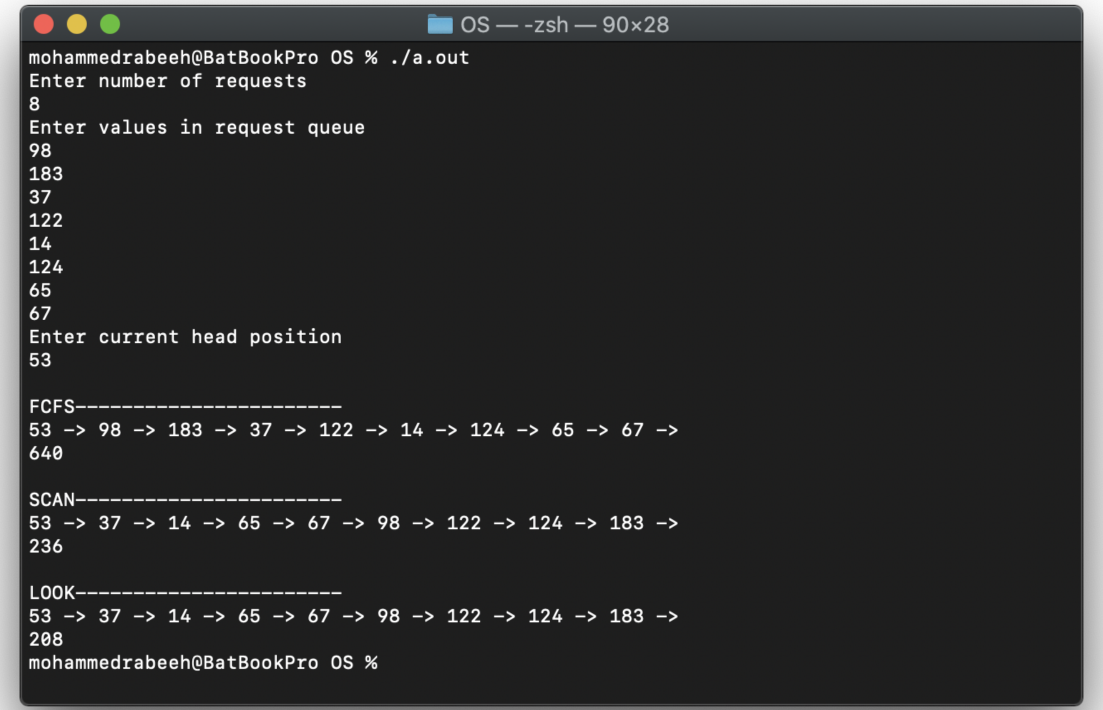
```c
#include <stdio.h>
#include <stdlib.h>

int no_of_frames;

int fifo_page_replace(int *frames, char *page_ref_string)
{
   int i = 0, j = 0, no_of_faults = 0, front = 0, hit = 0;
   for(i = 0; i < no_of_frames; i++)
```

```
        frames[i] = -1;

    for(i = 0; page_ref_string[i]; i++)
    {
        hit = 0;
        for(j = 0; j < no_of_frames; j++)
            if(frames[j] == page_ref_string[i])
                hit = 1;



        if(!hit)
        {
            ++no_of_faults;
            frames[front] = page_ref_string[i];
            front = (front + 1) % no_of_frames;
        }

        for(j = 0; j < no_of_frames; j++)
            if(frames[j] == -1)
                printf("F");
            else
                printf("%c", frames[j]);

        printf("\n");
    }

    printf("Number of page faults: %d\n", no_of_faults);
}
```

```
int lru_page_replace(int *frames, char *page_ref_string)
{
   int i = 0, j = 0, k =0, no_of_faults = 0, hit = 0, count =
0;
   int time[no_of_frames], min, minIndex;

   for(i = 0; i < no_of_frames; i++)
       frames[i] = -1;

   for(i = 0; page_ref_string[i]; i++)
   {
       hit = 0;

       for(j = 0; j < no_of_frames; j++)
       {
           if(frames[j] == page_ref_string[i])
           {
               hit = 1;
               time[j] = ++count;
           }
       }

       if(!hit)
       {
           ++no_of_faults;
           for(j = 0;j < no_of_frames; j++)
           {
               if(frames[j] == -1)
               {
```

```c
            frames[j] = page_ref_string[i];
            time[j] = ++count;
            break;
        }
    }
    if(j == no_of_frames)
    {
        min = time[0];
        minIndex = 0;

        for(k = 0; k < no_of_frames; k++)
        {
            if(min > time[k])
            {
                min = time[k];
                minIndex = k;
            }
        }

        frames[minIndex] = page_ref_string[i];
        time[minIndex] = ++count;
    }
}

for(j = 0; j < no_of_frames; j++)
    if(frames[j] == -1)
        printf("F");
    else
```

```c
               printf("%c", frames[j]);
       printf("\n");
   }

   printf("Number of page faults: %d\n", no_of_faults);
}

int main()
{
   char page_ref_string[30];
   int *frames;

   printf("Enter number of frames\n");
   scanf("%d", &no_of_frames);

   frames = malloc(no_of_frames * sizeof(int));


   printf("Enter page reference string\n");
   scanf("%s", page_ref_string);

   printf("\nFIFO\n---------------------------\n");
   fifo_page_replace(frames, page_ref_string);

   printf("\nLRU\n---------------------------\n");
   lru_page_replace(frames, page_ref_string);
}
```



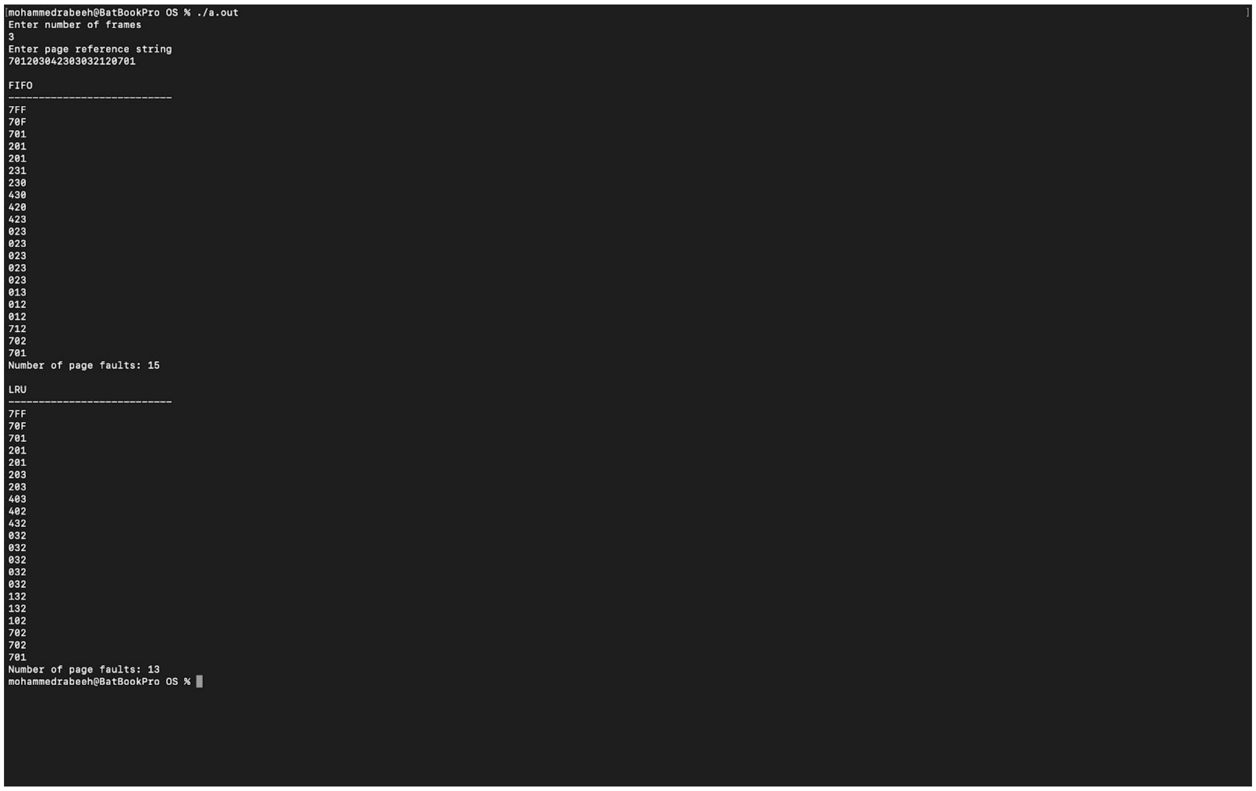
```
mohammedrabeeh@BatBookPro OS % ./a.out
Enter number of requests
8
Enter values in request queue
98
183
37
122
14
124
65
67
Enter current head position
53

FCFS-----------------------
53 -> 98 -> 183 -> 37 -> 122 -> 14 -> 124 -> 65 -> 67 ->
640

SCAN-----------------------
53 -> 37 -> 14 -> 65 -> 67 -> 98 -> 122 -> 124 -> 183 ->
236

LOOK-----------------------
53 -> 37 -> 14 -> 65 -> 67 -> 98 -> 122 -> 124 -> 183 ->
208
mohammedrabeeh@BatBookPro OS %
```

Disk Scheduling

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int no_of_requests;

int compar(const void *a, const void *b)
{
    return *(int *)a - *(int *)b;
}
```

```
int fcfs_disk_schedule(int *request_queue, int head)
{
   int i, total_head_movements = 0;
   printf("%d -> ", head);

   for(i = 0; i < no_of_requests; i++)
   {
       printf("%d -> ", request_queue[i]);
       total_head_movements += abs(head - request_queue[i]);
       head = request_queue[i];
   }

   printf("\n%d\n", total_head_movements);
}

int scan_disk_schedule(int *request_queue, int head)
{
   int total_head_movements = 0, i, head_location;

   for(i = 0; head > request_queue[i] && i < no_of_requests;
i++);
   head_location = (i == no_of_requests) ? i-1: i;

   printf("%d -> ", head);

   while(i > 0)
   {
       printf("%d -> ", request_queue[i-1]);
```

```c
        total_head_movements += abs(head - request_queue[i-1]);
        head = request_queue[i - 1];
        --i;
    }

    total_head_movements += head;
    head = 0;
    i = head_location;
    while(i < no_of_requests)
    {
        printf("%d -> ", request_queue[i]);
        total_head_movements += abs(head - request_queue[i]);
        head = request_queue[i];
        ++i;
    }

    printf("\n%d\n", total_head_movements);
}

int look_disk_schedule(int *request_queue, int head)
{
    int total_head_movements = 0, i, head_location;

    for(i = 0; head > request_queue[i] && i < no_of_requests;
i++);
    head_location = (i == no_of_requests) ? i-1: i;

    printf("%d -> ", head);
    while(i > 0)
    {
        printf("%d -> ", request_queue[i-1]);
        total_head_movements += abs(head - request_queue[i-1]);
        head = request_queue[i - 1];
        --i;
    }

    i = head_location;
    while(i < no_of_requests)
    {
        printf("%d -> ", request_queue[i]);
        total_head_movements += abs(head - request_queue[i]);
        head = request_queue[i];
        ++i;
    }

    printf("\n%d\n", total_head_movements);
}

int main()
{
    int head, i;
    int *request_queue;

    printf("Enter number of requests\n");
    scanf("%d", &no_of_requests);

    request_queue = (int *) malloc(no_of_requests*sizeof(int));
    printf("Enter values in request queue\n");
    for(i = 0; i < no_of_requests; i++)
        scanf("%d", &request_queue[i]);

    printf("Enter current head position\n");
    scanf("%d", &head);

    printf("\nFCFS-----------------------\n");
    fcfs_disk_schedule(request_queue, head);

    qsort(request_queue, no_of_requests, sizeof(int), compar);

    printf("\nSCAN-----------------------\n");
    scan_disk_schedule(request_queue, head);

    printf("\nLOOK-----------------------\n");
    look_disk_schedule(request_queue, head);
}
```



```
mohammedrabeeh@BatBookPro OS % ./a.out
Enter number of frames
3
Enter page reference string
70120304230303212070 1
```

```
mohammedrabeeh@BatBookPro OS % ./a.out
Enter number of frames
3
Enter page reference string
701203042303032120701

FIFO
----------------------------
7FF
70F
701
201
201
231
230
430
420
423
023
023
023
023
023
013
012
012
712
702
701
Number of page faults: 15

LRU
----------------------------
7FF
70F
701
201
201
203
203
403
402
432
032
032
032
032
032
132
132
102
702
702
701
Number of page faults: 13
mohammedrabeeh@BatBookPro OS %
```