UCS 1411 - Operating Systems Lab

Exercise 8 - Implementation of Memory Management Algorithms

Mahesh Bharadwaj K - 185001089

1 Develop a C program to implement the Memory Management Algorithms

Linked List Header File

```
typedef Partition *Data;
typedef struct Node
    Data d;
    struct Node *next;
} Node;
typedef Node *List;
List createEmptyList()
    Node *head = (Node *)malloc(sizeof(Node));
    head->d = NULL;
    head->next = NULL;
    return head;
}
void insert(List head, const Data d)
    Node *new = (Node *)malloc(sizeof(Node));
    new->d = d;
    Node *tmp = head;
    while (tmp->next != NULL)
        if (tmp->next->d->start > d->start)
            break;
        tmp = tmp->next;
    new->next = tmp->next;
    tmp->next = new;
}
Data delete (List prev)
    Data rVal=NULL;
    if (!prev)
        return rVal;
    if (!prev->next)
        return rVal;
```

```
Node *tmp = prev->next;
   rVal = tmp->d;
   prev->next = prev->next->next;
   free(tmp);
   return rVal;
}
void display(List head)
{
   Node *tmp = head->next;
   if (tmp == NULL)
       printf(" Empty!\n");
       return;
   }
   int count = 0;
   while(tmp != NULL){
       tmp = tmp -> next;
       count++;
   }
   printf("\n ");
    for(int i = 0; i < count; i++)</pre>
       printf("+----- ");
   printf("\n ");
   tmp = head -> next;
   while (tmp != NULL)
       printf("| %-4s
                         ", printState(*(tmp->d)));
       tmp = tmp->next;
   printf("\n ");
   for(int i = 0; i < count; i++)</pre>
       printf("+----- ");
   printf("\n ");
   tmp = head -> next;
   for(int i = 0; i < count; i++){</pre>
       printf("%-3d
                       %-3d ", tmp -> d -> start, tmp -> d -> end);
       tmp = tmp -> next;
   }
}
```

Main Program

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

typedef struct Partition
{
   unsigned int start;
   unsigned int end;
```

```
unsigned int size;
    int state;
} Partition;
char *const printState(const Partition P)
   static char str[5];
   if (P.state < -1)
        exit(1);
   else if (P.state == -1)
        strcpy(str, "Hole");
   else
        str[0] = 'P';
        str[1] = (P.state / 10) + 48;
        str[2] = (P.state \% 10) + 48;
        str[3] = ' ';
        str[4] = '\0';
   return str;
}
#define HOLE -1
#include "LinkedList.h"
typedef enum Mode
   FirstFit = 1,
   BestFit,
   WorstFit
} Mode;
void FFAlloc(List, List, List, const int, const unsigned int);
void BFAlloc(List, List, List, const int, const unsigned int);
void WFAlloc(List, List, List, const int, const unsigned int);
void Dealloc(List, List, List, const int);
void Coalesce(List, List);
int main()
   int n, pid, choice = -1;
   unsigned int size;
   Mode m;
   Partition *tmp;
   List memory = createEmptyList();
   List free = createEmptyList();
   List allocated = createEmptyList();
   printf(" Enter the number of partitions: ");
   scanf("%d", &n);
   for (int i = 0; i < n; i++)
        tmp = (Partition *)malloc(sizeof(Partition));
        printf(" Enter the start and end address: ");
```

```
scanf("%d %d", &(tmp->start), &(tmp->end));
    tmp->size = tmp->end - tmp->start;
    tmp->state = HOLE;
    insert(memory, tmp);
    insert(free, tmp);
}
while (1)
   printf("\t\tMEMORY ALLOCATION TECHNIQUES\n");
   printf(" 1 - First Fit\n");
   printf(" 2 - Best Fit\n");
   printf(" 3 - Worst Fit\n");
   printf(" 0 - Exit\n");
   printf(" ----\n");
   printf(" Enter your choice: ");
    scanf("%d", &m);
    if (m < 0 \mid | m > 3)
       printf("Invalid Mode!\n");
       continue;
    }
    if (m == 0)
       return 0;
    while (1)
    {
       printf("\n\n");
       printf("\t\t\t\tOPTIONS\n");
       printf(" 1 - Entry / Allocate\n");
       printf(" 2 - Exit / De-Allocate\n");
       printf(" 3 - Display\n");
       printf(" 4 - Coalescing of Holes\n");
       printf(" 5 - Back\n");
       printf(" ----\n");
       printf(" Enter your choice: ");
       scanf("%d", &choice);
       if (choice < 1 || choice > 5)
        {
            printf(" Invalid Input\n");
            continue;
        if (choice == 5)
           break;
       switch (choice)
       {
        case 1:
           printf("\n Enter the PID of process: ");
           scanf("%d", &pid);
            printf(" Enter the size required: ");
            scanf("%d", &size);
            switch (m)
```

```
{
                case FirstFit:
                    FFAlloc(memory, free, allocated, pid, size);
                    break;
                case BestFit:
                    BFAlloc(memory, free, allocated, pid, size);
                case WorstFit:
                    WFAlloc(memory, free, allocated, pid, size);
                default:
                    break;
                }
                break;
            case 2:
                printf(" Enter PID of process to exit: ");
                scanf("%d", &pid);
                Dealloc(memory, free, allocated, pid);
                break:
            case 3:
                printf(" ALLOCATED PARTITIONS:\n");
                display(allocated);
                printf("\n FREE PARTITIONS:\n");
                display(free);
                printf("\n ALL PARTITIONS:\n");
                display(memory);
                break;
            case 4:
                Coalesce(memory, free);
            default:
                break;
            }
        }
   }
}
void FFAlloc(List memory, List free, List alloc, const int pid, const unsigned int size)
{
   Partition *fragment;
   if (free->next == NULL)
        printf(" No Free Space Available!\n");
        return;
   }
   int flag = 0;
   unsigned int total_size;
   Partition *p;
   List tmp = free;
   while (tmp->next != NULL)
    {
        if (tmp->next->d->state != HOLE)
            tmp = tmp->next;
            continue;
        }
```

```
if (tmp->next->d->size >= size)
            flag = 1;
            if (tmp->next->d->size == size)
                p = delete (tmp);
                p->state = pid;
                insert(alloc, p);
                break;
            }
            else
            {
                p = delete (tmp);
                fragment = (Partition *)malloc(sizeof(Partition));
                fragment->end = p->end;
                fragment->start = p->start + size;
                fragment->state = HOLE;
                fragment->size = fragment->end - fragment->start;
                p->end = p->start + size;
                p->state = pid;
                p->size = size;
                insert(memory, fragment);
                insert(free, fragment);
                insert(alloc, p);
                break;
            }
        }
        tmp = tmp->next;
   }
    if (!flag)
       printf(" Unable to Allocate Required Memory!\n");
        printf(" Successfully Allocated!\n");
}
void BFAlloc(List memory, List free, List alloc, const int pid, const unsigned int size)
{
    if (free->next == NULL)
        printf(" No Free Space Available!\n");
   unsigned int left_over = 999;
   Node *ptr = NULL;
   Partition *p, *fragment;
   List tmp = free;
   while (tmp->next != NULL)
        if (tmp->next->d->state != HOLE)
        {
            tmp = tmp->next;
```

```
continue;
        if (tmp->next->d->size >= size)
            if (tmp->next->d->size - size < left_over)</pre>
                left_over = tmp->next->d->size - size;
                ptr = tmp;
        tmp = tmp->next;
   }
   if (!ptr)
        printf(" Unable to allocate required memory!\n");
        return;
   }
   p = delete (ptr);
   p->state = pid;
   p->size = size;
   if (left_over == 0)
        insert(alloc, p);
   else
        fragment = (Partition *)malloc(sizeof(Partition));
        fragment->start = p->start + size;
        fragment->end = p->end;
        fragment->state = HOLE;
        fragment->size = fragment->end - fragment->start;
        p->end = p->start + size;
        insert(alloc, p);
        insert(memory, fragment);
        insert(free, fragment);
   printf(" Successfully Allocated Memory!\n");
}
void WFAlloc(List memory, List free, List alloc, const int pid, const unsigned int size)
{
   if (free->next == NULL)
        printf(" No Free Space Available!\n");
        return;
   }
   unsigned int left_over = 0;
   Node *ptr = NULL;
   Partition *p, *fragment;
   List tmp = free;
   while (tmp->next != NULL)
        if (tmp->next->d->state != HOLE)
        {
```

```
tmp = tmp->next;
            continue;
        if (tmp->next->d->size >= size)
            if (tmp->next->d->size - size > left_over)
                left_over = tmp->next->d->size - size;
                ptr = tmp;
        tmp = tmp->next;
   }
   if (!ptr)
        printf(" Unable to allocate required memory!\n");
        return;
   p = delete (ptr);
   p->state = pid;
   p->size = size;
    if (left_over == 0)
        insert(alloc, p);
    else
    {
        fragment = (Partition *)malloc(sizeof(Partition));
        fragment->start = p->start + size;
        fragment->end = p->end;
        fragment->state = HOLE;
        fragment->size = fragment->end - fragment->start;
        p->end = p->start + size;
        insert(alloc, p);
        insert(memory, fragment);
        insert(free, fragment);
   }
   printf(" Successfully Allocated Memory!\n");
void Dealloc(List memory, List free, List alloc, const int pid)
{
    if (alloc->next == NULL)
        printf(" No Process Allocated!\n");
        return;
   }
   Partition *p;
   Node *tmp = alloc;
   int flag = 0;
   while (tmp->next != NULL)
    {
        if (tmp->next->d->state == pid)
            flag = 1;
            break;
        }
```

}

```
tmp = tmp->next;
   if (flag == 0)
        printf(" No such Process Found!\n");
        return;
   }
   p = delete (tmp);
   p->state = HOLE;
   insert(free, p);
   printf(" Successfully De-Allocated Memory\n");
}
void Coalesce(List memory, List free)
   if (!free->next)
       return;
   if (!free->next->next)
       return;
   Node *1 = NULL,
         *r = NULL;
   Partition *left = NULL,
              *right = NULL,
              *p = NULL;
   Node *tmp = free, *tmp2 = memory;
   while (tmp->next != NULL && tmp->next->next != NULL)
        if (tmp->next->d->end == tmp->next->next->d->start)
        {
            1 = tmp;
            left = tmp->next->d;
            r = tmp->next;
            right = tmp->next->next->d;
            p = (Partition *)malloc(sizeof(Partition));
            p->start = left->start;
            p->end = right->end;
            p->size = p->end - p->start;
            p->state = HOLE;
            delete (r);
            delete (1);
            insert(free, p);
            while (tmp2->next != NULL && tmp2->next->next != NULL)
                if (tmp2->next->d == left)
                {
                    1 = tmp2;
                    r = tmp2->next;
                    delete (r);
                    delete (1);
                    insert(memory, p);
                    break;
                tmp2 = tmp2->next;
```

Output

```
Enter the number of partitions: 5
Enter the start and end address: 100 150
Enter the start and end address: 160 170
Enter the start and end address: 200 250
Enter the start and end address: 275 300
Enter the start and end address: 350 450
MEMORY ALLOCATION TECHNIQUES
1 - First Fit
2 - Best Fit
3 - Worst Fit
0 - Exit
Enter your choice: 1
OPTIONS
1 - Entry / Allocate
2 - Exit / De-Allocate
3 - Display
4 - Coalescing of Holes
5 - Back
 _____
Enter your choice: 1
Enter the PID of process: 1
Enter the size required: 10
Successfully Allocated!
OPTIONS
1 - Entry / Allocate
2 - Exit / De-Allocate
3 - Display
4 - Coalescing of Holes
5 - Back
Enter your choice: 3
 ALLOCATED PARTITIONS:
 | P01 |
    110
 100
FREE PARTITIONS:
              +----+
                            +----+
 | Hole | Hole | Hole | Hole | Hole
                            +----+
```

100 110 110 150 160 170 200 250 275 300 350 450

OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 1

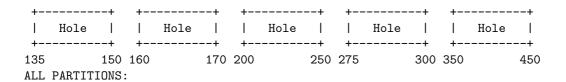
Enter the PID of process: 2 Enter the size required: 25 Successfully Allocated!

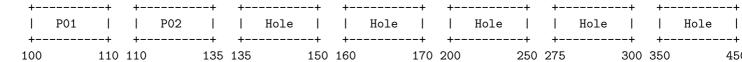
OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 3
ALLOCATED PARTITIONS:

+----+ +----+ | P01 | P02 | +----+ +----+ 100 110 110 135 FREE PARTITIONS:





OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes

5 - Back

Enter your choice: 2

Enter PID of process to exit: 2

Successfully De-Allocated Memory

OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 4

OPTIONS

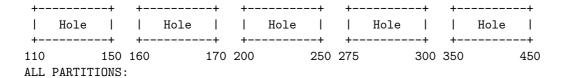
- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

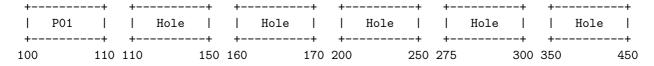
Enter your choice: 3

ALLOCATED PARTITIONS:

P01 | +-----+ 100 110

FREE PARTITIONS:





OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 2

Enter PID of process to exit: 1 Successfully De-Allocated Memory

OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 4

OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 3 ALLOCATED PARTITIONS:

Empty!

FREE PARTITIONS:

| + | | + | + | | + | + | | + | + | | + | + | | + |
|-----|--------|-------|-----|------|-----|-----|------|-----|-----|------|-----|-----|------|-----|
| 1 | Hole | 1 | - | Hole | 1 | 1 | Hole | 1 | 1 | Hole | 1 | 1 | Hole | 1 |
| + | | + | + | | + | + | | + | + | | + | + | | + |
| 100 | | 150 | 160 | | 170 | 200 | | 250 | 275 | | 300 | 350 | | 450 |
| ALL | PARTIT | IONS: | ; | | | | | | | | | | | |

| ++ +- | | | + | ++ | | | ++ | | | | + | ++ | | |
|-------|------|-----|-----|------|-----|-----|------|-----|-----|------|-----|-----|------|-----|
| - 1 | Hole | - 1 | - | Hole | - 1 | - [| Hole | - 1 | | Hole | - 1 | - 1 | Hole | - 1 |
| + | | + | + | | + | + | | + | + | | + | + | | + |
| 100 | | 150 | 160 | | 170 | 200 | | 250 | 275 | | 300 | 350 | | 450 |

OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 5

MEMORY ALLOCATION TECHNIQUES

- 1 First Fit
- 2 Best Fit
- 3 Worst Fit
- O Exit

Enter your choice: 2

OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 1

Enter the PID of process: 3 Enter the size required: 10 Successfully Allocated Memory!

OPTIONS

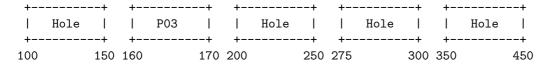
- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 3
ALLOCATED PARTITIONS:

+----+ | P03 | +----+ 160 170 FREE PARTITIONS:

| Hole |

ALL PARTITIONS:



OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

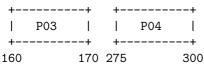
Enter your choice: 1

Enter the PID of process: 4
Enter the size required: 25
Successfully Allocated Memory!

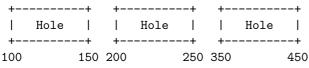
OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 3
ALLOCATED PARTITIONS:



FREE PARTITIONS:



ALL PARTITIONS:

| ++ | | | ++ | | | + | | + | + | | + | + | | | |
|-----|------|-----|-----|-----|-----|-----|------|-----|-----|-----|-----|-----|------|-----|--|
| - 1 | Hole | - 1 | - [| P03 | - 1 | - [| Hole | - | - [| P04 | | - 1 | Hole | - 1 | |
| + | | + | + | | + | + | | + | + | | + | + | | + | |
| 100 | | 150 | 160 | | 170 | 200 | | 250 | 275 | | 300 | 350 | | 450 | |

OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 2

Enter PID of process to exit: 3
Successfully De-Allocated Memory

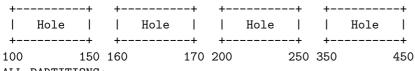
OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

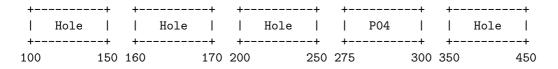
Enter your choice: 3 ALLOCATED PARTITIONS:

| + | | + |
|-----|-----|-----|
| | P04 | - 1 |
| + | | + |
| 275 | | 300 |

FREE PARTITIONS:



ALL PARTITIONS:



OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 2

Enter PID of process to exit: 4 Successfully De-Allocated Memory

OPTIONS

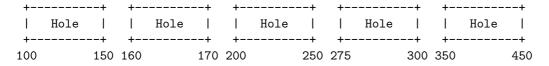
- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 3 ALLOCATED PARTITIONS:

Empty!

FREE PARTITIONS:

| + | | + | + | | + | + | | + | + | | + | + | | + |
|-----------------|------|-----|-----|------|-----|-----|------|-----|-----|------|-----|-----|------|-----|
| 1 | Hole | 1 | 1 | Hole | 1 | 1 | Hole | 1 | 1 | Hole | 1 | 1 | Hole | 1 |
| + | | + | + | | + | + | | + | + | | + | + | | + |
| 100 | | 150 | 160 | | 170 | 200 | | 250 | 275 | | 300 | 350 | | 450 |
| ALL PARTITIONS: | | | | | | | | | | | | | | |



OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 5

MEMORY ALLOCATION TECHNIQUES

- 1 First Fit
- 2 Best Fit
- 3 Worst Fit
- O Exit

Enter your choice: 3

OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 1

Enter the PID of process: 5 Enter the size required: 10 Successfully Allocated Memory!

OPTIONS

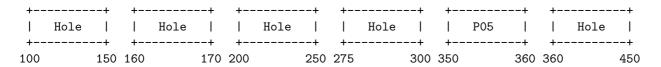
- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 3
ALLOCATED PARTITIONS:

+----+ | P05 |

350 360 FREE PARTITIONS:

+-----+ +-----+ +-----+ +-----+ +-----+ | Hole | | Hole



OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

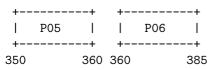
Enter your choice: 1

Enter the PID of process: 6
Enter the size required: 25
Successfully Allocated Memory!

OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 3 ALLOCATED PARTITIONS:



FREE PARTITIONS:

| + | | + | + | | + | + | | + | + | | + | + | | + |
|-----------------|------|-----|-----|------|-----|-----|------|-----|-----|------|-----|-----|------|-----|
| | Hole | 1 | - | Hole | | 1 | Hole | - | 1 | Hole | - | 1 | Hole | 1 |
| + | | + | + | | + | + | | + | + | | + | + | | + |
| 100 | | 150 | 160 | | 170 | 200 | | 250 | 275 | | 300 | 385 | | 450 |
| ALL PARTITIONS: | | | | | | | | | | | | | | |

| + | | + | + | | + | + | | + | + | | + | + | | + | + | | + | + | | + |
|-----|------|-----|-----|------|-----|-----|------|-----|-----|------|-----|-----|-----|-----|-----|-----|-----|-----|------|-----|
| I | Hole | - 1 | - 1 | P05 | - 1 | - | P06 | | - 1 | Hole | |
| + | | + | + | | + | + | | + | + | | + | + | | + | + | | + | + | | + |
| 100 | | 150 | 160 | | 170 | 200 | | 250 | 275 | | 300 | 350 | | 360 | 360 | | 385 | 385 | | 450 |

OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 2

Enter PID of process to exit: 5
Successfully De-Allocated Memory

OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 4

OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 2

Enter PID of process to exit: 6
Successfully De-Allocated Memory

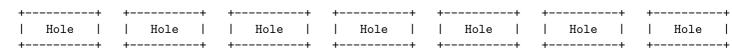
OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 3 ALLOCATED PARTITIONS:

Empty!

FREE PARTITIONS:



- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 4

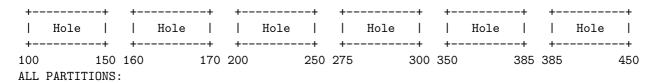
OPTIONS

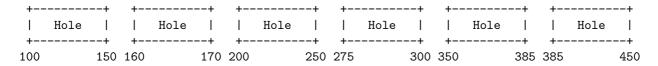
- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 3
ALLOCATED PARTITIONS:

Empty!

FREE PARTITIONS:





OPTIONS

- 1 Entry / Allocate
- 2 Exit / De-Allocate
- 3 Display
- 4 Coalescing of Holes
- 5 Back

Enter your choice: 5

MEMORY ALLOCATION TECHNIQUES

- 1 First Fit
- 2 Best Fit
- 3 Worst Fit
- 0 Exit

Enter your choice: 0