Department of Computer Science and Engineering

S.G.Shivanirudh , 185001146, Semester IV

25 March 2020

UCS1411 - Operating Systems Laboratory

Lab Exercise 8: Implementation of Memory Management Algorithms

Objective:

Develop a C program to implement the Memory Management Algorithms.

Code:

Q.To write a C program to implement the Memory Management Algorithms.

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

#define hole "H"

struct Box{
    int start_byte;
    int end_byte;
    int size;
    char* status;
    struct Box* next;
};

typedef struct Box Node;

void initialise(Node* n){

    n->start_byte=-1;
    n->end_byte=-1;
    n->size=0;
    n->status=(char*)malloc(10);
    strcpy(n->status,hole);
    n->next=NULL;
}

//Accept details of partitions
void readNode(Node *n){
    printf("Enter the start and end address of partition: ");
    scanf("%d",&n->start_byte);scanf("%d",&n->end_byte);

    n->size=n->end_byte-n->start_byte;
}

//Copy one partition to another
void copyNode(Node *OG,Node *copy){
    copy->start_byte=OG->start_byte;
    copy->end_byte=OG->end_byte;
    copy->size=OG->size;
    strcpy(copy->status,OG->status);
    copy->next=NULL;
}

//Count number of partitions
int partitionCount(Node* LL){
    Node *tmp=LL;
    int ctr=0;
    tmp=tmp->next;
    while(tmp){
        ctr++;
        tmp=tmp->next;
    }
    return ctr;
}

//Insert into Linked List
void insert(Node *LL, Node *n){

    Node *stmp=LL;
    while(stmp->next && n->start_byte >= stmp->next->start_byte){
        stmp=stmp->next;
    }
    n->next=stmp->next;
    stmp->next=n;
}

//Delete Node from Linked List
void delete(Node *LL,char *status){
    Node *tmp=LL;
    Node *prev=LL;
    tmp=tmp->next;
    while(strcmp(status,tmp->status)!=0){
        prev=tmp;
        tmp=tmp->next;
    }
    prev->next=tmp->next;
    free(tmp);
}

//Display memory
void displayMemory(Node *LL){

    int no_of_partitions=partitionCount(LL);

    Node *tmp=LL;
    tmp=tmp->next;

    if(tmp==NULL){
        printf("\nNULL\n");
    }
    else{
        //Display top line
        for(int i=0;i<no_of_partitions;i++){
            printf(" ______________  ");
        }

        //Display order of processes
        printf("\n");
        while(tmp){
            printf("|_____%4s_____| ",tmp->status);
            tmp=tmp->next;
        }
        printf("\n");

        //Display time line
        tmp=LL;
        tmp=tmp->next;
        while(tmp){
            printf("%-8d%8d ",tmp->start_byte,tmp->end_byte);
            tmp=tmp->next;
        }
        printf("\n\n");
    }
}
```

```c

#include "LinkedList.h"

//Select first hole that is able to satisfy the process
void FirstFitInsert(Node *AllocLL, Node *FreePoolLL,Node *newNode){

    Node *tmp=FreePoolLL;
    Node *prev=tmp;
    tmp=tmp->next;

    //To determine whether a process can be allotted memory or not
    int entry=0;
    while(tmp){

        if(tmp->size < newNode->size){
            prev=tmp;
            tmp=tmp->next;
        }
        else{
            newNode->start_byte=tmp->start_byte;
            newNode->end_byte=newNode->start_byte + newNode->size;

            //Check if size required is exactly the same size as a partition
            if(newNode->size==tmp->size){
                prev->next=tmp->next;
                free(tmp);
            }
            else{
                tmp->start_byte = newNode->end_byte;
                tmp->size -= newNode->size;
            }

            insert(AllocLL,newNode);
            entry=1;
            break;
        }
    }
    if(entry==0){
        printf("\n Process %s cannot be allocated memory. \n", newNode->status);
    }
}

//Select the smallest hole that can satisfy the process
void BestFitInsert(Node *AllocLL,Node *FreePoolLL,Node *newNode){

    Node *tmp=FreePoolLL;
    Node *prev=tmp;

    //Consider the smallest hole out of all possible ones
    Node *maybe=(Node*)malloc(sizeof(Node));
    initialise(maybe);

    tmp=tmp->next;

    //To determine whether a process can be allotted memory or not
    int entry=0;

    //Determine possible value of maybe, and hence check if process can be allotted memory
    while(tmp){
        if(tmp->size < newNode->size){
            tmp=tmp->next;
        }
        else{
            maybe=tmp;
            entry=1;
            break;
        }
    }

    if(entry){
        tmp=FreePoolLL;
```

```c
        tmp=tmp->next;

        //Find smallest hole
        while(tmp){
            if(tmp->size < newNode->size){
                prev=tmp;
            }
            else{
                if(tmp->size < maybe->size)
                    maybe=tmp;
            }
            tmp=tmp->next;
        }

        newNode->start_byte=maybe->start_byte;
        newNode->end_byte=newNode->start_byte + newNode->size;

        //Check if size required is exactly the same as that of a partition
        if(newNode->size==maybe->size){
            prev->next=maybe->next;
            free(maybe);
        }
        else{
            maybe->start_byte = newNode->end_byte;
            maybe->size -= newNode->size;
        }

        insert(AllocLL,newNode);
    }
    else{
        printf("\n Process %s cannot be allocated memory. \n", newNode->status);
    }
}

//Select largest hole that can satisfy the process
void WorstFitInsert(Node *AllocLL,Node *FreePoolLL,Node *newNode){

    Node *tmp=FreePoolLL;
    Node *prev=tmp;

    //Consider the largest hole out of all possible ones
    Node *maybe=(Node*)malloc(sizeof(Node));
    initialise(maybe);

    tmp=tmp->next;

    //To determine whether a process can be allotted memory or not
    int entry=0;

    //Determine possible value of maybe, and hence check if process can be allotted memory
    while(tmp){
        if(tmp->size < newNode->size){
            tmp=tmp->next;
        }
        else{
            maybe=tmp;
            entry=1;
            break;
        }
    }

    if(entry){

        tmp=FreePoolLL;
        tmp=tmp->next;

        //Find largest hole
        while(tmp){
            if(tmp->size < newNode->size){
                prev=tmp;
            }
            else{
                if(tmp->size > maybe->size)
                    maybe=tmp;
            }
            tmp=tmp->next;
        }

        newNode->start_byte=maybe->start_byte;
        newNode->end_byte=newNode->start_byte + newNode->size;

        //Check if size required is exactly the same as that of a partition
        if(newNode->size==maybe->size){
            prev->next=maybe->next;
            free(maybe);
        }
        else{
            maybe->start_byte = newNode->end_byte;
            maybe->size -= newNode->size;
        }

        insert(AllocLL,newNode);
    }
    else{
        printf("\n Process %s cannot be allocated memory. \n", newNode->status);
    }
}

void DeallocMem(Node *AllocLL,Node *FreePoolLL,char *status){

    Node *tmp=AllocLL;
    Node *prev=tmp;
    tmp=tmp->next;

    while(tmp&&strcmp(status,tmp->status)!=0){

        prev=tmp;
        tmp=tmp->next;
    }

    //Copy contents of Node to be deleted to another node
    prev->next=tmp->next;
    Node *ftmp=(Node*)malloc(sizeof(Node));
    initialise(ftmp);

    copyNode(tmp,ftmp);

    free(tmp);
    strcpy(ftmp->status,hole);
    insert(FreePoolLL,ftmp);
}


void createPhysicalMem(Node *AllocLL,Node *FreePoolLL){

    //Physical Memory
    Node *PM=(Node*)malloc(sizeof(Node));
    initialise(PM);

    Node *fp=FreePoolLL;Node *all=AllocLL;
    fp=fp->next;all=all->next;

    //Perform merge sort
    while(fp && all){
        Node *ftmp=(Node*)malloc(sizeof(Node));
        initialise(ftmp);
        copyNode(fp,ftmp);
        Node *atmp=(Node*)malloc(sizeof(Node));
        initialise(atmp);
        copyNode(all,atmp);

        if(fp->start_byte <= all->start_byte){

            insert(PM,ftmp);
            fp=fp->next;
        }
        else{

            insert(PM,atmp);
            all=all->next;
        }
    }

    while(fp){
        Node *ftmp=(Node*)malloc(sizeof(Node));
        initialise(ftmp);
        copyNode(fp,ftmp);

        insert(PM,ftmp);
        fp=fp->next;
    }
    while(all){
        Node *atmp=(Node*)malloc(sizeof(Node));
        initialise(atmp);
        copyNode(all,atmp);

        insert(PM,atmp);
        all=all->next;
    }
    displayMemory(PM);

}

void holeJoin(Node *FreePoolLL){

    Node *tmp=FreePoolLL;
    Node *prev=tmp;
    tmp=tmp->next;

    while(tmp){
        if(tmp->start_byte == prev->end_byte){
            prev->end_byte = tmp->end_byte;
            prev->size += tmp->size;
            prev->next=tmp->next;
            free(tmp);
            tmp=prev;
        }
        prev=tmp;
        tmp=tmp->next;

    }
}

void main(){

    int no_of_partitions;

    Node *AllocLL=(Node*)malloc(sizeof(Node));
    initialise(AllocLL);

    Node *FreePoolLL=(Node*)malloc(sizeof(Node));
    initialise(FreePoolLL);

    printf("\n Enter Memory representation:\n");
    printf("\n Enter number of partitons: ");scanf("%d",&no_of_partitions);

    for(int i=0;i<no_of_partitions;i++){
        Node* newNode=(Node*)malloc(sizeof(Node));
        initialise(newNode);
        readNode(newNode);
        insert(FreePoolLL,newNode);
    }
    int FitOpt;
    int operation;
    do{

        printf("\n Choose the memory allocation algorithm: \n");
        printf(" 1.First Fit \n 2.Worst Fit \n 3.Best Fit \n 0. Exit \n");
        printf(" Your choice: ");scanf("%d",&FitOpt);
        if(FitOpt){
            do{

                printf("\n Choose operation: \n 1.Process Entry \n 2.Process Exit \n 3.Display \n");
                printf(" 4.Coalescing holes \n 0.Back \n Your choice: ");
                scanf("%d",&operation);

                if(operation==1){
                    Node *process=(Node*)malloc(sizeof(Node));
                    initialise(process);
                    printf("\n Enter process ID: ");scanf(" %s",process->status);
                    printf("\n Enter size required: ");scanf("%d",&process->size);
                    if(FitOpt==1){
                        FirstFitInsert(AllocLL,FreePoolLL,process);
                    }
                    else if(FitOpt==2){
                        WorstFitInsert(AllocLL,FreePoolLL,process);
                    }
                    else if(FitOpt==3){
                        BestFitInsert(AllocLL,FreePoolLL,process);
                    }
                    else if(FitOpt!=0){
                        printf("\n Invalid algorithm. \n");
                    }
                    else;
                }
                else if(operation==2){
                    char *status=(char*)malloc(100);
                    printf("\n Enter process ID: ");scanf(" %s",status);
                    DeallocMem(AllocLL,FreePoolLL,status);
                }
                else if(operation==3){
                    printf("\n Allocated Memory: \n");
                    displayMemory(AllocLL);
                    printf("\n Free Pool: \n");
                    displayMemory(FreePoolLL);
                    printf("\n Physical Memory: \n");
                    createPhysicalMem(AllocLL,FreePoolLL);
                }
                else if(operation==4){
                    holeJoin(FreePoolLL);
                    printf("\n Free Pool: \n");
                    displayMemory(FreePoolLL);
                }
                else if(operation!=0){
                    printf("\n Invalid operation. \n");
                }
                else;

            }while(operation);
        }
    }while(FitOpt);
}
```

Output:

```

 Enter Memory representation:

 Enter number of partitons: 5
Enter the start and end address of partition: 0 100
Enter the start and end address of partition: 100 600
Enter the start and end address of partition: 600 800
Enter the start and end address of partition: 800 1100
Enter the start and end address of partition: 1100 1700

 Choose the memory allocation algorithm:
 1.First Fit
 2.Worst Fit
 3.Best Fit
 0. Exit
 Your choice: 1

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 1

 Enter process ID: P1

 Enter size required: 212

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 1

 Enter process ID: P2

 Enter size required: 417

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 1

 Enter process ID: p3

 Enter size required: 112

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 1

 Enter process ID: P4

 Enter size required: 426

 Process P4 cannot be allocated memory.

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 3

 Allocated Memory:
 ______________   ______________   ______________
|_____  P1_____| |_____  p3_____| |_____  P2_____|
100          312 312          424 1100        1517


 Free Pool:
 ______________   ______________   ______________   ______________
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
0            100 424          600 600          800 800         1100
 ______________
|_____   H_____|
1517        1700
 Physical Memory:
 ______________   ______________   ______________   ______________
|_____   H_____| |_____  P1_____| |_____  p3_____| |_____   H_____|
0            100 100          312 312          424 424          600
 ______________   ______________   ______________   ______________
|_____   H_____| |_____   H_____| |_____  P2_____| |_____   H_____|
600          800 800         1100 1100        1517 1517        1700
 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

 Enter process ID: P1

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

 Enter process ID: P2

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

 Enter process ID: p3

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 3

 Allocated Memory:

NULL

 Free Pool:
 ______________   ______________   ______________   ______________
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
0            100 100          312 312          424 424          600
 ______________   ______________   ______________   ______________
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
600          800 800         1100 1100        1517 1517        1700
 Physical Memory:
 ______________   ______________   ______________   ______________
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
0            100 100          312 312          424 424          600
 ______________   ______________   ______________   ______________
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
600          800 800         1100 1100        1517 1517        1700
 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 0

 Choose the memory allocation algorithm:
 1.First Fit
 2.Worst Fit
 3.Best Fit
 0. Exit
 Your choice: 2

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
```

```
4.Coalescing holes
0.Back
Your choice: 1

Enter process ID: P1

Enter size required: 212


Choose operation:
1.Process Entry
2.Process Exit
3.Display
4.Coalescing holes
0.Back
Your choice: 1

Enter process ID: P2

Enter size required: 417

Choose operation:
1.Process Entry
2.Process Exit
3.Display
4.Coalescing holes
0.Back
Your choice: 1

Enter process ID: P3

Enter size required: 112

Choose operation:
1.Process Entry
2.Process Exit
3.Display
4.Coalescing holes
0.Back
Your choice: 1

Enter process ID: P4

Enter size required: 426

Process P4 cannot be allocated memory.

Choose operation:
1.Process Entry
2.Process Exit
3.Display
4.Coalescing holes
0.Back
Your choice: 3

Allocated Memory:
--------------   --------------   --------------
```

```
|_____  P2_____| |_____  P1_____| |_____  P3_____|
100          517 1100        1312 1312        1424


 Free Pool:
 --------------   --------------   --------------   --------------
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
0            100 517          600 600          800 800         1100
 --------------
|_____   H_____|
1424        1700

 Physical Memory:
 --------------   --------------   --------------   --------------
|_____   H_____| |_____  P2_____| |_____   H_____| |_____   H_____|
0            100 100          517 517          600 600          800
 --------------   --------------   --------------   --------------
|_____   H_____| |_____  P1_____| |_____  P3_____| |_____   H_____|
800         1100 1100        1312 1312        1424 1424        1700


 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

 Enter process ID: P1

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

 Enter process ID: P2

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

 Enter process ID: P3

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 3

 Allocated Memory:

NULL

 Free Pool:
 ______________   ______________   ______________   ______________
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
0            100 100          517 517          600 600          800
 ______________   ______________   ______________   ______________
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
800         1100 1100        1312 1312        1424 1424        1700

 Physical Memory:
 ______________   ______________   ______________   ______________
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
0            100 100          517 517          600 600          800
 ______________   ______________   ______________   ______________
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
800         1100 1100        1312 1312        1424 1424        1700

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 0


 Choose the memory allocation algorithm:
 1.First Fit
 2.Worst Fit
 3.Best Fit
 0. Exit
 Your choice: 3

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 1

 Enter process ID: P1

 Enter size required: 212

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 1

 Enter process ID: P2

 Enter size required: 417

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 1

 Enter process ID: P3

 Enter size required: 112

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 1

 Enter process ID: P4

 Enter size required: 426

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 3

 Allocated Memory:
 --------------   --------------   --------------   --------------
|_____  P2_____| |_____  P3_____| |_____  P1_____| |_____  P4_____|
100          517 600          712 800         1012 1100        1526


 Free Pool:
 --------------   --------------   --------------   --------------
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
0            100 517          600 712          800 1012        1100
 --------------
|_____   H_____|
1526        1700

Physical Memory:
 --------------   --------------   --------------   --------------
|_____   H_____| |_____  P2_____| |_____   H_____| |_____  P3_____|
0            100 100          517 517          600 600          712
 --------------   --------------   --------------   --------------
      --------------
```

```
|_____   H_____| |_____  P1_____| |_____   H_____| |_____  P4_____|
     |_____   H_____|
712          800 800          1012 1012         1100 1100         1526
     1526        1700


 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

 Enter process ID: P1

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

 Enter process ID: P2

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

 Enter process ID: P3

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 2

 Enter process ID: P4

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 3

 Allocated Memory:

NULL
```

```

 Free Pool:
 ______________   ______________   ______________   ______________
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
0            100 100          517 517          600 600          712
 ______________   ______________   ______________   ______________
      ______________
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
     |_____   H_____|
712          800 800         1012 1012        1100 1100        1526
     1526        1700


 Physical Memory:
 ______________   ______________   ______________   ______________
|_____   H_____| |_____   H_____| |_____   H_____| |_____   H_____|
0            100 100          517 517          600 600          712
 ______________   ______________   ______________   ______________
      ______________
|_____   H_____| |_____  P1_____| |_____   H_____| |_____  P4_____|
     |_____   H_____|
712          800 800         1012 1012        1100 1100        1526
     1526        1700

 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 4

 Free Pool:
 ______________
|_____   H_____|
0           1700


 Choose operation:
 1.Process Entry
 2.Process Exit
 3.Display
 4.Coalescing holes
 0.Back
 Your choice: 0

 Choose the memory allocation algorithm:
 1.First Fit
 2.Worst Fit
 3.Best Fit
 0. Exit
 Your choice: 0
```