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

typedef struct

{

}ele;

int start; int end; int size;

char status[10];

struct node

{

ele e;

struct node\* next;

};

void insert(struct node \*a,ele n)

{

struct node \*temp;

temp=(struct node\*)malloc(sizeof(struct node)); temp->e=n;

temp->next=NULL; while(a->next!=NULL)

{

a=a->next;

}

a->next=temp;

}

void output(struct node \*a)

{

int i,count=0; struct node \*temp; temp=a->next; while(temp!=NULL)

{

count++; temp=temp->next;

}

int length = (24\*count)+count; printf("\n\t\t"); for(i=0;i<length+1;i++)

{

printf("-");

}

printf("\n\t\t"); int place,k=0; temp=a->next;

for(i=0;i<length;i++)

{

if(i%24==0)

{

}

else

{

place=0; printf("|");

place++;

if(place==12 && k<count)

{

printf("%s ",temp->e.status); temp=temp->next;

k++;

}

else if(place>12 || place<11) printf(" ");

}

}

printf("\n\t\t"); for(i=0;i<length+1;i++)

{

printf("-");

}

int j; printf("\n\t\t"); temp=a->next; for(i=0;i<count;i++)

{

for(j=0;j<23;j++)

if(j==0)

{

}

else

printf("%d",temp->e.start); temp=temp->next;

printf(" ");

}

temp=a->next; for(i=0;i<count;i++)

{

if(i==count-1)

printf("%d",temp->e.end); temp=temp->next;

}

}

void sort(struct node \*f)

{

ele swap;

struct node \*temp,\*t; t=f->next;

while(t!=NULL)

{

temp=t->next; while(temp!=NULL)

{

if(t->e.start>temp->e.start)

{

swap=t->e;

t->e=temp->e; temp->e=swap;

}

temp=temp->next;

}

t=t->next;

}

}

void firstFit(struct node \*f,struct node \*a,char \*p,int s)

{

struct node \*temp,\*pretemp; ele t;

strcpy(t.status,p); t.size=s;

temp=f->next; int flag=0;

while(temp!=NULL)

{

if(s<=temp->e.size)

{

t.start=temp->e.start; t.end=t.start+t.size; insert(a,t);

temp->e.start+=t.size;

temp->e.size=temp->e.end-temp->e.start; flag=1;

break;

}

temp=temp->next;

}

temp=f->next; pretemp=f; while(temp!=NULL)

{

if(temp->e.size==0)

{

pretemp->next=temp->next; free(temp);

break;

}

}

if(flag==0)

{

}

sort(f);

sort(a);

pretemp=temp; temp=temp->next;

printf("Not Alloted\n");

printf("\nFree Pool\n"); output(f);

printf("\nAlloted Memory\n"); output(a);

}

void bestFit(struct node \*f,struct node \*a,char \*p,int s)

{

struct node \*temp,\*pretemp; ele t;

strcpy(t.status,p); t.size=s;

temp=f->next; int flag=0; int max=10000;

while(temp!=NULL)

{

if(temp->e.size<max && temp->e.size>=s)

{

max=temp->e.size; flag=1;

}

temp=temp->next;

}

temp=f->next; while(temp!=NULL)

{

if(temp->e.size==max && max!=10000)

{

t.start=temp->e.start; t.end=t.start+t.size; insert(a,t);

temp->e.start+=t.size;

temp->e.size=temp->e.end-temp->e.start; break;

}

temp=temp->next;

}

temp=f->next;

pretemp=f; while(temp!=NULL)

{

if(temp->e.size==0)

{

pretemp->next=temp->next; free(temp);

break;

}

}

if(flag==0)

{

}

sort(f);

sort(a);

pretemp=temp; temp=temp->next;

printf("Not Alloted\n");

printf("\nFree Pool\n"); output(f);

printf("\nAlloted Memory\n"); output(a);

}

void worstFit(struct node \*f,struct node \*a,char \*p,int s)

{

struct node \*temp,\*pretemp; ele t;

strcpy(t.status,p); t.size=s;

temp=f->next; int flag=0; int max=0;

while(temp!=NULL)

{

if(temp->e.size>max && temp->e.size>=s)

{

max=temp->e.size; flag=1;

}

temp=temp->next;

}

temp=f->next; while(temp!=NULL)

{

if(temp->e.size==max && max!=0)

{

t.start=temp->e.start; t.end=t.start+t.size; insert(a,t);

temp->e.start+=t.size;

temp->e.size=temp->e.end-temp->e.start; break;

}

temp=temp->next;

}

temp=f->next; pretemp=f; while(temp!=NULL)

{

if(temp->e.size==0)

{

pretemp->next=temp->next; free(temp);

break;

}

}

if(flag==0)

{

}

sort(f);

sort(a);

pretemp=temp; temp=temp->next;

printf("Not Alloted\n");

printf("\nFree Pool\n"); output(f);

printf("\nAlloted Memory\n"); output(a);

}

void dealloc(struct node \*f,struct node \*a,char \*p)

{

struct node \*temp,\*pretemp; temp=a->next;

pretemp=a; while(temp!=NULL)

{

if(strcmp(temp->e.status,p)==0)

{

strcpy(temp->e.status,"H "); insert(f,temp->e);

pretemp->next=temp->next; free(temp);

break;

}

pretemp=temp; temp=temp->next;

}

sort(f);

sort(a);

printf("\nFree Pool\n"); output(f);

printf("\nAlloted Memory\n"); output(a);

}

struct node\* merge(struct node \*f,struct node \*a)

{

struct node \*new,\*temp,\*temp1;

new=(struct node\*)malloc(sizeof(struct node)); new->next=NULL;

temp=f->next; temp1=a->next; while(temp!=NULL)

{

insert(new,temp->e); temp=temp->next;

}

while(temp1!=NULL)

{

}

sort(new); return new;

}

insert(new,temp1->e); temp1=temp1->next;

struct node \*coalesce(struct node \*f)

{

struct node \*temp,\*temp1; temp=f->next;

int s,count=0; while(temp!=NULL)

{

if(temp->next!=NULL)

temp1=temp->next;

else

break;

if(strcmp(temp->e.status,temp1->e.status)==0 && temp-

>e.end==temp1->e.start)

{

temp->e.end=temp1->e.end;

temp->e.size=temp->e.end-temp->e.start; temp->next=temp1->next;

free(temp1); temp=f->next;

}

else

temp=temp->next;

}

return f;

}

void main()

{

struct node \*allot,\*free; struct node \*merged;

allot=(struct node\*)malloc(sizeof(struct node)); allot->next=NULL;

free=(struct node\*)malloc(sizeof(struct node)); free->next=NULL;

int n;

printf("Enter number of partitions : "); scanf("%d",&n);

int i; for(i=0;i<n;i++)

{

ele temp;

printf("\nStart address for partition %d : ",i+1); scanf("%d",&temp.start);

printf("End address for partition %d : ",i+1); scanf("%d",&temp.end);

temp.size=temp.end-temp.start; strcpy(temp.status,"H ");

}

again:

insert(free,temp);

{

int fit;

printf("\n\n1.First Fit\n2.Best Fit\n3.Worst Fit\n"); scanf("%d",&fit);

if(fit==1)

printf("\n\t\tFirst Fit\n");

else if(fit==2)

printf("\n\t\tBest Fit\n"); else if(fit==3)

printf("\n\t\tWorst Fit\n");

int ch=1; while(ch!=6)

{

printf("\n\n1.Allocate\n2.De- Allocate\n3.Display\n4.Coalescing Holes\n5.Back\n6.Exit\n");

scanf("%d",&ch); switch(ch)

");

");

{

case 1:

{

char pid[10]; int s;

printf("Enter process id : scanf("%s",pid); printf("Enter size : scanf("%d",&s); if(fit==1)

firstFit(free,allot,pid,s);

else if(fit==2)

bestFit(free,allot,pid,s);

else if(fit==3)

");

worstFit(free,allot,pid,s);

}

case 2:

{

}

case 3:

{

break;

char pid[10]; printf("Enter process id :

scanf("%s",pid);

dealloc(free,allot,pid); break;

merged=merge(free,allot);

printf("\nFree Pool\n"); output(free);

Memory\n");

printf("\nAlloted output(allot);

Memory\n");

}

case 4:

{

printf("\nPhysical

output(merged); break;

free=coalesce(free); printf("\nFree Pool\n");

output(free);

Memory\n");

printf("\nAlloted output(allot);

Memory\n");

}

}

}

case 5:

{

}

case 6:

{

}

default:

}

printf("\nPhysical

merged=merge(free,allot); output(merged);

break;

goto again; break;

printf("Thank You\n"); break;

printf("Invalid Input\n");

}

/\* Output:

Enter number of partitions : 5

|  |  |  |
| --- | --- | --- |
| Start address for partition | 1 : | 100 |
| End address for partition 1 | : | 110 |
| Start address for partition | 2 : | 110 |
| End address for partition 2 | : | 112 |
| Start address for partition | 3 : | 112 |
| End address for partition 3 | : | 117 |
| Start address for partition | 4 : | 117 |
| End address for partition 4 | : | 120 |
| Start address for partition | 5 : | 120 |
| End address for partition 5 | : | 125 |

1. First Fit
2. Best Fit
3. Worst Fit 1

First Fit

1. Allocate
2. De-Allocate
3. Display
4. Coalescing Holes
5. Back
6. Exit 1

Enter process id : p1 Enter size : 5

Free Pool

| H | H | H | H | H |

105 110 112

117 120 125

Alloted Memory

| p1 |

100 105

1. Allocate
2. De-Allocate
3. Display
4. Coalescing Holes
5. Back
6. Exit 1

Enter process id : p2 Enter size : 4

Free Pool

| H | H | H | H | H |

109 110 112

117 120 125

Alloted Memory

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| | | p1 | | | p2 | | |
| 100 |  | 105 |  | 109 |

1. Allocate
2. De-Allocate
3. Display
4. Coalescing Holes
5. Back
6. Exit 1

Enter process id : p3 Enter size : 5

Free Pool

| H | H |

H | H |

109 110 117

120 125

Alloted Memory

| p1 | p2 |

p3 |

100 105 112

117

1. Allocate
2. De-Allocate
3. Display
4. Coalescing Holes
5. Back
6. Exit 3

Free Pool

| H | H |

H | H |

109 110 117

120 125

Alloted Memory

| p1 | p2 |

p3 |

100 105 112

117

Physical Memory

| p1 | p2 | H | H | p3 |

H | H |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 110 | 100 | 112 | 105 | 117 | 109 |
| 120 |  | 125 |  | | |
| 1. | Allocate |  |
| 2. | De-Allocate |  |
| 3. | Display |  |
| 4. | Coalescing Holes |  |
| 5. | Back |  |
| 6. | Exit |  |
| 2 |  |  |

Enter process id : p2 Free Pool

| H | H | H | H | H |

105 109 110

117 120 125

Alloted Memory

| p1 | p3 |

100 112 117

1. Allocate
2. De-Allocate
3. Display
4. Coalescing Holes
5. Back
6. Exit 3

Free Pool

| H | H | H | H | H |

105 109 110

117 120 125

Alloted Memory

| p1 | p3 |

100 112 117

Physical Memory

| p1 | H | H | H | p3 |

H | H |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 100 |  |  | 105 |  |  | 109 |
| 110 | 112 |  |  | 117 |  |  |
| 120 | 125 |  |  |  |  |  |
| 1. Allocate 2. De-Allocate 3. Display 4. Coalescing Holes 5. Back 6. Exit 4 |  |  |  |  |  |  |
| Free Pool |  |  |  |  |  |  |
| | |  | H | | |  | H | | |
| 105  Alloted Memory |  |  | 117 |  |  | 125 |
| | |  | p1 | | |  | p3 | | |

100 112 117

Physical Memory

| p1 | H |

p3 | H |

100 105 112

117 125

1. Allocate
2. De-Allocate
3. Display
4. Coalescing Holes
5. Back
6. Exit 6

Thank You

\*/