EX:8 MEMORY ALLOCATION TECHNIQUES

-S.Vishakan CSE-C 18 5001 196

Source Code:

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

struct Frame{       //Node for Linked List
   int pid;
   int size;
   struct Frame *next;
};

typedef struct Frame frame;

int partitions;

frame *head = NULL;     //Maintaining Head & Tail Pointers for Linked List
frame *tail = NULL;

void    insertNode(frame **head, frame **tail, int pid, int size);
void    initMemory();
void    printMemory(frame **head);
int     firstFit(frame **head, int pid, int size);
int     bestFit(frame **head, frame **tail, int pid, int size);
int     worstFit(frame **head, frame **tail, int pid, int size);
int     deAllocate(frame **head, int pid);
void    coalesceFree(frame **head);

int main(void){
   int opt = 1, i = 0, alloc_opt = 0, pid, size, status = 0;

   initMemory();
   printMemory(&head);

   while(opt != 0){
      printf("\n\n\t\tMain Menu: ");
      printf("\n\t1. Allocate a Process\n\t2. Deallocate a Process\n\t3. Display current memory status\
n\t4. Coalesce free memory\n\t0. Exit\n\tYour Choice -> ");
      scanf("%d", &opt);

      switch(opt){
         case 1:
            printf("\nEnter the Process ID: ");
```

```
            scanf("%d", &pid);
            printf("\nEnter the Process Size: ");
            scanf("%d", &size);
            printf("\n\t\tAllocation Strategy: \n\t1. First Fit\n\t2. Best Fit\n\t3. Worst Fit\n\t0. Exit\n\
tYour Choice -> ");
            scanf("%d", &alloc_opt);
            switch(alloc_opt){
               case 1:
                  status = firstFit(&head, pid, size);
                  if(!status){
                     printf("\nProcess cannot be allocated.");
                  }
                  else{
                     printf("\nProcess allocated.");
                  }
                  break;

               case 2:
                  status = bestFit(&head, &tail, pid, size);
                  if(!status){
                     printf("\nProcess cannot be allocated.");
                  }
                  else{
                     printf("\nProcess allocated.");
                  }
                  break;

               case 3:
                  status = worstFit(&head, &tail, pid, size);
                  if(!status){
                     printf("\nProcess cannot be allocated.");
                  }
                  else{
                     printf("\nProcess allocated.");
                  }
                  break;

               case 0:
                  break;

               default:
                  printf("\nInvalid Option.");
                  break;
            }

            break;

         case 2:
            printf("\nEnter PID to deallocate: ");
            scanf("%d", &pid);
            status = deAllocate(&head, pid);
            if(!status){
```

```
                printf("\nProcess P%d not found.", pid);
            }
            else{
                printf("\nProcess P%d deallocated.", pid);
            }
            break;

        case 3:
            printMemory(&head);
            break;

        case 4:
            coalesceFree(&head);
            printMemory(&head);
            break;

        case 0:
            printf("\n\tThank You!\n");
            break;

        default:
            printf("\n\tInvalid Choice.");
            break;
      }
   };
}


void insertNode(frame **head, frame **tail, int pid, int size){     //Insert a Node into the Linked
List at tail end
   frame *new_node = (frame *) malloc(sizeof(frame));
   new_node->pid = pid;
   new_node->size = size;
   new_node->next = NULL;

   if((*head) == NULL){
      (*head) = (*tail) = new_node;
   }
   else{
      (*tail)->next = new_node;
      (*tail) = new_node;
   }
}


void initMemory(){      //Initialising the memory partition table
   int i = 0, start, end, size;

   printf("\nEnter the Memory Representation: ");
   printf("\nEnter the no. of partitions in memory: ");
   scanf("%d", &partitions);
```

```
    for(i = 0; i < partitions; i++){
        printf("\nEnter the starting and ending address of partition %d: ", i);
        scanf("%d %d", &start, &end);
        size = end - start;
        insertNode(&head, &tail, -1, size);
    }
}


void printMemory(frame **head){     //Printing the memory partition table
    frame *node = (*head);
    printf("\n\t\tMemory Partition Table\n");
    while(node != NULL){
        if(node->pid == -1){
            printf("\nStatus: Free\tSize: %d", node->size);
        }
        else{
            printf("\nStatus: P%d\tSize: %d", node->pid, node->size);
        }
        node = node->next;
    }
}


int firstFit(frame **head, int pid, int size){      //First Fit Algorithm
    frame *temp = *head;
    frame *new_node = (frame *)malloc(sizeof(frame));
    int diff, flag = 1;

    while(temp != NULL && size > temp->size){
        temp = temp->next;
    }

    if(temp == NULL){
        flag = 0;
    }
    else if(size != temp->size){
        diff = temp->size - size;
        new_node->pid = -1;
        new_node->size = diff;
        temp->size = size;
        temp->pid = pid;
        new_node->next = temp->next;
        temp->next = new_node;
    }
    else{
        temp->pid = pid;
        free(new_node);
    }
    return flag;

}
```

```
int bestFit(frame **head, frame **tail, int pid, int size){     //Best Fit Algorithm
   int new_size;
   frame *temp = *head;
   frame *temp1 = NULL;

   while(temp != NULL){
      if((temp->pid == -1) && (temp->size >= size)){
         if(temp1 == NULL){
            temp1 = temp;
         }
         else{
            if(temp1->size > temp->size){
               temp1 = temp;
            }
         }
      }
      temp = temp->next;
   }

   new_size = temp1->size - size;

   if(new_size > 0){
      frame *new_node = (frame *)malloc(sizeof(frame));
      new_node->next = temp1->next;
      temp1->next = new_node;
      temp1->pid = pid;
      temp1->size = size;
      new_node->size = new_size;
      new_node->pid = -1;
   }
   else{
      temp1->size = size;
      temp1->pid = pid;
   }
   if(temp1 == NULL){
      return 0;
   }

   return 1;

}




int worstFit(frame **head, frame **tail, int pid, int size){       //Worst Fit Algorithm
   int new_size;
   frame *temp = *head;
   frame *temp1 = NULL;
```

```
    while(temp != NULL){
        if((temp->pid == -1) && (temp->size > size)){
            if(temp1 == NULL){
                temp1 = temp;
            }
            else if(temp1->size < temp->size){
                temp1 = temp;
            }
        }
        temp = temp->next;
    }

    new_size = temp1->size - size;

    if(new_size > 0){
        temp1->pid = pid;
        frame *new_node = (frame *)malloc(sizeof(frame));
        new_node->size = new_size;
        new_node->pid = -1;
        new_node->next = temp1->next;
        temp1->size = size;
        temp1->next = new_node;
    }
    else if(new_size == 0){
        temp1->pid = pid;
    }
    else{
        return 0;
    }

    return 1;
}


int deAllocate(frame **head, int pid){      //Deallocating a process
    frame *temp = *head;
    int flag = 0;
    while(temp != NULL){
        if(temp->pid == pid){
            temp->pid = -1;
            flag = 1;
        }
        else{
            temp = temp->next;
        }
    }

    return flag;
}
```

```
void coalesceFree(frame **head){        //Coalescing the free space to prevent fragmentation
   frame *temp = *head;
   frame *temp1 = NULL;

   while(temp != NULL){
      if(temp->pid == -1){
         temp1 = temp->next;
         while(temp1 != NULL && temp1->pid == -1){
            temp->size = temp->size + temp1->size;
            temp->next = temp1->next;
            temp1 = temp->next;
         }
         temp = temp1;
      }
      else{
         temp = temp->next;
      }
   }
}
```

OUTPUT:

```
vishakan@Legion:~/Desktop/Operating-Systems/Ex8 Memory Allocation$ gcc Allocation.c -o a
vishakan@Legion:~/Desktop/Operating-Systems/Ex8 Memory Allocation$ ./a

Enter the Memory Representation:
Enter the no. of partitions in memory: 5

Enter the starting and ending address of partition 0: 0 80

Enter the starting and ending address of partition 1: 81 100

Enter the starting and ending address of partition 2: 101 200

Enter the starting and ending address of partition 3: 201 250

Enter the starting and ending address of partition 4: 251 280

            Memory Partition Table

Status: Free    Size: 80
Status: Free    Size: 19
Status: Free    Size: 99
Status: Free    Size: 49
Status: Free    Size: 29

            Main Menu:
        1. Allocate a Process
        2. Deallocate a Process
        3. Display current memory status
        4. Coalesce free memory
        0. Exit
        Your Choice -> 1

Enter the Process ID: 1

Enter the Process Size: 90

            Allocation Strategy:
        1. First Fit
        2. Best Fit
        3. Worst Fit
        0. Exit
        Your Choice -> 1

Process allocated.

            Main Menu:
        1. Allocate a Process
        2. Deallocate a Process
        3. Display current memory status
        4. Coalesce free memory
```

0. Exit
Your Choice -> 1

Enter the Process ID: 2

Enter the Process Size: 29

Allocation Strategy:
1. First Fit
2. Best Fit
3. Worst Fit
0. Exit
Your Choice -> 2

Process allocated.

Main Menu:
1. Allocate a Process
2. Deallocate a Process
3. Display current memory status
4. Coalesce free memory
0. Exit
Your Choice -> 1

Enter the Process ID: 3

Enter the Process Size: 20

Allocation Strategy:
1. First Fit
2. Best Fit
3. Worst Fit
0. Exit
Your Choice -> 3

Process allocated.

Main Menu:
1. Allocate a Process
2. Deallocate a Process
3. Display current memory status
4. Coalesce free memory
0. Exit
Your Choice -> 1

Enter the Process ID: 4

Enter the Process Size: 40

Allocation Strategy:
1. First Fit
2. Best Fit

3. Worst Fit
0. Exit
Your Choice -> 2

Process allocated.

Main Menu:
1. Allocate a Process
2. Deallocate a Process
3. Display current memory status
4. Coalesce free memory
0. Exit
Your Choice -> 3

Memory Partition Table

Status: P3 Size: 20
Status: Free Size: 60
Status: Free Size: 19
Status: P1 Size: 90
Status: Free Size: 9
Status: P4 Size: 40
Status: Free Size: 9
Status: P2 Size: 29

Main Menu:
1. Allocate a Process
2. Deallocate a Process
3. Display current memory status
4. Coalesce free memory
0. Exit
Your Choice -> 4

Memory Partition Table

Status: P3 Size: 20
Status: Free Size: 79
Status: P1 Size: 90
Status: Free Size: 9
Status: P4 Size: 40
Status: Free Size: 9
Status: P2 Size: 29

Main Menu:
1. Allocate a Process
2. Deallocate a Process
3. Display current memory status
4. Coalesce free memory
0. Exit
Your Choice -> 2

Enter PID to deallocate: 3

Process P3 deallocated.

Main Menu:
1. Allocate a Process
2. Deallocate a Process
3. Display current memory status
4. Coalesce free memory
0. Exit
Your Choice -> 3

Memory Partition Table

Status: Free Size: 20
Status: Free Size: 79
Status: P1 Size: 90
Status: Free Size: 9
Status: P4 Size: 40
Status: Free Size: 9
Status: P2 Size: 29

Main Menu:
1. Allocate a Process
2. Deallocate a Process
3. Display current memory status
4. Coalesce free memory
0. Exit
Your Choice -> 4

Memory Partition Table

Status: Free Size: 99
Status: P1 Size: 90
Status: Free Size: 9
Status: P4 Size: 40
Status: Free Size: 9
Status: P2 Size: 29

Main Menu:
1. Allocate a Process
2. Deallocate a Process
3. Display current memory status
4. Coalesce free memory
0. Exit
Your Choice -> 0

Thank You!