EX:9 PAGING TECHNIQUE

-S.Vishakan CSE-C 18 5001 196

Source Code:

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

struct PageTable
{
   int page_no;
   int frame_no;
   struct PageTable *next;
};

struct FreeFrame
{
   int frame_no;
   struct FreeFrame *next;
};

typedef struct PageTable pagetable;
typedef struct FreeFrame freeframe;

pagetable *process[100];
freeframe *head;

int page_size, mem_size, no_frames, no_free, frames[100];

void initMemory();
void allocateProcess();
void deallocateProcess();
void printPageTable(pagetable *head);
void showAllPageTables();
void showFreeFrames();
void addressMapper();

int main(void)
{
   initMemory();
   showFreeFrames();
   printf("\nNo. of free frames left: %d", no_free);

   int opt = 1;
   while (opt != 0)
```

```
    {
        printf("\n\n\t\tMain Menu\n");
        printf("\n\t1. Allocate a Process\n\t2. Deallocate a Process\n\t3. Display Page Table\n\t4.
Display Free Frames\n\t5. Address Mapping\n\t0. Exit\n\tYour Choice -> ");
        scanf("%d", &opt);

        switch (opt)
        {
        case 1:
            allocateProcess();
            break;
        case 2:
            deallocateProcess();
            break;
        case 3:
            showAllPageTables();
            break;
        case 4:
            showFreeFrames();
            break;
        case 5:
            addressMapper();
            break;
        case 0:
            printf("\n\t\tThank You!\n");
            break;
        default:
            printf("\n\tInvalid Option.");
            break;
        }
    }
}

void initMemory()
{   //Initialising page table and frame table
    int i = 0;
    freeframe *temp, *prev;

    printf("Enter the total physical memory (in KB) : ");
    scanf("%d", &mem_size);
    printf("\nEnter the page size (in KB) : ");
    scanf("%d", &page_size);
    no_frames = mem_size / page_size;
    printf("\nNo. of frames: %d", no_frames);

    for (i = 0; i < no_frames / 2; i++)
    {       //simulating the fact that some frames are already occupied
        int frame_no = random() % 737 % no_frames;
        frames[frame_no] = 1;
        no_free++;
        temp = (freeframe *)malloc(sizeof(freeframe));
        temp->frame_no = frame_no;
```

```
        temp->next = NULL;
        if (no_free == 1)
        {
            head = temp;
        }
        else
        {
            prev->next = temp;
        }
        prev = temp;
    }
}

void printPageTable(pagetable *head)
{   //printing the page table for a particular process
    pagetable *temp;

    if (head != NULL)
    {
        printf("\n-------------------------");
    }

    for (temp = head; temp != NULL; temp = temp->next)
    {
        printf("\nPage : %d\tFrame : %d", temp->page_no, temp->frame_no);

    }

    printf("\n-------------------------\n");
}

void showAllPageTables()
{   //printing the page table for all processes
    int i = 0;
    printf("\n\t\tPage Table");
    for (i = 0; i < 10; i++)
    {
        if(process[i] != NULL){
            printf("\n\tPID : %d", i);
            printPageTable(process[i]);
        }
    }
}

void allocateProcess()
{   //allocating frames for a process
    int pid, mem, no_pages, i = 0;
    pagetable *temp, *phead, *prev;

    printf("\nEnter Process ID: ");
    scanf("%d", &pid);
    printf("\nEnter the Memory Required (in KB) : ");
```

```
    scanf("%d", &mem);
    no_pages = mem / page_size;

    if (mem * 1.0 / page_size > no_pages)
    {
        no_pages++;
    }

    printf("\nNo. of Pages: %d", no_pages);

    if (no_pages > no_free)
    {
        printf("\nProcess cannot be allocated as there are only %d free frames left.", no_free);
        return;
    }

    for (i = 0; i < no_pages; i++)
    {   //obtaining frames from the free frame list through the header node
        temp = (pagetable *)malloc(sizeof(pagetable));
        temp->page_no = i;
        temp->frame_no = head->frame_no;
        temp->next = NULL;
        freeframe *frame = head;
        head = head->next;
        free(frame);
        no_free--;

        if (i == 0)
        {
            phead = temp;
        }
        else
        {
            prev->next = temp;
        }

        prev = temp;
    }
    process[pid] = phead;
    printPageTable(process[pid]);
    printf("\nNo. of free frames left: %d", no_free);
}

void deallocateProcess()
{   //deallocate frames for a particular process
    int pid, frame;
    pagetable *temppage, *del, *pthead;
    freeframe *ffhead, *tempframe;

    printf("\nEnter the Process ID: ");
    scanf("%d", &pid);
```

```
    pthead = process[pid];
    if (pthead == NULL)
    {
        printf("\nProcess %d has not been allocated.", pid);
        return;
    }

    ffhead = head;
    if(ffhead != NULL){     //going to the end of the linked list
        for (ffhead = head; ffhead->next != NULL; ffhead = ffhead->next)
            ;
    }

    for (temppage = pthead; temppage != NULL;)
    {   //deleting the pages allocated to the process
        del = temppage;
        frame = del->frame_no;
        tempframe = (freeframe *)malloc(sizeof(freeframe));
        tempframe->frame_no = frame;
        tempframe->next = NULL;
                            //appending the deallocated frame to the free frame list
        if (ffhead != NULL)
        {
            ffhead->next = tempframe;
            ffhead = tempframe;
        }
        else
        {       //if the free frame list is empty
            ffhead = tempframe;
            head = ffhead;
        }

        temppage = temppage->next;
        no_free++;
        free(del);
    }

    process[pid] = NULL;
    printf("\nSuccessfully deallocated Process %d.", pid);
}

void addressMapper()
{   //to find the physical address of a process given the logical address
    int pid, logical_addr, page_num, frame_num, offset, phys_addr, i = 0;
    pagetable *fhead, *temp;

    printf("\nEnter Process ID: ");
    scanf("%d", &pid);
    printf("\nEnter Logical Address of %d : ", pid);
    scanf("%d", &logical_addr);

    page_num = logical_addr / (page_size * 1024);
```

```
    offset = logical_addr % (page_size * 1024);

    fhead = process[pid];
    temp = fhead;

    for (i = 0; i < page_num; i++)
    {
        temp = temp->next;
    }

    frame_num = temp->frame_no;
    phys_addr = frame_num * page_size * 1024 + offset;
    printf("\nPhysical Address of %d is : %d.", pid, phys_addr);
}

void showFreeFrames()
{   //listing the free frames
    freeframe *temp;
    printf("\n\t\tFree Frame List: \n");
    for (temp = head; temp != NULL; temp = temp->next)
    {
        printf("%d ", temp->frame_no);
    }
    printf("\n");
}
```

OUTPUT:

```
(base) vishakan@Legion:~/Desktop/Operating-Systems/Ex9 Paging$ ./p
Enter the total physical memory (in KB) : 20

Enter the page size (in KB) : 2

No. of frames: 10
            Free Frame List:
2 3 1 9 5

No. of free frames left: 5

            Main Menu

        1. Allocate a Process
        2. Deallocate a Process
        3. Display Page Table
        4. Display Free Frames
        5. Address Mapping
        0. Exit
        Your Choice -> 1

Enter Process ID: 1

Enter the Memory Required (in KB) : 5

No. of Pages: 3
------------------------
Page : 0        Frame : 2
Page : 1        Frame : 3
Page : 2        Frame : 1
------------------------

No. of free frames left: 2


            Main Menu

        1. Allocate a Process
        2. Deallocate a Process
        3. Display Page Table
        4. Display Free Frames
        5. Address Mapping
        0. Exit
        Your Choice -> 1

Enter Process ID: 2

Enter the Memory Required (in KB) : 5

No. of Pages: 3
```

Process cannot be allocated as there are only 2 free frames left.

Main Menu

1. Allocate a Process
2. Deallocate a Process
3. Display Page Table
4. Display Free Frames
5. Address Mapping
0. Exit
Your Choice -> 1

Enter Process ID: 2

Enter the Memory Required (in KB) : 4

No. of Pages: 2

Page : 0 Frame : 9
Page : 1 Frame : 5

No. of free frames left: 0

Main Menu

1. Allocate a Process
2. Deallocate a Process
3. Display Page Table
4. Display Free Frames
5. Address Mapping
0. Exit
Your Choice -> 3

Page Table
PID : 1

Page : 0 Frame : 2
Page : 1 Frame : 3
Page : 2 Frame : 1

PID : 2

Page : 0 Frame : 9
Page : 1 Frame : 5

Main Menu

1. Allocate a Process
2. Deallocate a Process
3. Display Page Table
4. Display Free Frames
5. Address Mapping
0. Exit

Your Choice -> 4

Free Frame List:

Main Menu

1. Allocate a Process
2. Deallocate a Process
3. Display Page Table
4. Display Free Frames
5. Address Mapping
0. Exit

Your Choice -> 2

Enter the Process ID: 2

Successfully deallocated Process 2.

Main Menu

1. Allocate a Process
2. Deallocate a Process
3. Display Page Table
4. Display Free Frames
5. Address Mapping
0. Exit

Your Choice -> 4

Free Frame List:

9 5

Main Menu

1. Allocate a Process
2. Deallocate a Process
3. Display Page Table
4. Display Free Frames
5. Address Mapping
0. Exit

Your Choice -> 5

Enter Process ID: 1

Enter Logical Address of 1 : 2048

Physical Address of 1 is : 6144.

Main Menu

1. Allocate a Process
2. Deallocate a Process
3. Display Page Table
4. Display Free Frames
5. Address Mapping
0. Exit

Your Choice -> 0

Thank You!