**Name : Aradhya Sakalley**

**Sapid : 60004210253**

**Batch : C3-2**

**Branch : Computer Engineering**

**Experiment 3**

**Aim :** To implement Page Replacement (FIFO,LRU,OPT) and Memory Allocation (First Fit, Best Fit and Worst Fit) techniques.

**Theory:**

Page Replacement Techniques:

* FIFO (First-In-First-Out): In FIFO page replacement, the oldest page in memory is the one that gets replaced when a new page needs to be loaded. It’s a simple and easy-to-implement algorithm but may not always provide the best performance.
* LRU (Least Recently Used): LRU replaces the page that has not been used for the longest time. It maintains a queue of pages in memory and moves the most recently accessed page to the front. LRU typically provides better performance than FIFO but can be more complex to implement efficiently.
* OPT (Optimal): OPT replaces the page that will not be used for the longest time in the future. It’s an idealized algorithm that requires knowledge of future page references, which is often not possible. OPT serves as a performance benchmark for other page replacement algorithms.

Memory Allocation Techniques:

* First Fit: In First Fit memory allocation, the operating system assigns the first available block of memory that is large enough to accommodate the process. It’s straightforward but can lead to memory fragmentation.
* Best Fit: Best Fit memory allocation assigns the smallest available block of memory that can accommodate the process. This minimizes wasted memory space but may lead to fragmentation and requires searching for the best-fit block.
* Worst Fit: Worst Fit memory allocation assigns the largest available block of memory to the process. This can result in the least amount of fragmentation but may lead to inefficient use of memory as large gaps can occur.

These techniques are used in operating systems to manage memory and page replacement, ensuring efficient utilization of resources and optimizing system performance.

**Page replacement :**

1. **FIFO :**

**Code :**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

int main(){

    int n,i,j,frames,pageFaults=0,s;

    printf("enter the number of requests:");

    scanf("%d",&n);

    int incomingStream[n];

    printf("enter the incoming stream of requests: ");

    for(i=0;i<n;i++){

        scanf("%d",&incomingStream[i]);

    }

    printf("enter the number of frames:");

    scanf("%d",&frames);

    int temp[frames];

    for(i=0;i<frames;i++){

        temp[i] = -1;

    }

    //FIFO ALGO

    for(i=0;i<n;i++){

        s = 0;

        for(j=0;j<frames;j++){

            if(incomingStream[i] == temp[j]){

                s++;

                pageFaults--;

            }

        }

        pageFaults++;

        if(pageFaults <= frames && (s == 0)){

            temp[i] = incomingStream[i];

        }

        else if(s == 0){

            temp[(pageFaults-1) % frames] = incomingStream[i];

        }

        printf("\n");

        printf("%d\t\t\t",incomingStream[i]);

        for(j = 0; j < frames; j++)

        {

            if(temp[j] != -1)

                printf(" %d\t\t\t", temp[j]);

            else

                printf(" - \t\t\t");

        }

        printf("\n");

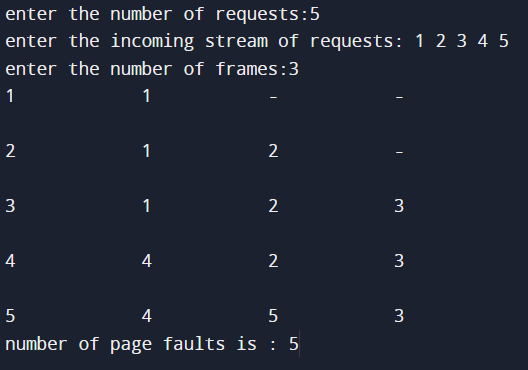
    }

    printf("number of page faults is : %d",pageFaults);

    return 0;

}

**Output :**

****

1. **LRU :**

**Code :**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_FRAMES 100

#define MAX\_REQUESTS 1000

int main() {

    int requests[MAX\_REQUESTS], frames[MAX\_FRAMES];

    int n, m, i, j, k, p, q, min, min\_idx, page\_faults;

    printf("Enter the number of requests: ");

    scanf("%d", &n);

    printf("Enter the requests: ");

    for (i = 0; i < n; i++) {

        scanf("%d", &requests[i]);

    }

    printf("Enter the number of frames: ");

    scanf("%d", &m);

    for (i = 0; i < m; i++) {

        frames[i] = -1;

    }

    page\_faults = 0;

    for (i = 0; i < n; i++) {

        p = requests[i];

        q = -1;

        for (j = 0; j < m; j++) {

            if (frames[j] == p) {

                q = j;

                break;

            }

        }

        if (q == -1) {

            if (i < m) {

                q = i;

            } else {

                min = 99999;

                for (j = 0; j < m; j++) {

                    k = frames[j];

                    p = 0;

                    for (p = i - 1; p >= 0; p--) {

                        if (requests[p] == k) {

                            break;

                        }

                    }

                    if (p < min) {

                        min = p;

                        min\_idx = j;

                    }

                }

                q = min\_idx;

            }

            frames[q] = requests[i];

            page\_faults++;

        }

        printf("\n%d\t", requests[i]);

        for (j = 0; j < m; j++) {

            if (frames[j] == -1) {

                printf("-\t");

            } else {

                printf("%d\t", frames[j]);

            }

        }

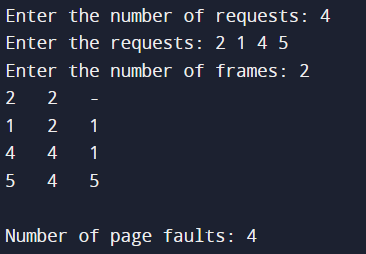
    }

    printf("\n\nNumber of page faults: %d\n", page\_faults);

    return 0;

}

**Output :**

****

1. **OPTIMAL :**

**Code :**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

int main() {

    int n, i, j, frames, pageFaults = 0, s, k, max, maxIndex;

    printf("Enter the number of requests: ");

    scanf("%d", &n);

    int incomingStream[n];

    printf("Enter the incoming stream of requests: ");

    for(i = 0; i < n; i++) {

        scanf("%d", &incomingStream[i]);

    }

    printf("Enter the number of frames: ");

    scanf("%d", &frames);

    int temp[frames];

    for(i = 0; i < frames; i++) {

        temp[i] = -1;

    }

    // Optimal Algorithm

    for(i = 0; i < n; i++) {

        s = 0;

        for(j = 0; j < frames; j++) {

            if(incomingStream[i] == temp[j]) {

                s++;

                pageFaults--;

            }

        }

        pageFaults++;

        if(pageFaults <= frames && (s == 0)) {

            temp[i] = incomingStream[i];

        } else if(s == 0) {

            max = INT\_MIN;

            for(j = 0; j < frames; j++) {

                int found = 0;

                for(k = i + 1; k < n; k++) {

                    if(incomingStream[k] == temp[j]) {

                        found = 1;

                        if(k > max) {

                            max = k;

                            maxIndex = j;

                        }

                        break;

                    }

                }

                if(found == 0) {

                    maxIndex = j;

                    break;

                }

            }

            temp[maxIndex] = incomingStream[i];

        }

        printf("\n");

        printf("%d\t\t\t", incomingStream[i]);

        for(j = 0; j < frames; j++) {

            if(temp[j] != -1) {

                printf(" %d\t\t\t", temp[j]);

            } else {

                printf(" - \t\t\t");

            }

        }

        printf("\n");

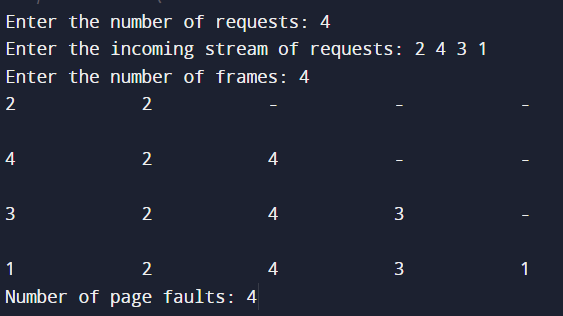
    }

    printf("Number of page faults: %d", pageFaults);

    return 0;

}

**Output :**

****

**Memory allocation techniques :**

1. **Best Fit :**

**Code :**

#include <stdio.h>

void bubbleSort(int arr[], int n)

{

    int i, j, temp;

    for (i = 0; i < n - 1; i++)

    {

        for (j = 0; j < n - i - 1; j++)

        {

            if (arr[j] > arr[j + 1])

            {

                temp = arr[j];

                arr[j] = arr[j + 1];

                arr[j + 1] = temp;

            }

        }

    }

}

void bestFit(int blockSize[], int m, int processSize[], int n, int holeSize[], int sortedBlock[])

{

    int i, j;

    int allocation[n];

    for (i = 0; i < n; i++)

    {

        allocation[i] = -1;

    }

    for (i = 0; i < n; i++) // n -> number of processes

    {

        int bestBlock;

        for (j = 0; j < m; j++) // m -> number of blocks

        {

            if (sortedBlock[j] != -1 && sortedBlock[j] >= processSize[i])

            {

                bestBlock = sortedBlock[j];

                sortedBlock[j] = -1;

                break;

            }

        }

        for (j = 0; j < m; j++) // m -> number of blocks

        {

            if (bestBlock == holeSize[j])

            {

                allocation[i] = j; // memory block assigned to process

                holeSize[j] -= processSize[i]; // reduce available memory in this block

                break; // go to the next process in the queue

            }

        }

    }

    printf("\nProcess No.\tProcess Size\tAllocated Block no.\n");

    for (int i = 0; i < n; i++)

    {

        printf("%i\t\t", i + 1);

        printf("%i\t\t", processSize[i]);

        if (allocation[i] != -1)

            printf("%i", allocation[i] + 1);

        else

            printf("Not Allocated");

        printf("\n");

    }

    printf("\nBlock No.\tBlock Size\tHole Size\n");

    for (int i = 0; i < m; i++)

    {

        printf("%i\t\t", i + 1);

        printf("%i\t\t", blockSize[i]);

        if (holeSize[i] != blockSize[i])

            printf("%i", holeSize[i]);

        else

            printf("Unused block");

        // printf("%i", holeSize[i]);

        printf("\n");

    }

}

void main()

{

    int m, n;

    printf("Enter number of blocks in the memory: "); // m = 5

    scanf("%d", &m);

    printf("Enter number of processes in the input queue: "); // n = 4

    scanf("%d", &n);

    int blockSize[m], processSize[n], holeSize[m], sortedBlock[m];

    printf("Enter each block size: "); // {100, 500, 200, 300, 600}

    for (int i = 0; i < m; i++)

    {

        scanf("%d", &blockSize[i]);

        holeSize[i] = blockSize[i];

        sortedBlock[i] = blockSize[i];

    }

    printf("Enter each process size: "); // {212, 417, 112, 426}

    for (int i = 0; i < n; i++)

    {

        scanf("%d", &processSize[i]);

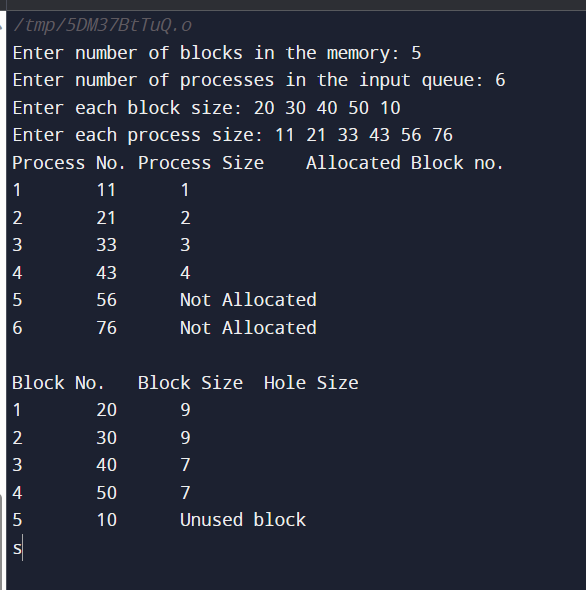
    }

    bubbleSort(sortedBlock, m);

    bestFit(blockSize, m, processSize, n, holeSize, sortedBlock);

}

**Output :**

****

1. **First Fit:**

**Code :**

#include <stdio.h>

void firstFit(int blockSize[], int m, int processSize[], int n, int holeSize[])

{

    int i, j;

    int allocation[n];

    for (i = 0; i < n; i++)

    {

        allocation[i] = -1;

    }

    for (i = 0; i < n; i++) // n -> number of processes

    {

        for (j = 0; j < m; j++) // m -> number of blocks

        {

            if (holeSize[j] == blockSize[j] && holeSize[j] >= processSize[i])

            {

                allocation[i] = j; // memory block assigned to process

                holeSize[j] -= processSize[i]; // reduce available memory in this block

                break; // go to the next process in the queue

            }

        }

    }

    printf("\nProcess No.\tProcess Size\tAllocated Block no.\n");

    for (int i = 0; i < n; i++)

    {

        printf("%i\t\t", i + 1);

        printf("%i\t\t", processSize[i]);

        if (allocation[i] != -1)

            printf("%i", allocation[i] + 1);

        else

            printf("Not Allocated");

        printf("\n");

    }

    printf("\nBlock No.\tBlock Size\tHole Size\n");

    for (int i = 0; i < m; i++)

    {

        printf("%i\t\t", i + 1);

        printf("%i\t\t", blockSize[i]);

        if (holeSize[i] != blockSize[i])

            printf("%i", holeSize[i]);

        else

            printf("Unused block");

        // printf("%i", holeSize[i]);

        printf("\n");

    }

}

void main()

{

    int m, n;

    printf("Enter number of blocks in the memory: "); // m = 5

    scanf("%d", &m);

    printf("Enter number of processes in the input queue: "); // n = 4

    scanf("%d", &n);

    int blockSize[m], processSize[n], holeSize[m];

    printf("Enter each block size: "); // {100, 500, 200, 300, 600}

    for (int i = 0; i < m; i++)

    {

        scanf("%d", &blockSize[i]);

        holeSize[i] = blockSize[i];

    }

    printf("Enter each process size: "); // {212, 417, 112, 426}

    for (int i = 0; i < n; i++)

    {

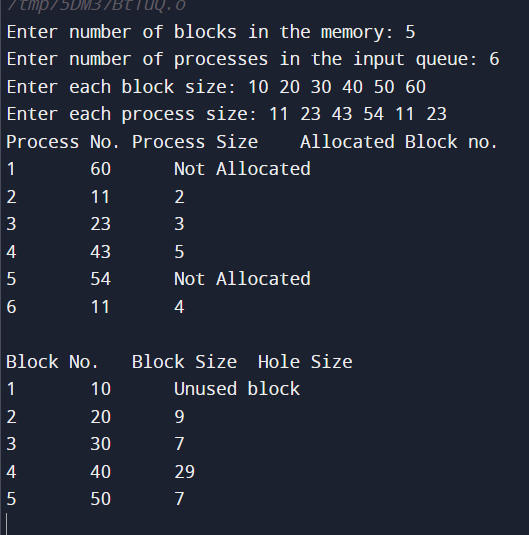
        scanf("%d", &processSize[i]);

    }

    firstFit(blockSize, m, processSize, n, holeSize);

}

**Output :**

****

1. **Worst Fit :**

**Code :**

#include <stdio.h>

void bubbleSort(int arr[], int n)

{

    int i, j, temp;

    for (i = 0; i < n - 1; i++)

    {

        for (j = 0; j < n - i - 1; j++)

        {

            if (arr[j] < arr[j + 1])

            {

                temp = arr[j];

                arr[j] = arr[j + 1];

                arr[j + 1] = temp;

            }

        }

    }

}

void bestFit(int blockSize[], int m, int processSize[], int n, int holeSize[], int sortedBlock[])

{

    int i, j;

    int allocation[n];

    for (i = 0; i < n; i++)

    {

        allocation[i] = -1;

    }

    for (i = 0; i < n; i++) // n -> number of processes

    {

        int bestBlock;

        for (j = 0; j < m; j++) // m -> number of blocks

        {

            if (sortedBlock[j] != -1 && sortedBlock[j] >= processSize[i])

            {

                bestBlock = sortedBlock[j];

                sortedBlock[j] = -1;

                break;

            }

        }

        for (j = 0; j < m; j++) // m -> number of blocks

        {

            if (bestBlock == holeSize[j])

            {

                allocation[i] = j; // memory block assigned to process

                holeSize[j] -= processSize[i]; // reduce available memory in this block

                break; // go to the next process in the queue

            }

        }

    }

    printf("\nProcess No.\tProcess Size\tAllocated Block no.\n");

    for (int i = 0; i < n; i++)

    {

        printf("%i\t\t", i + 1);

        printf("%i\t\t", processSize[i]);

        if (allocation[i] != -1)

            printf("%i", allocation[i] + 1);

        else

            printf("Not Allocated");

        printf("\n");

    }

    printf("\nBlock No.\tBlock Size\tHole Size\n");

    for (int i = 0; i < m; i++)

    {

        printf("%i\t\t", i + 1);

        printf("%i\t\t", blockSize[i]);

        if (holeSize[i] != blockSize[i])

            printf("%i", holeSize[i]);

        else

            printf("Unused block");

        // printf("%i", holeSize[i]);

        printf("\n");

    }

}

void main()

{

    int m, n;

    printf("Enter number of blocks in the memory: "); // m = 5

    scanf("%d", &m);

    printf("Enter number of processes in the input queue: "); // n = 4

    scanf("%d", &n);

    int blockSize[m], processSize[n], holeSize[m], sortedBlock[m];

    printf("Enter each block size: "); // {100, 500, 200, 300, 600}

    for (int i = 0; i < m; i++)

    {

        scanf("%d", &blockSize[i]);

        holeSize[i] = blockSize[i];

        sortedBlock[i] = blockSize[i];

    }

    printf("Enter each process size: "); // {212, 417, 112, 426}

    for (int i = 0; i < n; i++)

    {

        scanf("%d", &processSize[i]);

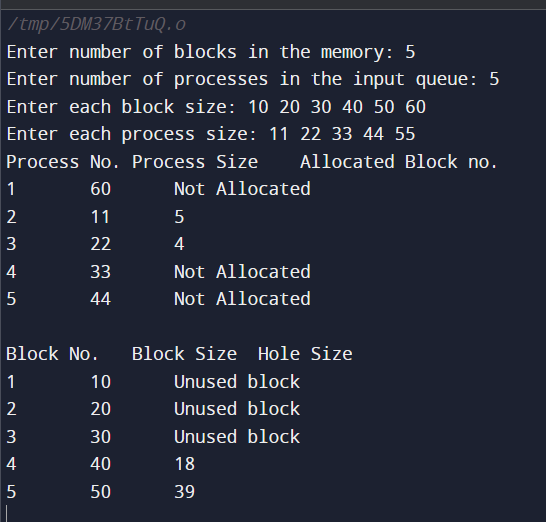
    }

    bubbleSort(sortedBlock, m);

    bestFit(blockSize, m, processSize, n, holeSize, sortedBlock);

}

**Output :**

****

**Conclusion :** Thus we have explored and implemented various Page replacement and memory allocation techniques.