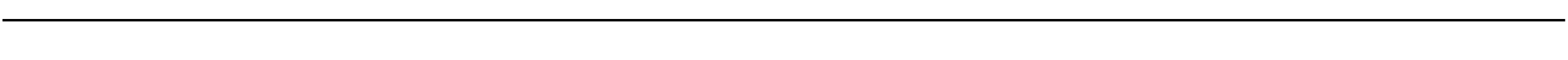
**Experiment 7**

**Title:** Memory Management: Page replacement policies

Write a program to demonstrate the concept of page replacement policies for handling page faults eg: FIFO, LRU, LFU, Optimal etc.

**Estimated time to complete this experiment:** 2 hours



**Objective:** Learning about Page replacement policies. Implementing a program for allocating memory frames to the requesting processes when all the frames are occupied with previous pages. This can be done using various page replacement policies and a conclusion can be derived about the most efficient technique for page replacement.

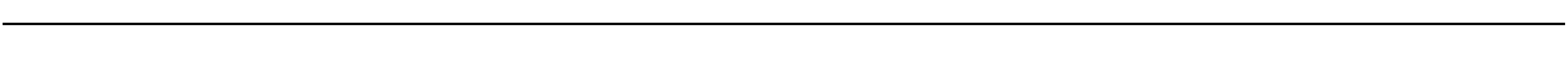


**Expected Outcome of Experiment:** To map pages of required program to memory frames.



**Books/ Journals/ Websites referred:**

1. William Stallings, Operating System: Internals and Design Principles, Prentice Hall, 8thEdition, 2014, ISBN-10: 0133805913 • ISBN-13: 9780133805918.
2. Abraham Silberschatz, Peter Baer Galvin and Greg Gagne, Operating System Concepts, John Wiley &Sons, Inc., 9thEdition, 2016, ISBN 978-81-265-5427-0



**Pre Lab/ Prior Concepts:** Any Programming platform.

**Brief description:**

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when new page comes in.

Page Fault – A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.



**New Concepts to be learned:** Page replacement policies.



**Requirements:** PC with any programming platform.



**Theory:**

1. **First In First Out (FIFO)**

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

1. Advantages:
   1. It is simple and easy to understand & implement.
2. Disadvantages:
   1. The process effectiveness is low.
   2. When we increase the number of frames while using FIFO, we are giving more memory to processes. So, page fault should decrease, but here the page faults are increasing. This problem is called as Belady’s Anomaly.
   3. Every frame needs to be taken account off.
3. **Least Recently Used:**

In this algorithm page will be replaced which is least recently used.

1. Advantages:
   1. It is open for full analysis.
   2. In this, we replace the page which is least recently used, thus free from Belady’s Anomaly.
   3. Easy to choose page which has faulted and hasn’t been used for a long time.
2. Disadvantages:
   1. It requires additional Data Structure to be implemented.
   2. Hardware assistance is high.
3. **Optimal Page replacement:**

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

1. Advantages:
   1. Complexity is less and easy to implement.
   2. Assistance needed is low i.e Data Structure used are easy and light.
2. Disadvantages:
   1. It is perfect, but not possible in practice as the operating system cannot know future requests.
   2. Error handling is tough.
3. **Least Frequently Used (LFU):**

The page to be replaced is the one used least often of the pages currently in the memory.

1. Advantages:
   1. Long sighted
2. Disadvantages:
   1. Cache pollution



**Program:**

#include <iostream>

#include <vector>

using namespace std;

void Take1D(vector<int> &IncomingPages)

{

    int NoOfPages;

    cout << "Enter the number of incoming pages" << endl;

    cin >> NoOfPages;

    IncomingPages.resize(NoOfPages);

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

    {

        cin >> IncomingPages[i];

    }

}

void LRU(vector<int> IncomingPages, vector<int> Frame, int NoOfPages, int NoOfFrames)

{

    int NextFrameIndex = 0;

    Frame.resize(NoOfFrames, -1); *// initialize all frame entries to -1*

    vector<int> LastUsedTime(NoOfFrames, -1); *// initialize all last used times to -1*

    cout << "Incoming Pages: ";

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

    {

        cout << IncomingPages[i] << " ";

*// check if the incoming page is already in the frame*

        bool Found = false;

        int MinLastUsedTime = INT32\_MAX;

        int MinLastUsedTimeIndex = -1;

        for (int j = 0; j < NoOfFrames; j++)

        {

            if (Frame[j] == IncomingPages[i])

            {

                Found = true;

                LastUsedTime[j] = i; *// update the last used time of the page*

                break;

            }

            else if (LastUsedTime[j] < MinLastUsedTime)

            {

                MinLastUsedTime = LastUsedTime[j];

                MinLastUsedTimeIndex = j;

            }

        }

*// if not found, replace the least recently used page in the frame with the incoming page*

        if (!Found)

        {

            Frame[MinLastUsedTimeIndex] = IncomingPages[i];

            LastUsedTime[MinLastUsedTimeIndex] = i; *// update the last used time of the page*

        }

        cout << "Frames: ";

        for (int j = 0; j < NoOfFrames; j++)

        {

            if (Frame[j] == -1)

            {

                cout << "X "; *// indicate an empty frame with 'X'*

            }

            else

            {

                cout << Frame[j] << " ";

            }

        }

        cout << endl;

    }

}

void FIFO(vector<int> IncomingPages, vector<int> Frame, int NoOfPages, int NoOfFrames)

{

    int NextFrameIndex = 0;

    Frame.resize(NoOfFrames, -1);

    cout << "Incoming Pages: ";

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

    {

        cout << IncomingPages[i] << " ";

*// check if the incoming page is already in the frame*

        bool Found = false;

        for (int j = 0; j < NoOfFrames; j++)

        {

            if (Frame[j] == IncomingPages[i])

            {

                Found = true;

                break;

            }

        }

*// if not found, replace the oldest page in the frame with the incoming page*

        if (!Found)

        {

            Frame[NextFrameIndex] = IncomingPages[i];

            NextFrameIndex = (NextFrameIndex + 1) % NoOfFrames; *// update the index of the next frame to replace*

        }

*// print the content of the frames at every instance*

        cout << "Frames: ";

        for (int j = 0; j < NoOfFrames; j++)

        {

            if (Frame[j] == -1)

            {

                cout << "X ";

            }

            else

            {

                cout << Frame[j] << " ";

            }

        }

        cout << endl;

    }

}

int main()

{

    int NoOfPages;

    int NoOfFrames;

    vector<int> IncomingPages;

    vector<int> Frame;

    Take1D(IncomingPages);

    cout << "Enter the number of frames" << endl;

    cin >> NoOfFrames;

    NoOfPages = IncomingPages.size();

    int option;

    cout << "Select a page replacement algorithm:" << endl;

    cout << "1. First In First Out (FIFO)" << endl;

    cout << "2. Least Recently Used (LRU)" << endl;

    cin >> option;

    switch (option) {

        case 1:

            cout << "Using FIFO page replacement algorithm" << endl;

            FIFO(IncomingPages, Frame, NoOfPages, NoOfFrames);

            break;

        case 2:

            cout << "Using LRU page replacement algorithm" << endl;

            LRU(IncomingPages, Frame, NoOfPages, NoOfFrames);

            break;

        default:

            cout << "Invalid option selected. Please try again." << endl;

    }

    return 0;

}

**Output:**

**Text

Description automatically generated**

**Text

Description automatically generated**



**Conclusion:** Hence we have understoodhow memory is allocated to processes when all the page frames in main memory are occupied.