COURSE CODE: DJS22ITL305 Sem: III

COURSE NAME: Operating System Laboratory CLASS:SY. BTech I1-1

# Name :Anish A. Shetty Sap id : 60003220045

**Experiment 6**

**CO/LO:** Apply appropriate process scheduling, memory mapping and disk scheduling methods.

**Aim**: Implementation of Page replacement algorithms (FIFO, LRU)

**FIFO :**

#include <stdio.h>

int main() {

int frames, pages, pageFaults = 0;

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

scanf("%d", &frames);

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

scanf("%d", &pages);

int incomingStream[pages];

printf("Enter the page reference sequence:\n");

for (int i = 0; i < pages; ++i) {

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

}

int temp[frames];

for (int m = 0; m < frames; m++) {

temp[m] = -1;

}

printf("Page Reference Sequence: ");

for (int i = 0; i < pages; ++i) {

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

}

printf("\n");

int m, n, s;

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

s = 0;

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

if (incomingStream[m] == temp[n]) {

s++;

pageFaults--;

}

}

pageFaults++;

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

temp[m] = incomingStream[m];

} else if (s == 0) {

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

}

printf("Frames at stage [%d]: ", m + 1);

for (int i = 0; i < frames; ++i) {

if (temp[i] == -1) {

printf("- ");

} else {

printf("%d ", temp[i]);

}

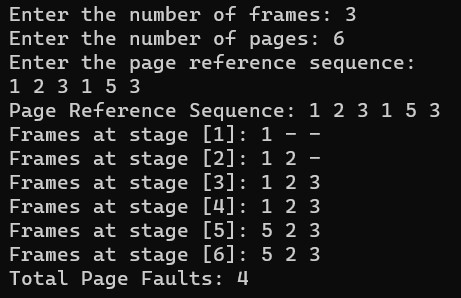
}

printf("\n");

}

printf("Total Page Faults: %d\n", pageFaults);

return 0;

}

**LRU :**

#include <stdio.h>

#include <stdbool.h>

int main() {

int capacity, pages, pageFaults = 0;

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

scanf("%d", &capacity);

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

scanf("%d", &pages);

int incomingStream[pages];

int indexes[capacity];

int set[capacity];

bool pagePresent[capacity];

for (int i = 0; i < capacity; i++) {

indexes[i] = -1;

pagePresent[i] = false;

}

printf("Enter the page reference sequence:\n");

for (int i = 0; i < pages; i++) {

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

}

printf("Page Reference Sequence: ");

for (int i = 0; i < pages; i++) {

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

}

printf("\n");

for (int i = 0; i < pages; i++) {

if (pagePresent[incomingStream[i]]) {

// Page is already in memory, do nothing

} else {

if (pageFaults < capacity) {

int emptySlot = -1;

for (int j = 0; j < capacity; j++) {

if (!pagePresent[j]) {

emptySlot = j;

break;

}

}

if (emptySlot != -1) {

set[emptySlot] = incomingStream[i];

pagePresent[emptySlot] = true;

indexes[incomingStream[i]] = i;

}

} else {

int minIndex = pages + 1;

int victimPage;

for (int j = 0; j < capacity; j++) {

if (indexes[set[j]] < minIndex) {

minIndex = indexes[set[j]];

victimPage = j;

}

}

pagePresent[victimPage] = false;

set[victimPage] = incomingStream[i];

pagePresent[victimPage] = true;

indexes[incomingStream[i]] = i;

pageFaults++;

}

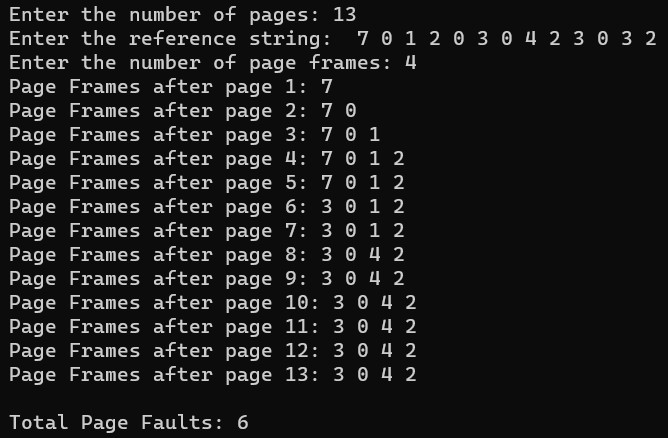
}

}

printf("Total Page Faults: %d\n", pageFaults);

return 0;

}



**BOOKS AND WEB RESOURCES:**

* "Operating System Concepts" by Abraham Silberschatz, Peter B. Galvin, and Greg Gagne

* "Operating Systems: Three Easy Pieces" by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau

* GeeksforGeeks

* Tutorialspoint

**Conclusion:**

In this study, we implemented and compared FIFO and LRU page replacement algorithms in C language. Both algorithms were tested with the same reference string "1 2 3 4 1 2 5 1 2". The FIFO algorithm, replacing the oldest page, and the LRU algorithm, replacing the least recently used page, both resulted in 9 page faults. These findings highlight the importance of considering specific application requirements and access patterns when choosing the appropriate page replacement strategy.