**Lab 05:Page Replacement**

**1.Simulate contigous memory allocation technique for**

**1.Worst fit 2.Best fit 3.First fit**

**Worst -fit**

#include <stdio.h>

void implimentWorstFit(int blockSize[], int blocks, int processSize[], int processes)

{

int allocation[processes];

int occupied[blocks];

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

allocation[i] = -1;

}

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

occupied[i] = 0;

}

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

{

int indexPlaced = -1;

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

{

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

{

if (indexPlaced == -1)

indexPlaced = j;

else if (blockSize[indexPlaced] < blockSize[j])

indexPlaced = j;

}

}

if (indexPlaced != -1)

{

allocation[i] = indexPlaced;

occupied[indexPlaced] = 1;

blockSize[indexPlaced] -= processSize[i];

}

}

printf("\nProcess No. \t\tProcess Size \t\tBlock no.\n");

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

{

printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);

if (allocation[i] != -1)

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

else

printf("Not Allocated\n");

}

}

int main()

{

int blockSize[] = {100, 50, 30, 120, 35};

int processSize[] = {40, 10, 30, 60};

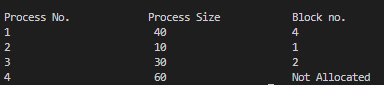
int blocks = sizeof(blockSize)/sizeof(blockSize[0]);

int processes = sizeof(processSize)/sizeof(processSize[0]);

implimentWorstFit(blockSize, blocks, processSize, processes);

return 0;

}

**OUTPUT:**

**Best-fit**

#include <stdio.h>

void implimentBestFit(int blockSize[], int blocks, int processSize[], int processes)

{

int allocation[processes];

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

allocation[i] = -1;

}

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

{

int indexPlaced = -1;

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

{

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

{

if (indexPlaced == -1)

indexPlaced = j;

else if (blockSize[j] < blockSize[indexPlaced])

indexPlaced = j;

}

}

if (indexPlaced != -1)

{

allocation[i] = indexPlaced;

blockSize[indexPlaced] -= processSize[i];

}

}

printf("\nProcess No. \t\tProcess Size \t\tBlock no.\n");

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

{

printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);

if (allocation[i] != -1)

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

else

printf("Not Allocated\n");

}

}

int main()

{

int blockSize[] = {50, 20, 100, 90};

int processSize[] = {10, 30, 60, 110};

int blocks = sizeof(blockSize)/sizeof(blockSize[0]);

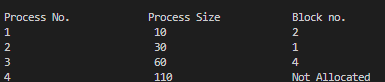
int processes = sizeof(processSize)/sizeof(processSize[0]);

implimentBestFit(blockSize, blocks, processSize, processes);

return 0 ;

}

**OUTPUT:**

****

**First-fit**

#include <stdio.h>

void implimentFirstFit(int blockSize[], int blocks, int processSize[], int processes)

{

int allocate[processes];

int occupied[blocks];

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

{

allocate[i] = -1;

}

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

occupied[i] = 0;

}

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

{

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

{

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

{

allocate[i] = j;

occupied[j] = 1;

break;

}

}

}

printf("\nProcess No.\tProcess Size \t\tBlock no.\n");

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

{

printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);

if (allocate[i] != -1)

printf("%d\n",allocate[i] + 1);

else

printf("Not Allocated\n");

}

}

void main()

{

int blockSize[] = {10, 20, 30,40};

int processSize[] = {15, 6,30};

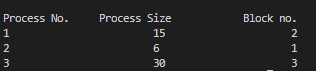
int m = sizeof(blockSize)/sizeof(blockSize[0]);

int n = sizeof(processSize)/sizeof(processSize[0]);

implimentFirstFit(blockSize, m, processSize, n);

}

**OUTPUT:**



**2.Simulate First-In First-out(FIFO)**

#include <stdio.h>

int main() {

int incomingStream[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1};

int pageFaults = 0;

int pageHits = 0;

int frames;

int m, n, s, pages;

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

scanf("%d", &frames);

printf("Incoming\t");

for (m = 1; m <= frames; m++) {

printf("Frame %d\t", m);

}

printf("\n");

pages = sizeof(incomingStream) / sizeof(incomingStream[0]);

int temp[frames];

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

temp[m] = -1;

}

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

s = 0;

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

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

s = 1;

pageHits++;

break;

}

}

if (s == 0) {

pageFaults++;

if (pageFaults <= frames) {

temp[pageFaults - 1] = incomingStream[m];

} else {

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

}

}

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

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

if (temp[n] != -1)

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

else

printf(" - \t");

}

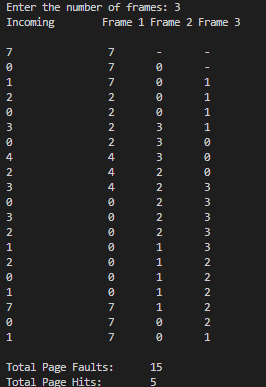
}

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

printf("Total Page Hits:\t%d\n", pageHits);

return 0;

}

**OUTPUT:**

**3.Simulate Optimal Page Replacement Algorithm(OPR)**

#include <stdio.h>

int isPageInFrames(int page, int frames[], int numFrames) {

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

if (frames[i] == page) {

return 1;

}

}

return 0;

}

void printFrameHeaders(int numFrames) {

printf("Stream\t");

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

printf("Frame%d\t", i + 1);

}

printf("\n");

}

void printFrames(int page, int frames[], int numFrames, int occupiedFrames) {

printf("%d\t", page);

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

if (i < occupiedFrames) {

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

} else {

printf("-\t");

}

}

printf("\n");

}

int findFrameToReplace(int refStr[], int frames[], int refStrLen, int currentIndex, int numFrames) {

int farthest = currentIndex;

int frameToReplace = 0;

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

int j;

for (j = currentIndex; j < refStrLen; j++) {

if (frames[i] == refStr[j]) {

if (j > farthest) {

farthest = j;

frameToReplace = i;

}

break;

}

}

if (j == refStrLen) {

return i;

}

}

return frameToReplace;

}

void optimalPageReplacement(int refStr[], int refStrLen, int numFrames) {

int frames[numFrames];

int occupiedFrames = 0;

int pageHits = 0;

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

frames[i] = -1;

}

printFrameHeaders(numFrames);

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

int page = refStr[i];

if (isPageInFrames(page, frames, occupiedFrames)) {

pageHits++;

printFrames(page, frames, numFrames, occupiedFrames);

} else {

if (occupiedFrames < numFrames) {

frames[occupiedFrames] = page;

occupiedFrames++;

} else {

int frameToReplace = findFrameToReplace(refStr, frames, refStrLen, i + 1, numFrames);

frames[frameToReplace] = page;

}

printFrames(page, frames, numFrames, occupiedFrames);

}

}

printf("\nPage Hits: %d\n", pageHits);

printf("Page Faults: %d\n", refStrLen - pageHits);

}

int main() {

int refStr[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1};

int refStrLen = sizeof(refStr) / sizeof(refStr[0]);

int numFrames;

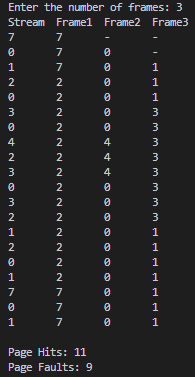
printf("Enter the number of frames: ");

scanf("%d", &numFrames);

optimalPageReplacement(refStr, refStrLen, numFrames);

return 0;

}

**OUTPUT:**

**4.Simulate Least Recently Used (LRU)Algorithm.**

#include <stdio.h>

#include <limits.h>

int checkHit(int incomingPage, int queue[], int occupied) {

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

if (incomingPage == queue[i])

return 1;

}

return 0;

}

void printFrame(int queue[], int frames) {

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

if (i < frames)

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

else

printf("-\t\t");

}

}

int main() {

int incomingStream[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1};

int n = sizeof(incomingStream) / sizeof(incomingStream[0]);

int frames;

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

scanf("%d", &frames);

int queue[frames];

int distance[frames];

int occupied = 0;

int pagefault = 0;

int pagehit = 0;

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

queue[i] = -1;

}

printf("Stream\t\t");

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

printf("Frame%d\t\t", i + 1);

printf("\n");

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

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

if (checkHit(incomingStream[i], queue, occupied)) {

printFrame(queue, frames);

pagehit++;

} else if (occupied < frames) {

queue[occupied] = incomingStream[i];

pagefault++;

occupied++;

printFrame(queue, occupied);

} else {

int max = INT\_MIN;

int index;

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

distance[j] = 0;

for (int k = i - 1; k >= 0; k--) {

++distance[j];

if (queue[j] == incomingStream[k])

break;

}

if (distance[j] > max) {

max = distance[j];

index = j;

}

}

queue[index] = incomingStream[i];

printFrame(queue, frames);

pagefault++;

}

printf("\n");

}

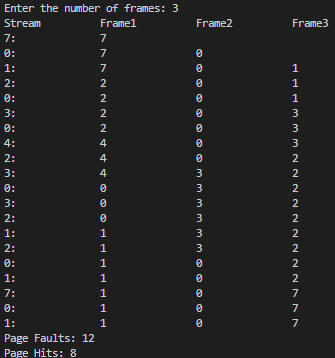
printf("Page Faults: %d\n", pagefault);

printf("Page Hits: %d\n", pagehit);

return 0;

}

**OUTPUT:**

****