Lab 06:

**6.Write a C program to simulate the following contiguous memory allocation techniques**

**a)Worst-fit**

**b) Best-fit**

**c) First-fit**

#include <stdio.h>

struct Block {

int size;

int allocated;

};

struct File {

int size;

int block\_no;

};

void resetBlocks(struct Block blocks[], int n) {

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

blocks[i].allocated = 0;

}

}

void firstFit(struct Block blocks[], int n\_blocks, struct File files[], int n\_files) {

printf("\n\tMemory Management Scheme – First Fit\n");

printf("File\_no:\tFile\_size\tBlock\_no:\tBlock\_size:\n");

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

files[i].block\_no = -1;

for (int j = 0; j < n\_blocks; j++) {

if (!blocks[j].allocated && blocks[j].size >= files[i].size) {

files[i].block\_no = j + 1;

blocks[j].allocated = 1;

printf("%d\t\t%d\t\t%d\t\t%d\n", i + 1, files[i].size, j + 1, blocks[j].size);

break;

}

}

if (files[i].block\_no == -1) {

printf("%d\t\t%d\t\t\_\t\t\_\n", i + 1, files[i].size);

}

}

}

void bestFit(struct Block blocks[], int n\_blocks, struct File files[], int n\_files) {

printf("\n\tMemory Management Scheme – Best Fit\n");

printf("File\_no:\tFile\_size\tBlock\_no:\tBlock\_size:\n");

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

int bestIdx = -1;

for (int j = 0; j < n\_blocks; j++) {

if (!blocks[j].allocated && blocks[j].size >= files[i].size) {

if (bestIdx == -1 || blocks[j].size < blocks[bestIdx].size) {

bestIdx = j;

}

}

}

if (bestIdx != -1) {

blocks[bestIdx].allocated = 1;

files[i].block\_no = bestIdx + 1;

printf("%d\t\t%d\t\t%d\t\t%d\n", i + 1, files[i].size, bestIdx + 1, blocks[bestIdx].size);

} else {

printf("%d\t\t%d\t\t\_\t\t\_\n", i + 1, files[i].size);

}

}

}

void worstFit(struct Block blocks[], int n\_blocks, struct File files[], int n\_files) {

printf("\n\tMemory Management Scheme – Worst Fit\n");

printf("File\_no:\tFile\_size\tBlock\_no:\tBlock\_size:\n");

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

int worstIdx = -1;

for (int j = 0; j < n\_blocks; j++) {

if (!blocks[j].allocated && blocks[j].size >= files[i].size) {

if (worstIdx == -1 || blocks[j].size > blocks[worstIdx].size) {

worstIdx = j;

}

}

}

if (worstIdx != -1) {

blocks[worstIdx].allocated = 1;

files[i].block\_no = worstIdx + 1;

printf("%d\t\t%d\t\t%d\t\t%d\n", i + 1, files[i].size, worstIdx + 1, blocks[worstIdx].size);

} else {

printf("%d\t\t%d\t\t\_\t\t\_\n", i + 1, files[i].size);

}

}

}

int main() {

int n\_blocks, n\_files, choice;

printf("Memory Management Scheme\n");

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

scanf("%d", &n\_blocks);

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

scanf("%d", &n\_files);

struct Block blocks[n\_blocks];

struct File files[n\_files];

printf("\nEnter the size of the blocks:\n");

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

printf("Block %d: ", i + 1);

scanf("%d", &blocks[i].size);

blocks[i].allocated = 0;

}

printf("Enter the size of the files:\n");

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

printf("File %d: ", i + 1);

scanf("%d", &files[i].size);

}

do {

printf("\n1. First Fit\n2. Best Fit\n3. Worst Fit\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

resetBlocks(blocks, n\_blocks); // Reset block allocation before each strategy

switch (choice) {

case 1:

firstFit(blocks, n\_blocks, files, n\_files);

break;

case 2:

bestFit(blocks, n\_blocks, files, n\_files);

break;

case 3:

worstFit(blocks, n\_blocks, files, n\_files);

break;

case 4:

printf("\nExiting...\n");

break;

default:

printf("Invalid choice.\n");

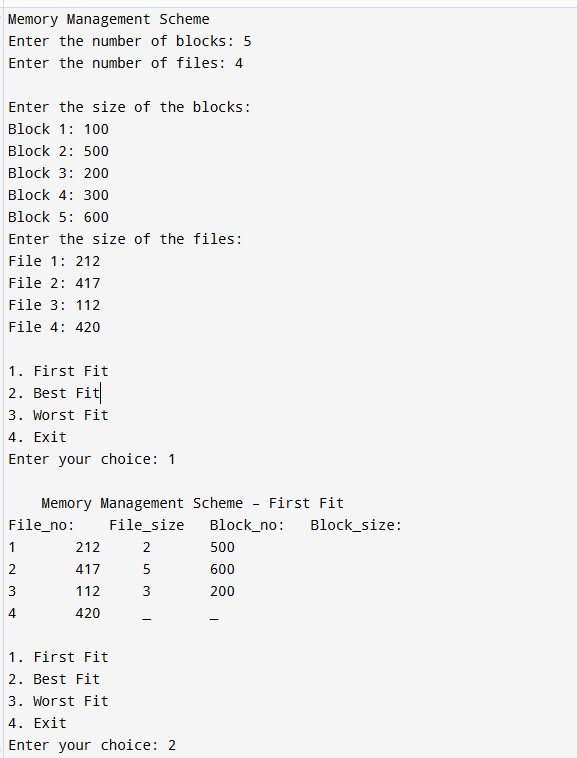
}

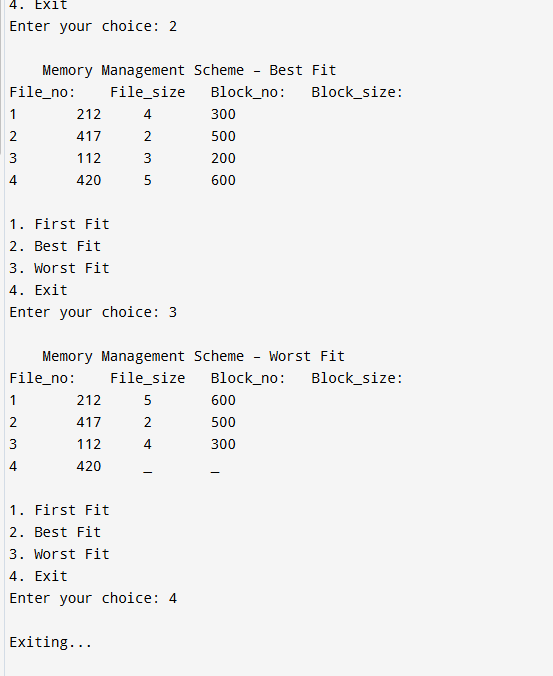
} while (choice != 4);

return 0;

}

Output:





**Lab program no 7**

**Write a C program to simulate page replacement algorithms**

1. **FIFO**

#include <stdio.h>

int main() {

int frames, pages[50], n, frame[10], i, j, k, avail, count = 0;

printf("Enter number of pages: ");

scanf("%d", &n);

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

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

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

printf("Enter number of frames: ");

scanf("%d", &frames);

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

frame[i] = -1;

printf("\nPage\tFrames\t\tPage Fault\n");

j = 0;

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

avail = 0;

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

if(frame[k] == pages[i]) {

avail = 1;

break;

}

}

if(avail == 0) {

frame[j] = pages[i];

j = (j + 1) % frames;

count++;

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

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

if(frame[k] != -1)

printf("%d ", frame[k]);

else

printf("- ");

}

printf("\tYes\n");

} else {

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

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

if(frame[k] != -1)

printf("%d ", frame[k]);

else

printf("- ");

}

printf("\tNo\n");

}

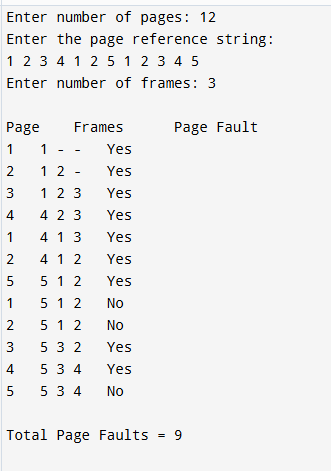
}

printf("\nTotal Page Faults = %d\n", count);

return 0;

}

Output:



b)recently used

#include <stdio.h>

int main() {

int n, frames, i, j, k, faults = 0;

printf("Enter number of pages: ");

scanf("%d", &n);

int pages[n];

printf("Enter the reference string: ");

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

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

printf("Enter number of frames: ");

scanf("%d", &frames);

int frame\_arr[frames];

int time[frames];

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

frame\_arr[i] = -1;

time[i] = 0;

}

int counter = 0;

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

int flag = 0;

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

if(frame\_arr[j] == pages[i]) {

flag = 1;

counter++;

time[j] = counter;

break;

}

}

if(flag == 0) {

faults++;

int min\_time = time[0], min\_pos = 0;

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

if(time[k] < min\_time) {

min\_time = time[k];

min\_pos = k;

}

}

frame\_arr[min\_pos] = pages[i];

counter++;

time[min\_pos] = counter;

}

printf("Frames after accessing %d: ", pages[i]);

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

if(frame\_arr[j] == -1)

printf("- ");

else

printf("%d ", frame\_arr[j]);

}

printf("\n");

}

printf("Total page faults: %d\n", faults);

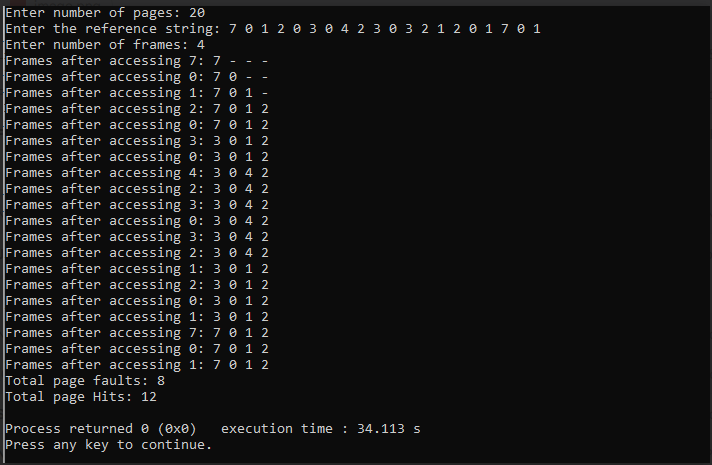
int Hits = n-faults;

printf("Total page Hits: %d\n",Hits);

return 0;

}

Output:



c)optimal

#include <stdio.h>

int main() {

int n, frames, i, j, k, faults = 0;

printf("Enter number of pages: ");

scanf("%d", &n);

int pages[n];

printf("Enter the reference string: ");

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

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

printf("Enter number of frames: ");

scanf("%d", &frames);

int frame\_arr[frames];

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

frame\_arr[i] = -1;

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

int flag = 0;

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

if(frame\_arr[j] == pages[i]) {

flag = 1;

break;

}

}

if(flag == 0) {

faults++;

int pos = -1;

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

if(frame\_arr[j] == -1) {

pos = j;

break;

}

}

if(pos == -1) {

int farthest = i, replace\_index = 0;

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

int found = 0;

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

if(frame\_arr[j] == pages[k]) {

if(k > farthest) {

farthest = k;

replace\_index = j;

}

found = 1;

break;

}

}

if(!found) {

replace\_index = j;

break;

}

}

pos = replace\_index;

}

frame\_arr[pos] = pages[i];

}

printf("%d: ", pages[i]);

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

if(frame\_arr[j] == -1)

printf("\_ ");

else

printf("%d ", frame\_arr[j]);

}

printf("\n");

}

printf("Total page faults: %d\n", faults);

int Hits = n-faults;

printf("Total page Hits: %d\n",Hits);

return 0;

}

Output:

