31.Construct a C program to simulate the First in First Out paging technique of

memory management.

Program:

#include <stdio.h>

#include <stdlib.h>

#define FRAME\_SIZE 3 // Number of page frames

void fifo(int pages[], int n) {

int frame[FRAME\_SIZE], pageFaults = 0, index = 0;

int isPageInFrame;

// Initialize frames

for (int i = 0; i < FRAME\_SIZE; i++) {

frame[i] = -1; // -1 indicates an empty frame

}

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

isPageInFrame = 0;

// Check if the page is already in one of the frames

for (int j = 0; j < FRAME\_SIZE; j++) {

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

isPageInFrame = 1;

break;

}

}

// If the page is not in the frame, we have a page fault

if (!isPageInFrame) {

frame[index] = pages[i]; // Replace the oldest page

index = (index + 1) % FRAME\_SIZE; // Move to the next frame

pageFaults++;

}

// Display current frame status

printf("Current frame: ");

for (int j = 0; j < FRAME\_SIZE; j++) {

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

}

printf("\n");

}

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

}

int main() {

int pages[] = {7, 0, 1, 2, 0, 3, 0, 4}; // Example page reference string

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

fifo(pages, n);

return 0;

}

Output

Current frame:7-1-1

Current frame: 7 0 -1

Current frame: 7 0 1

Current frame:2 0 1

Current frame: 2 0 1

Current frame:2 3 1

Current frame:2 3 0

Current frame: 4 3 0

Total Page Faults: 7

32. Construct a C program to simulate the Least Recently Used paging technique of

memory management

program:

#include <stdio.h>

#include <stdlib.h>

#define FRAME\_SIZE 3 // Number of frames in memory

void simulateLRU(int pages[], int n) {

int frames[FRAME\_SIZE], time[FRAME\_SIZE], i, j, k, lru, pageFaults = 0;

for (i = 0; i < FRAME\_SIZE; i++) {

frames[i] = -1; // Initialize frames

}

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

int page = pages[i];

int found = 0;

// Check if page is already in frames

for (j = 0; j < FRAME\_SIZE; j++) {

if (frames[j] == page) {

found = 1;

time[j] = i; // Update the time of usage

break;

}

}

// If page not found, we have a page fault

if (!found) {

lru = 0;

for (j = 1; j < FRAME\_SIZE; j++) {

if (time[j] < time[lru]) {

lru = j; // Find the least recently used page

}

}

frames[lru] = page; // Replace it with the new page

time[lru] = i; // Update the time of usage

pageFaults++;

}

}

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

}

int main() {

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

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

simulateLRU(pages, n);

return 0;

}

Output

Total Page Faults: 8

33. Construct a C program to simulate the optimal paging technique of memory

Management

Program:

#include <stdio.h>

#define FRAME\_SIZE 3

#define PAGE\_REFERENCES 10

void optimalPageReplacement(int pages[], int n) {

int frame[FRAME\_SIZE], pageFaults = 0;

for (int i = 0; i < FRAME\_SIZE; i++) {

frame[i] = -1; // Initialize frames

}

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

int j, flag = 0;

for (j = 0; j < FRAME\_SIZE; j++) {

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

flag = 1; // Page hit

break;

}

}

if (!flag) { // Page fault

int lruIndex = -1, farthest = -1;

for (j = 0; j < FRAME\_SIZE; j++) {

int k;

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

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

if (k > farthest) {

farthest = k;

lruIndex = j;

}

break;

}

}

if (k == n) { // If page is not found in future

lruIndex = j;

break;

}

}

frame[lruIndex] = pages[i]; // Replace the page

pageFaults++;

}

}

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

}

int main() {

int pages[PAGE\_REFERENCES] = {0, 1, 2, 0, 3, 0, 4, 2, 3, 0};

optimalPageReplacement(pages, PAGE\_REFERENCES);

return 0;

}

Output

Total Page Faults: 6

34. Consider a file system where the records of the file are stored one after another

both physically and logically. A record of the file can only be accessed by

reading all the previous records. Design a C program to simulate the file

allocation strategy.

Program:

#include <stdio.h>

#include <string.h>

#define MAX\_RECORDS 100

#define RECORD\_SIZE 50

typedef struct {

char data[RECORD\_SIZE];

} Record;

void readRecords(Record records[], int count) {

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

printf("Record %d: %s\n", i + 1, records[i].data);

}

}

int main() {

Record records[MAX\_RECORDS];

int count;

printf("Enter the number of records (max %d): ", MAX\_RECORDS);

scanf("%d", &count);

getchar(); // Consume newline character

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

printf("Enter data for record %d: ", i + 1);

fgets(records[i].data, RECORD\_SIZE, stdin);

records[i].data[strcspn(records[i].data, "\n")] = 0; // Remove newline

}

printf("\nReading all records:\n");

readRecords(records, count);

return 0;

}

Output

Enter the number of records (max 100): 1

Enter data for record 1: 1

Reading all records:

Record 1: 1

35. Consider a file system that brings all the file pointers together into an index

block. The ith entry in the index block points to the ith block of the file. Design

a C program to simulate the file allocation strategy

program:

#include <stdio.h>

#include <stdlib.h>

#define BLOCK\_SIZE 1024 // Size of each block

#define NUM\_BLOCKS 5 // Number of blocks

typedef struct {

int blockPointers[NUM\_BLOCKS]; // Index block

} IndexBlock;

void allocateFileBlocks(IndexBlock \*index) {

for (int i = 0; i < NUM\_BLOCKS; i++) {

index->blockPointers[i] = i; // Pointing to each block

}

}

void displayFileAllocation(IndexBlock \*index) {

printf("File Allocation:\n");

for (int i = 0; i < NUM\_BLOCKS; i++) {

printf("Index %d -> Block %d\n", i, index->blockPointers[i]);

}

}

int main() {

IndexBlock \*index = (IndexBlock \*)malloc(sizeof(IndexBlock));

allocateFileBlocks(index);

displayFileAllocation(index);

free(index);

return 0;

}

Output

File Allocation:

Index 0 -> Block 0

Index 1 -> Block 1

Index 2 -> Block 2

Index 3 -> Block 3

Index 4 -> Block 4

36. With linked allocation, each file is a linked list of disk blocks; the disk blocks

may be scattered anywhere on the disk. The directory contains a pointer to the

first and last blocks of the file. Each block contains a pointer to the next block.

Design a C program to simulate the file allocation strategy.

Program:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct Block {

char data[256]; // Data stored in the block

struct Block\* next; // Pointer to the next block

} Block;

typedef struct File {

Block\* first; // Pointer to the first block

Block\* last; // Pointer to the last block

} File;

// Function to create a new block

Block\* createBlock(const char\* data) {

Block\* newBlock = (Block\*)malloc(sizeof(Block));

strcpy(newBlock->data, data);

newBlock->next = NULL;

return newBlock;

}

// Function to add a block to the file

void addBlock(File\* file, const char\* data) {

Block\* newBlock = createBlock(data);

if (file->last) {

file->last->next = newBlock;

} else {

file->first = newBlock; // First block

}

file->last = newBlock; // Update last block

}

// Function to display the file contents

void displayFile(File\* file) {

Block\* current = file->first;

while (current) {

printf("%s -> ", current->data);

current = current->next;

}

printf("NULL\n");

}

int main() {

File myFile = {NULL, NULL}; // Initialize file

addBlock(&myFile, "Block 1");

addBlock(&myFile, "Block 2");

addBlock(&myFile, "Block 3");

displayFile(&myFile); // Display the linked list of blocks

return 0;

}  
output

Block 1 -> Block 2 -> Block 3 -> NULL

37. Construct a C program to simulate the First Come First Served disk scheduling

algorithm.

Program:

#include <stdio.h>

void findWaitingTime(int processes[], int n, int bt[], int wt[]) {

wt[0] = 0; // Waiting time for the first process is 0

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

wt[i] = bt[i - 1] + wt[i - 1]; // Calculate waiting time

}

}

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {

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

tat[i] = bt[i] + wt[i]; // Turnaround time = Burst time + Waiting time

}

}

void findavgTime(int processes[], int n, int bt[]) {

int wt[n], tat[n];

findWaitingTime(processes, n, bt, wt);

findTurnAroundTime(processes, n, bt, wt, tat);

printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

printf("%d\t%d\t\t%d\t\t%d\n", processes[i], bt[i], wt[i], tat[i]);

}

}

int main() {

int processes[] = {1, 2, 3}; // Process IDs

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

int burst\_time[] = {10, 5, 8}; // Burst times

findavgTime(processes, n, burst\_time);

return 0;

}

Output

Process Burst Time Waiting Time Turnaround Time

1 10 0 10

2 5 10 15

3 8 15 23

38. Design a C program to simulate SCAN disk scheduling algorithm.

Program:

#include <stdio.h>

#include <stdlib.h>

void scan(int arr[], int size, int head, int direction) {

int seek\_sequence[size + 1];

int index = 0;

// Sort the request array

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

for (int j = i + 1; j < size; j++) {

if (arr[i] > arr[j]) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

// Find the position of the head

int pos;

for (pos = 0; pos < size; pos++) {

if (arr[pos] >= head) {

break;

}

}

// Move towards the end of the disk

for (int i = pos; i < size; i++) {

seek\_sequence[index++] = arr[i];

}

// Move to the end of the disk

if (direction == 1) {

seek\_sequence[index++] = 199; // Assuming 199 is the end of the disk

}

// Move back towards the beginning of the disk

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

seek\_sequence[index++] = arr[i];

}

// Print the seek sequence

printf("Seek Sequence: ");

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

printf("%d ", seek\_sequence[i]);

}

printf("\n");

}

int main() {

int requests[] = { 98, 183, 37, 122, 14, 124, 65, 67 };

int head = 53;

int direction = 1; // 1 for right, 0 for left

int size = sizeof(requests) / sizeof(requests[0]);

scan(requests, size, head, direction);

return 0;

}

Output

Seek Sequence: 65 67 98 122 124 183 199 37 14

39. Develop a C program to simulate C-SCAN disk scheduling algorithm.

Program:

#include <stdio.h>

void cscan(int arr[], int size, int head, int direction) {

int seek\_sequence[size + 1];

int distance = 0, cur\_track;

int index = 0;

// Sort the request array

for (int i = 0; i < size - 1; i++) {

for (int j = 0; j < size - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

// Run the C-SCAN algorithm

if (direction == 1) { // Moving towards the end

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

if (arr[i] >= head) {

seek\_sequence[index++] = head;

distance += arr[i] - head;

head = arr[i];

}

}

seek\_sequence[index++] = head;

head = 0; // Jump to the beginning

distance += head; // Add jump distance

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

if (arr[i] < head) {

seek\_sequence[index++] = head;

distance += arr[i] - head;

head = arr[i];

}

}

} else { // Moving towards the beginning

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

if (arr[i] <= head) {

seek\_sequence[index++] = head;

distance += head - arr[i];

head = arr[i];

}

}

seek\_sequence[index++] = head;

head = 199; // Jump to the end

distance += head; // Add jump distance

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

if (arr[i] > head) {

seek\_sequence[index++] = head;

distance += head - arr[i];

head = arr[i];

}

}

}

// Print the seek sequence

printf("Seek Sequence: ");

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

printf("%d ", seek\_sequence[i]);

}

printf("\nTotal Seek Distance: %d\n", distance);

}

int main() {

int arr[] = { 98, 183, 37, 122, 14, 124, 65, 67 };

int size = sizeof(arr) / sizeof(arr[0]);

int head = 53; // Initial head position

int direction = 1; // 1 for right, 0 for left

cscan(arr, size, head, direction);

return 0;

}

Output

Seek Sequence: 53 65 67 98 122 124 183

Total Seek Distance: 130

40. Illustrate the various File Access Permission and different types of users in

Linux.

Program:

#include <stdio.h>

#include <stdlib.h>

#include <sys/stat.h>

int main() {

char filename[] = "file.txt";

int new\_permissions = S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IWGRP |S\_IROTH;

if (chmod(filename, new\_permissions) == 0) { printf("File

permissions changed successfully.\n");

} else {

perror("chmod");

return 1;}

return 0;

}

**Output**

chmod: No such file or directory