OS LAB PREPARATION

1. question

index file allocation program

```
#include <stdio.h>
#include <stdlib.h>

void main() {
    int f[50], index[50], i, n, st, len, j, c, k, ind, count = 0;

    for (i = 0; i < 50; i++)
        f[i] = 0;

x:
    printf("Enter the index block: ");
    scanf("%d", &ind);

    if (f[ind] != 1) {
        printf("Enter number of blocks needed and number of files for the
index %d on the disk:\n", ind);
        scanf("%d", &n);
    } else {
        printf("%d index is already allocated.\n", ind);
        goto x;
    }

y:
    count = 0;

    for (i = 0; i < n; i++) {
        scanf("%d", &index[i]);
        if (f[index[i]] == 0)
            count++;
    }

    if (count == n) {
        for (j = 0; j < n; j++)
            f[index[j]] = 1;
```

```
        printf("Allocated\n");
        printf("File Indexed\n");

        for (k = 0; k < n; k++)
            printf("%d----------->%d : %d\n", ind, index[k], f[index[k]]);
    } else {
        printf("File in the index is already allocated.\n");
        printf("Enter another file indexed.\n");
        goto y;
    }

    printf("Do you want to enter more files? (Yes - 1/No - 0)");
    scanf("%d", &c);

    if (c == 1)
        goto x;
    else
        exit(0);
}
```

bash program

```
//for loop
@echo off
REM Iterating 5 times
for /l %%i in (1, 1, 5) do (
    echo Iteration: %%i
)

// while loop

@echo off
set count=1
:loop
if %count% leq 5 (
    echo Count: %count%
    set /a count+=1
    goto :loop
)


// untill loop
```

```
@echo off
set count=1
:loop
if %count% gtr 5 (
    goto :endloop
)
echo Count: %count%
set /a count+=1
goto :loop
:endloop
```

2. question

LRU in C

```
#include <stdio.h>
#include <stdlib.h>

#define NUM_FRAMES 3 // Number of frames in memory

// Structure to represent a page
typedef struct {
    int pageNumber;
    int timeStamp; // Timestamp to track when the page was last accessed
} Page;

// Function to find the least recently used page
int findLRUPage(Page pages[], int numFrames) {
    int minTimeStamp = pages[0].timeStamp;
    int lruPageIndex = 0;

    for (int i = 1; i < numFrames; i++) {
        if (pages[i].timeStamp < minTimeStamp) {
            minTimeStamp = pages[i].timeStamp;
            lruPageIndex = i;
        }
    }

    return lruPageIndex;
}

// Function to simulate page replacement using LRU algorithm
```

```
void simulateLRU(int pageRequests[], int numRequests, int numFrames) {
    Page frames[numFrames];

    // Initialize frames
    for (int i = 0; i < numFrames; i++) {
        frames[i].pageNumber = -1; // -1 indicates an empty frame
        frames[i].timeStamp = 0;
    }

    int pageFaults = 0;

    // Simulate page requests
    for (int i = 0; i < numRequests; i++) {
        int pageRequested = pageRequests[i];
        int pageFound = 0;

        // Check if page is already in memory
        for (int j = 0; j < numFrames; j++) {
            if (frames[j].pageNumber == pageRequested) {
                frames[j].timeStamp = i; // Update timestamp
                pageFound = 1;
                break;
            }
        }

        // If page not found in memory, replace the least recently used page
        if (!pageFound) {
            int lruIndex = findLRUPage(frames, numFrames);
            frames[lruIndex].pageNumber = pageRequested;
            frames[lruIndex].timeStamp = i; // Update timestamp
            pageFaults++;
        }

        // Print current state of memory after each request
        printf("Memory: ");
        for (int j = 0; j < numFrames; j++) {
            printf("%d ", frames[j].pageNumber);
        }
        printf("\n");
    }

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

```
int main() {
    int pageRequests[] = {1, 3, 0, 3, 5, 6, 3}; // Sample page requests
    int numRequests = sizeof(pageRequests) / sizeof(pageRequests[0]);
    int numFrames = NUM_FRAMES;

    simulateLRU(pageRequests, numRequests, numFrames);

    return 0;
}
```

3.question

Best-Fit Allocation

```
#include <stdio.h>

#define MAX_BLOCKS 100

// Structure to represent memory blocks
typedef struct {
    int id;
    int size;
    int allocated;
} MemoryBlock;

// Function prototypes
void bestFit(int blockSize[], int m, int processSize[], int n);
void printAllocation(MemoryBlock blocks[], int m, int processSize[], int n);

int main() {
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize) / sizeof(blockSize[0]);
    int n = sizeof(processSize) / sizeof(processSize[0]);

    bestFit(blockSize, m, processSize, n);

    return 0;
}
```

```
void bestFit(int blockSize[], int m, int processSize[], int n) {
    MemoryBlock blocks[MAX_BLOCKS];

    // Initialize memory blocks
    for (int i = 0; i < m; i++) {
        blocks[i].id = i;
        blocks[i].size = blockSize[i];
        blocks[i].allocated = -1; // -1 represents unallocated
    }

    // Allocate memory to processes
    for (int i = 0; i < n; i++) {
        int bestIdx = -1;
        for (int j = 0; j < m; j++) {
            if (blocks[j].size ≥ processSize[i]) {
                if (bestIdx == -1 || blocks[j].size < blocks[bestIdx].size)
{
                    bestIdx = j;
                }
            }
        }

        // If no block can accommodate the current process
        if (bestIdx == -1) {
            printf("Cannot allocate memory for process %d\n", i);
            continue;
        }

        // Allocate the process to the best-fit block
        blocks[bestIdx].allocated = i;
        blocks[bestIdx].size -= processSize[i];
    }

    // Print memory allocation
    printAllocation(blocks, m, processSize, n);
}

void printAllocation(MemoryBlock blocks[], int m, int processSize[], int n)
{
    printf("Process No. \tProcess Size \tBlock No.\n");
    for (int i = 0; i < n; i++) {
        printf("%d \t\t%d \t\t", i, processSize[i]);
```

```
        if (blocks[i].allocated ≠ -1)
            printf("%d\n", blocks[i].id);
        else
            printf("Not Allocated\n");
    }
}
```

even or odd

```
@echo off
REM Prompting user to input a number
echo Enter a number:
set /p num="Number: "

REM Checking if the number is even or odd
set /a remainder=%num% %% 2
if %remainder% equ 0 (
    echo %num% is an even number.
) else (
    echo %num% is an odd number.
)
```

4.question

worst-fit

```
#include <stdio.h>

#define MAX_BLOCKS 100
#define MAX_JOBS 100

void worstFit(int blocks[], int m, int jobs[], int n) {
    int allocation[n];
    for (int i = 0; i < n; i++) {
        int worst_index = -1;
        for (int j = 0; j < m; j++) {
            if (blocks[j] ≥ jobs[i]) {
                if (worst_index == -1 || blocks[j] > blocks[worst_index]) {
                    worst_index = j;
                }
```

```
            }
        }
        if (worst_index != -1) {
            allocation[i] = worst_index;
            blocks[worst_index] -= jobs[i];
        } else {
            allocation[i] = -1;
        }
    }
    printf("\nJob No.\tJob Size\tBlock No.\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t", i + 1, jobs[i]);
        if (allocation[i] != -1) {
            printf("%d\n", allocation[i] + 1);
        } else {
            printf("Not Allocated\n");
        }
    }
}

int main() {
    int m, n;
    int blocks[MAX_BLOCKS], jobs[MAX_JOBS];

    printf("Enter the number of memory blocks: ");
    scanf("%d", &m);

    printf("Enter the size of each memory block:\n");
    for (int i = 0; i < m; i++) {
        scanf("%d", &blocks[i]);
    }

    printf("Enter the number of jobs: ");
    scanf("%d", &n);

    printf("Enter the size of each job:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &jobs[i]);
    }

    worstFit(blocks, m, jobs, n);

    return 0;
```

```
}
```

bash

```
opendir()
readdir()
```

5.question

first-fit allocation

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_BLOCKS 100

// Structure to represent a memory block
typedef struct {
    int starting_address;
    int size;
    int allocated;
} MemoryBlock;

// Function to allocate memory using First Fit method
void firstFit(MemoryBlock blocks[], int num_blocks, int process_size) {
    int i;
    for (i = 0; i < num_blocks; i++) {
        if (blocks[i].allocated == 0 && blocks[i].size ≥ process_size) {
            blocks[i].allocated = 1;
            printf("Process of size %d allocated at memory block starting at
address %d\n", process_size, blocks[i].starting_address);
            return;
        }
    }
    printf("Process of size %d cannot be allocated\n", process_size);
}

int main() {
    MemoryBlock blocks[MAX_BLOCKS];
    int num_blocks, process_size, i;
```

```
    printf("Enter the number of memory blocks: ");
    scanf("%d", &num_blocks);

    printf("Enter details of memory blocks (starting address and size):\n");
    for (i = 0; i < num_blocks; i++) {
        printf("Block %d: ", i + 1);
        scanf("%d %d", &blocks[i].starting_address, &blocks[i].size);
        blocks[i].allocated = 0;
    }

    printf("Enter the size of the process to be allocated: ");
    scanf("%d", &process_size);

    firstFit(blocks, num_blocks, process_size);

    return 0;
}
```

bash

```
wait()
exit()
```

6.question

shortest job first s

```
#include <stdio.h>

// Process structure
struct Process {
    int pid;        // Process ID
    int burst_time; // Burst time
    int arrival_time; // Arrival time
    int waiting_time; // Waiting time
    int turnaround_time; // Turnaround time
    int completed; // Completion status
};

// Function to find the process with the shortest burst time
int findShortestJob(struct Process processes[], int n, int current_time) {
```

```
    int shortest_index = -1;
    int shortest_burst = 999999; // Initialize with a large value

    // Loop through all processes
    for (int i = 0; i < n; i++) {
        // Check if the process has arrived and not yet completed
        if (processes[i].arrival_time ≤ current_time &&
!processes[i].completed) {
            // Check if current process has shorter burst time
            if (processes[i].burst_time < shortest_burst) {
                shortest_index = i;
                shortest_burst = processes[i].burst_time;
            }
        }
    }

    return shortest_index;
}

// Function to calculate waiting time and turnaround time
void calculateTimes(struct Process processes[], int n) {
    int total_waiting_time = 0, total_turnaround_time = 0;

    // Calculate waiting time and turnaround time for each process
    for (int i = 0; i < n; i++) {
        processes[i].turnaround_time = processes[i].burst_time +
processes[i].waiting_time;
        total_waiting_time += processes[i].waiting_time;
        total_turnaround_time += processes[i].turnaround_time;
    }

    // Calculate average waiting time and turnaround time
    double avg_waiting_time = (double)total_waiting_time / n;
    double avg_turnaround_time = (double)total_turnaround_time / n;

    printf("Average Waiting Time: %.2lf\n", avg_waiting_time);
    printf("Average Turnaround Time: %.2lf\n", avg_turnaround_time);
}

// Function to perform Shortest Job First scheduling
void SJFScheduling(struct Process processes[], int n) {
    int current_time = 0;
```

```
    // Loop until all processes are completed
    while (1) {
        int shortest_index = findShortestJob(processes, n, current_time);

        // If no process is found, break the loop
        if (shortest_index == -1)
            break;

        // Execute the shortest job
        printf("Executing process P%d at time %d\n",
processes[shortest_index].pid, current_time);
        processes[shortest_index].waiting_time = current_time -
processes[shortest_index].arrival_time;
        current_time += processes[shortest_index].burst_time;
        processes[shortest_index].completed = 1;
    }

    // Calculate waiting time and turnaround time
    calculateTimes(processes, n);
}

int main() {
    // Example data
    struct Process processes[] = {
        {1, 6, 0, 0, 0, 0},
        {2, 8, 1, 0, 0, 0},
        {3, 7, 2, 0, 0, 0},
        {4, 3, 3, 0, 0, 0}
    };

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

    // Perform Shortest Job First scheduling
    SJFScheduling(processes, n);

    return 0;
}
```

unix cmds

```
cp , ls , grep
```

7.question

paging in os

```
#include <stdio.h>
#include <stdlib.h>

#define PAGE_SIZE 1024 // Page size in bytes
#define MEMORY_SIZE 4096 // Total memory size in bytes
#define PAGE_TABLE_SIZE (MEMORY_SIZE / PAGE_SIZE) // Number of pages

// Page Table Entry structure
struct PageTableEntry {
    int frame_number; // Frame number
    int valid; // Valid bit
};

// Function to initialize page table
void initializePageTable(struct PageTableEntry page_table[]) {
    for (int i = 0; i < PAGE_TABLE_SIZE; i++) {
        page_table[i].frame_number = -1; // Initialize with -1 indicating
frame not allocated
        page_table[i].valid = 0; // Initialize valid bit to 0
    }
}

// Function to simulate memory access
void accessMemory(struct PageTableEntry page_table[], int logical_address) {
    int page_number = logical_address / PAGE_SIZE;
    int offset = logical_address % PAGE_SIZE;

    if (page_table[page_number].valid) {
        printf("Accessing logical address %d (Page %d, Offset %d) - Physical
Address: %d\n",
               logical_address, page_number, offset,
page_table[page_number].frame_number * PAGE_SIZE + offset);
    } else {
        printf("Page Fault: Page %d is not in memory\n", page_number);
    }
}

int main() {
    struct PageTableEntry page_table[PAGE_TABLE_SIZE];
```

```
    initializePageTable(page_table);

    // Allocate some frames to pages
    page_table[0].frame_number = 0;
    page_table[0].valid = 1;
    page_table[1].frame_number = 1;
    page_table[1].valid = 1;
    page_table[2].frame_number = 2;
    page_table[2].valid = 1;

    // Simulate memory access
    accessMemory(page_table, 2048); // Accessing logical address 2048
    accessMemory(page_table, 4096); // Accessing logical address 4096
    accessMemory(page_table, 6144); // Accessing logical address 6144

    return 0;
}
```

unix

```
getpid()
close()
```

8.question

LFU

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_PAGES 100
#define MAX_FRAMES 10

struct Page {
    int pageNumber;
    int frequency;
};

int main() {
    int pages[MAX_PAGES];
    struct Page frames[MAX_FRAMES];
```

```
    int pageFaults = 0;
    int totalReferences = 0;
    int numFrames, numPages;

    printf("Enter the number of frames: ");
    scanf("%d", &numFrames);

    printf("Enter the number of pages: ");
    scanf("%d", &numPages);

    printf("Enter the page reference string: ");
    for (int i = 0; i < numPages; i++) {
        scanf("%d", &pages[i]);
    }

    for (int i = 0; i < numFrames; i++) {
        frames[i].pageNumber = -1;
        frames[i].frequency = 0;
    }

    for (int i = 0; i < numPages; i++) {
        int pageExists = 0;
        int leastFreqIndex = 0;
        int leastFreq = frames[0].frequency;

        totalReferences++;

        for (int j = 0; j < numFrames; j++) {
            if (frames[j].pageNumber == pages[i]) {
                frames[j].frequency++;
                pageExists = 1;
                break;
            }
            if (frames[j].frequency < leastFreq) {
                leastFreq = frames[j].frequency;
                leastFreqIndex = j;
            }
        }

        if (!pageExists) {
            pageFaults++;
            frames[leastFreqIndex].pageNumber = pages[i];
            frames[leastFreqIndex].frequency = 1;
```

```
        }
    }

    printf("Page Faults: %d\n", pageFaults);
    printf("Page Fault Rate: %.2f%%\n", (float)pageFaults / numPages * 100);
    printf("Hit Rate: %.2f%%\n", 100 - ((float)pageFaults / numPages *
100));

    return 0;
}
```

bash

```
@echo off
setlocal

:input
cls
echo Enter the first number:
set /p num1=
echo Enter the second number:
set /p num2=
echo Enter the operation (+, -, *, /):
set /p op=

rem Perform arithmetic operation
if "%op%"=="+" (
    set /a result=%num1%+%num2%
) else if "%op%"=="-" (
    set /a result=%num1%-%num2%
) else if "%op%"=="*" (
    set /a result=%num1%*%num2%
) else if "%op%"=="/" (
    set /a result=%num1%/%num2%
) else (
    echo Invalid operation. Please enter +, -, *, or /.
    pause
    goto input
)

echo Result: %result%
```

```
endlocal
```

9.question

LRU

- same as 2nd question

bash

```
@echo off
REM Prompting user to input three numbers
echo Enter three numbers:
set /p num1="Number 1: "
set /p num2="Number 2: "
set /p num3="Number 3: "

REM Finding the greatest number among the three
if %num1% gtr %num2% (
    if %num1% gtr %num3% (
        echo %num1% is the greatest number.
    ) else (
        echo %num3% is the greatest number.
    )
) else (
    if %num2% gtr %num3% (
        echo %num2% is the greatest number.
    ) else (
        echo %num3% is the greatest number.
    )
)
```

10.question

FCFS

```
#include <stdio.h>
#include <stdlib.h>
```

```c
#define MAX_FRAMES 10
#define MAX_PAGES 50

int main() {
    int reference_string[MAX_PAGES];
    int frames[MAX_FRAMES];
    int page_faults = 0;
    int num_pages, num_frames, i, j;

    printf("Enter the number of pages: ");
    scanf("%d", &num_pages);

    printf("Enter the reference string: ");
    for (i = 0; i < num_pages; i++) {
        scanf("%d", &reference_string[i]);
    }

    printf("Enter the number of frames: ");
    scanf("%d", &num_frames);

    // Initializing frames with -1 indicating empty frame
    for (i = 0; i < num_frames; i++) {
        frames[i] = -1;
    }

    printf("\nReference String\tPage Frames\tPage Faults\n");
    printf("————————————————————————————————————————————————————\n");

    // Implementing FCFS page replacement algorithm
    for (i = 0; i < num_pages; i++) {
        int page_found = 0;

        // Check if page already exists in frames
        for (j = 0; j < num_frames; j++) {
            if (frames[j] == reference_string[i]) {
                page_found = 1;
                break;
            }
        }

        if (!page_found) {
            frames[i % num_frames] = reference_string[i];
            page_faults++;
```

```
            printf("    %d\t\t\t", reference_string[i]);
            for (j = 0; j < num_frames; j++) {
                printf("%d ", frames[j]);
            }
            printf("\t\t%d\n", page_faults);
        }
    }

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

    return 0;
}
```

bash

```
close()
stat()
```

11.question

Banker's Algo

```
#include <stdio.h>

#define MAX_PROCESSES 10
#define MAX_RESOURCES 10

int available[MAX_RESOURCES];
int maximum[MAX_PROCESSES][MAX_RESOURCES];
int allocation[MAX_PROCESSES][MAX_RESOURCES];
int need[MAX_PROCESSES][MAX_RESOURCES];
int processes, resources;

void input()
{
    printf("Enter number of processes: ");
    scanf("%d", &processes);

    printf("Enter number of resources: ");
    scanf("%d", &resources);
```

```
    printf("Enter available resources: ");
    for (int i = 0; i < resources; i++)
        scanf("%d", &available[i]);

    printf("Enter maximum resources for each process:\n");
    for (int i = 0; i < processes; i++)
        for (int j = 0; j < resources; j++)
            scanf("%d", &maximum[i][j]);

    printf("Enter allocated resources for each process:\n");
    for (int i = 0; i < processes; i++)
        for (int j = 0; j < resources; j++)
        {
            scanf("%d", &allocation[i][j]);
            need[i][j] = maximum[i][j] - allocation[i][j];
        }
}

int safetyCheck(int work[], int finish[])
{
    int safeSequence[MAX_PROCESSES], k = 0;
    int tempFinish[MAX_PROCESSES];

    for (int i = 0; i < processes; i++)
        tempFinish[i] = finish[i];

    for (int i = 0; i < processes; i++)
    {
        int found = 0;
        for (int j = 0; j < resources; j++)
        {
            if (tempFinish[i] == 0 && need[i][j] ≤ work[j])
            {
                work[j] += allocation[i][j];
                tempFinish[i] = 1;
                found = 1;
            }
            else
            {
                found = 0;
                break;
            }
```

```
        }
        if (found)
            safeSequence[k++] = i;
    }

    if (k == processes)
    {
        printf("Safe Sequence: ");
        for (int i = 0; i < processes; i++)
            printf("%d ", safeSequence[i]);
        printf("\n");
        return 1;
    }
    else
        return 0;
}

void bankers()
{
    int work[MAX_RESOURCES];
    int finish[MAX_PROCESSES];
    for (int i = 0; i < resources; i++)
        work[i] = available[i];

    for (int i = 0; i < processes; i++)
        finish[i] = 0;

    int safe = safetyCheck(work, finish);
    if (safe)
        printf("The system is in safe state.\n");
    else
        printf("The system is in unsafe state.\n");
}

int main()
{
    input();
    bankers();
    return 0;
}
Enter number of processes: 5
Enter number of resources: 3
Enter available resources: 3 3 2
```

```
Enter maximum resources for each process:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter allocated resources for each process:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Safe Sequence: 1 3 4 0 2
The system is in safe state.
```

bash

```
@echo off
setlocal enabledelayedexpansion

echo Enter a number to find its factorial:
set /p num=

set factorial=1

if %num% LSS 0 (
    echo Factorial is not defined for negative numbers.
) else if %num% EQU 0 (
    echo Factorial of 0 is 1.
) else (
    for /l %%i in (1,1,%num%) do (
        set /a factorial*=%%i
    )
    echo Factorial of %num% is !factorial!.
)

endlocal
```

12.question

FCFS:

- same as 10th question

bash

- same as 4th question

13.question

Linked List File allocation

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_FILES 10

// Structure to represent a file
typedef struct File {
    int startBlock;
    int endBlock;
    char fileName[20];
    struct File* next;
} File;

// Function to initialize the linked list
File* initializeList() {
    return NULL;
}

// Function to create a new file entry
File* createFile(char fileName[], int startBlock, int endBlock) {
    File* newFile = (File*)malloc(sizeof(File));
    if (newFile == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newFile->startBlock = startBlock;
    newFile->endBlock = endBlock;
    strcpy(newFile->fileName, fileName);
    newFile->next = NULL;
    return newFile;
```

```
}

// Function to insert a new file entry at the end of the linked list
File* insertFile(File* head, char fileName[], int startBlock, int endBlock)
{
    File* newFile = createFile(fileName, startBlock, endBlock);
    if (head == NULL) {
        return newFile;
    }
    File* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newFile;
    return head;
}

// Function to display the file allocation
void displayFiles(File* head) {
    if (head == NULL) {
        printf("No files allocated.\n");
        return;
    }
    printf("File Allocation:\n");
    printf("Filename\tStart Block\tEnd Block\n");
    File* temp = head;
    while (temp != NULL) {
        printf("%s\t\t%d\t\t%d\n", temp->fileName, temp->startBlock, temp-
>endBlock);
        temp = temp->next;
    }
}

int main() {
    File* fileList = initializeList();

    // Example usage
    fileList = insertFile(fileList, "File1", 1, 4);
    fileList = insertFile(fileList, "File2", 5, 8);
    fileList = insertFile(fileList, "File3", 9, 12);

    displayFiles(fileList);
```

```
    return 0;
}
File Allocation:
Filename    Start Block    End Block
File1       1              4
File2       5              8
File3       9              12
```

bash

- same as eight question

14.question

indexed file allocation

- same as first one

bash

```
@echo off
setlocal enabledelayedexpansion

set /p n="Enter the value of n: "
set sum=0

echo Enter %n% numbers:

for /l %%i in (1,1,%n%) do (
    set /p num="Number %%i: "
    set /a sum+=num
)

echo Sum of %n% numbers is: %sum%

endlocal
```

15.question

sequential file allocation

```c
#include <stdio.h>


void main() {
    int f[50], i, st, len, j, c, k, count = 0;


    // Initialize the file allocation array
    for(i = 0; i < 50; i++)
        f[i] = 0;

    printf("Files Allocated are:\n");

x:  // Label for looping
    count = 0; // Reset count for each file

    // Input starting block and length of files
    printf("Enter starting block and length of files: ");
    scanf("%d%d", &st, &len);

    // Check if consecutive blocks are free for the file
    for(k = st; k < (st + len); k++)
        if(f[k] == 0)
            count++;

    if(len == count) {
        // Allocate blocks to the file
        for(j = st; j < (st + len); j++)
            if(f[j] == 0) {
                f[j] = 1;
                printf("%d\t%d\n", j, f[j]);
            }

        // Check if all blocks are allocated
        if(j != (st + len - 1))
            printf("The file is allocated to disk\n");
    } else {
        printf("The file is not allocated\n");
    }

    // Prompt for more files
    printf("Do you want to enter more file (Yes - 1 / No - 0): ");
    scanf("%d", &c);
```

```
    // Repeat if user wants to enter more files
    if(c == 1)
        goto x;
    else
        exit(); // Exit the program


}
```

bash

- same as 2nd one

16.question

producer-comsumer

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define BUFFER_SIZE 5

sem_t empty;
sem_t full;
pthread_mutex_t mutex;

int buffer[BUFFER_SIZE];
int in = 0, out = 0;

void *producer(void *arg) {
    int item = 1;
    while (1) {
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);

        buffer[in] = item;
        printf("Produced: %d\n", item);
        in = (in + 1) % BUFFER_SIZE;
        item++;
```

```
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
        sleep(1); // Sleep to simulate time taken for production
    }
}

void *consumer(void *arg) {
    while (1) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);

        int item = buffer[out];
        printf("Consumed: %d\n", item);
        out = (out + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
        sleep(2); // Sleep to simulate time taken for consumption
    }
}

int main() {
    pthread_t producer_thread, consumer_thread;

    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    pthread_mutex_init(&mutex, NULL);

    pthread_create(&producer_thread, NULL, producer, NULL);
    pthread_create(&consumer_thread, NULL, consumer, NULL);

    pthread_join(producer_thread, NULL);
    pthread_join(consumer_thread, NULL);

    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);

    return 0;
}
Produced: 1
Produced: 2
```

```
Produced: 3
Produced: 4
Produced: 5
Consumed: 1
Produced: 6
Consumed: 2
Produced: 7
Consumed: 3
Produced: 8
Consumed: 4
Produced: 9
Consumed: 5
Produced: 10
Consumed: 6
Produced: 11
Consumed: 7
Produced: 12
Consumed: 8
Produced: 13
Consumed: 9
Produced: 14
Consumed: 10
Produced: 15
Consumed: 11
Produced: 16
Consumed: 12
Produced: 17
Consumed: 13
Produced: 18
Consumed: 14
Produced: 19
Consumed: 15
Produced: 20
Consumed: 16
```

bash

```
@echo off
setlocal

:input
cls
```

```
echo Enter the first number:
set /p num1=
echo Enter the second number:
set /p num2=
echo Enter the operation (+, -, *, /):
set /p op=

rem Perform arithmetic operation
if "%op%"=="+" (
    set /a result=%num1%+%num2%
) else if "%op%"=="-" (
    set /a result=%num1%-%num2%
) else if "%op%"=="*" (
    set /a result=%num1%*%num2%
) else if "%op%"=="/" (
    set /a result=%num1%/%num2%
) else (
    echo Invalid operation. Please enter +, -, *, or /.
    pause
    goto input
)

echo Result: %result%

endlocal
```

17.question

Round-Robin

```
#include <stdio.h>

#define MAX_PROCESS 10
#define TIME_QUANTUM 2

struct Process {
    int id;
    int arrival_time;
    int burst_time;
    int remaining_time;
};
```

```
void roundRobin(struct Process processes[], int n) {
    int remaining_processes = n;
    int current_time = 0;
    int quantum = TIME_QUANTUM;

    printf("Time   Process\n");

    while (remaining_processes > 0) {
        for (int i = 0; i < n; i++) {
            if (processes[i].remaining_time > 0) {
                int execute_time = (processes[i].remaining_time < quantum) ?
processes[i].remaining_time : quantum;

                processes[i].remaining_time -= execute_time;
                current_time += execute_time;

                printf("%d-%d\tP%d\n", current_time - execute_time,
current_time, processes[i].id);

                if (processes[i].remaining_time == 0) {
                    remaining_processes--;
                }
            }
        }
    }
}

int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct Process processes[MAX_PROCESS];

    for (int i = 0; i < n; i++) {
        printf("Enter arrival time and burst time for process %d: ", i + 1);
        scanf("%d %d", &processes[i].arrival_time,
&processes[i].burst_time);
        processes[i].id = i + 1;
        processes[i].remaining_time = processes[i].burst_time;
    }

    roundRobin(processes, n);
```

```
    return 0;
}
```

bash

- same as previous one

18.question

priority

```
#include <stdio.h>

#define MAX_PROCESSES 10

struct Process {
    int id;
    int burst_time;
    int priority;
};

// Function to swap two processes
void swap(struct Process *a, struct Process *b) {
    struct Process temp = *a;
    *a = *b;
    *b = temp;
}

// Function to perform Priority Scheduling
void priorityScheduling(struct Process proc[], int n) {
    int i, j;

    // Sort processes based on priority (higher priority comes first)
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (proc[j].priority < proc[j + 1].priority) {
                swap(&proc[j], &proc[j + 1]);
            }
        }
    }
```

```
    int total_waiting_time = 0;
    int total_turnaround_time = 0;

    printf("Process Execution Order:\n");
    printf("Process ID   Burst Time   Priority\n");
    for (i = 0; i < n; i++) {
        printf("%6d %12d %10d\n", proc[i].id, proc[i].burst_time,
proc[i].priority);
        total_turnaround_time += (total_waiting_time + proc[i].burst_time);
        total_waiting_time += proc[i].burst_time;
    }

    printf("\nAverage Waiting Time: %.2f\n", (float)total_waiting_time / n);
    printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time /
n);
}

int main() {
    int n, i;
    struct Process proc[MAX_PROCESSES];

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    printf("Enter burst time and priority for each process:\n");
    for (i = 0; i < n; i++) {
        printf("Process %d: ", i + 1);
        scanf("%d %d", &proc[i].burst_time, &proc[i].priority);
        proc[i].id = i + 1;
    }

    priorityScheduling(proc, n);

    return 0;
}
```

bash

- same as 6th one

19.question

Inter-process communication

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    int fd[2]; // File descriptors for the pipe
    pid_t pid; // Process ID variable

    // Create the pipe
    if (pipe(fd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    // Create a child process
    pid = fork();
    if (pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) { // Child process
        close(fd[0]); // Close the read end of the pipe

        // Redirect stdout to the pipe
        dup2(fd[1], STDOUT_FILENO);

        // Execute a command (e.g., "ls")
        execlp("ls", "ls", NULL);

        // If execlp fails
        perror("execlp");
        exit(EXIT_FAILURE);
    } else { // Parent process
        close(fd[1]); // Close the write end of the pipe

        // Read from the pipe and print to stdout
        char buffer[4096];
        ssize_t nbytes;
        while ((nbytes = read(fd[0], buffer, sizeof(buffer))) > 0) {
```

```
            write(STDOUT_FILENO, buffer, nbytes);
        }

        // Wait for the child process to finish
        wait(NULL);
    }

    return 0;
}
```

bash

- same as 10th one

20.question

Threading

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define BUFFER_SIZE 10

int buffer[BUFFER_SIZE];
int count = 0; // Number of items in the buffer
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond_prod = PTHREAD_COND_INITIALIZER;
pthread_cond_t cond_cons = PTHREAD_COND_INITIALIZER;

void *producer(void *arg) {
    int item = 0;
    while (1) {
        pthread_mutex_lock(&mutex);
        while (count == BUFFER_SIZE) {
            pthread_cond_wait(&cond_prod, &mutex);
        }
        buffer[count] = item;
        printf("Produced: %d\n", item);
        count++;
        item++;
        pthread_cond_signal(&cond_cons);
```

```
        pthread_mutex_unlock(&mutex);
    }
}

void *consumer(void *arg) {
    while (1) {
        pthread_mutex_lock(&mutex);
        while (count == 0) {
            pthread_cond_wait(&cond_cons, &mutex);
        }
        int item = buffer[count - 1];
        printf("Consumed: %d\n", item);
        count--;
        pthread_cond_signal(&cond_prod);
        pthread_mutex_unlock(&mutex);
    }
}

int main() {
    pthread_t producer_thread, consumer_thread;

    // Create producer and consumer threads
    pthread_create(&producer_thread, NULL, producer, NULL);
    pthread_create(&consumer_thread, NULL, consumer, NULL);

    // Wait for threads to finish
    pthread_join(producer_thread, NULL);
    pthread_join(consumer_thread, NULL);

    return 0;
}
```

bash

- same as 16th