B.E / B.Tech. PRACTICAL END SEMESTER EXAMINATIONS, APRIL/MAY 2025

Fourth Semester

AL3452 – OPERATING SYSTEMS

(Regulations 2021)

Time : 3 Hours Answer any one Question Max. Marks 100

Aim & Procedure	Program	Results	Viva-Voce	Record	Total
20	**40**	**20**	**10**	**10**	**100**

1. Write a C program to implement memory allocations using best fit algorithm.

```
#include <stdio.h>

void bestFit(int blockSize[], int m, int processSize[], int n) {

int allocation[n];



    // Initialize all allocations as -1 (indicating not allocated)

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

        allocation[i] = -1;



    // Pick each process and find the best fit block

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

        int bestIdx = -1;

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

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

                if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx])

                    bestIdx = j;
            }
        }

        // If a suitable block was found
        if (bestIdx != -1) {
            allocation[i] = bestIdx;
            blockSize[bestIdx] -= processSize[i];
        }
    }

    printf("Process No.\tProcess Size\tBlock no.\n");
    for (int i = 0; i < n; i++) {
        printf(" %d\t\t%d\t\t", i + 1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d", allocation[i] + 1);
        else
            printf("Not Allocated");
        printf("\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;

}
```

SAMPLE OUTPUT

```
Process No.     Process Size    Block no.

 1              212             4

 2              417             2

 3              112             3

 4              426             5
```

2. Write a C program to implement the concept of Paging.

```
#include <stdio.h>



int main() {

    int page_size = 128;

    int logical_memory_size = 1024;

    int physical_memory_size = 512;



    int num_pages = logical_memory_size / page_size;

    int num_frames = physical_memory_size / page_size;



    int page_table[num_pages];



    // Initialize page table with -1 (indicating not allocated)

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

        page_table[i] = -1;

    }



    // Manually allocate frames to pages

    page_table[0] = 3;
```

```
page_table[1] = 1;
page_table[2] = 0;
page_table[3] = -1; // Not allocated


// Display page table
printf("Page Table:\n");
printf("Page Number\tFrame Number\n");
for (int i = 0; i < num_pages; i++) {
    if (page_table[i] != -1)
        printf("%d\t\t%d\n", i, page_table[i]);
    else
        printf("%d\t\tNot Allocated\n", i);
}


// Translate logical address to physical address
int logical_address = 290;
int page_number = logical_address / page_size;
int offset = logical_address % page_size;


if (page_table[page_number] != -1) {
    int frame_number = page_table[page_number];
```

```
        int physical_address = frame_number * page_size + offset;
        printf("\nLogical Address: %d\n", logical_address);
        printf("Physical Address: %d\n", physical_address);
    } else {
        printf("\nPage fault! The page is not allocated in any frame.\n");
    }

    return 0;
}
```

OUTPUT

```
Page Table:
Page Number     Frame Number
0               3
1               1
2               0
3               Not Allocated

Logical Address: 290
Physical Address: 162
```

3. Write a C program to implement page replacement FIFO (First In First Out) algorithm

```
#include <stdio.h>

#define MAX_FRAMES 10

int main() {
    int pages[30], frames[MAX_FRAMES], n, f, i, j, k, pageFaults = 0, next = 0, found;

    printf("Enter number of pages: ");
    scanf("%d", &n);
    printf("Enter the page reference string: ");
    for(i = 0; i < n; i++)
        scanf("%d", &pages[i]);

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

    for(i = 0; i < f; i++)
        frames[i] = -1;
```

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

    found = 0;

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

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

            found = 1;

            break;

        }

    }

    if(!found) {

        frames[next] = pages[i];

        next = (next + 1) % f;

        pageFaults++;

    }

    printf("Frames: ");

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

        if(frames[k] != -1)

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

        else
```

```
                printf("- ");
        }
        printf("\n");
    }
    printf("Total Page Faults: %d\n", pageFaults);
    return 0;
}
```

OUTPUT

```
Enter number of pages: 7
Enter the page reference string: 1 4 0 4 5 3 7
Enter number of frames: 3
Frames: 1 - -
Frames: 1 4 -
Frames: 1 4 0
Frames: 1 4 0
Frames: 5 4 0
Frames: 5 3 0
Frames: 5 3 7
Total Page Faults: 6
```

4. Write a C program to implement Deadlock Detection algorithm.

```
#include <stdio.h>



#define MAX 10



int main() {

    int allocation[MAX][MAX], request[MAX][MAX], available[MAX];

    int work[MAX], finish[MAX];

    int n, m, i, j, k, flag;



    printf("Enter number of processes: ");

    scanf("%d", &n);

    printf("Enter number of resources: ");

    scanf("%d", &m);



    printf("Enter Allocation Matrix:\n");

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

        for(j = 0; j < m; j++)

            scanf("%d", &allocation[i][j]);
```

```
printf("Enter Request Matrix:\n");

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

    for(j = 0; j < m; j++)

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



printf("Enter Available Resources:\n");

for(j = 0; j < m; j++)

    scanf("%d", &available[j]);



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

    finish[i] = 0;



for(j = 0; j < m; j++)

    work[j] = available[j];



int done;

do {

    done = 0;
    for(i = 0; i < n; i++) {
        if(!finish[i]) {
            flag = 1;
            for(j = 0; j < m; j++) {
                if(request[i][j] > work[j]) {
                    flag = 0;
                    break;
                }
            }
            if(flag) {
                for(k = 0; k < m; k++)
                    work[k] += allocation[i][k];
                finish[i] = 1;
                done = 1;
            }
        }
    }
} while(done);
```

```
    flag = 0;

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

        if(!finish[i]) {

            flag = 1;

            break;

        }

    }



    if(flag)

        printf("Deadlock detected.\n");

    else

        printf("No deadlock detected.\n");



    return 0;

}
```

OUTPUT

```
Enter number of processes: 3

Enter number of resources: 3

Enter Allocation Matrix:
```

```
0 1 0

2 0 0

3 0 3

Enter Request Matrix:

0 0 0

2 0 2

0 0 0

Enter Available Resources:

0 0 0

Deadlock detected.
```

5 Write a C program to implement File Organization concept using the technique two level directory.

```
#include <stdio.h>

#include <string.h>



#define MAX_USERS 5

#define MAX_FILES 5

#define NAME_LEN 20



typedef struct {

    char filename[NAME_LEN];

} File;



typedef struct {

    char username[NAME_LEN];

    File files[MAX_FILES];

    int file_count;

} UserDirectory;



UserDirectory users[MAX_USERS];
```

```
int user_count = 0;

int find_user_index(char *username) {
    for (int i = 0; i < user_count; i++) {
        if (strcmp(users[i].username, username) == 0)
            return i;
    }
    return -1;
}

void create_user_directory() {
    if (user_count >= MAX_USERS) {
        printf("Maximum user limit reached.\n");
        return;
    }
    char username[NAME_LEN];
    printf("Enter username: ");
    scanf("%s", username);
    if (find_user_index(username) != -1) {
        printf("User directory already exists.\n");

        return;

    }

    strcpy(users[user_count].username, username);

    users[user_count].file_count = 0;

    user_count++;

    printf("User directory '%s' created.\n", username);

}



void create_file() {

    char username[NAME_LEN], filename[NAME_LEN];

    printf("Enter username: ");

    scanf("%s", username);

    int idx = find_user_index(username);

    if (idx == -1) {

        printf("User directory not found.\n");

        return;

    }

    if (users[idx].file_count >= MAX_FILES) {
        printf("Maximum file limit reached for user '%s'.\n", username);
        return;
    }
    printf("Enter filename: ");
    scanf("%s", filename);
    for (int i = 0; i < users[idx].file_count; i++) {
        if (strcmp(users[idx].files[i].filename, filename) == 0) {
            printf("File already exists.\n");
            return;
        }
    }
    strcpy(users[idx].files[users[idx].file_count].filename, filename);
    users[idx].file_count++;
    printf("File '%s' created in user directory '%s'.\n", filename, username);
}


void display_directories() {
    if (user_count == 0) {
        printf("No user directories found.\n");
        return;
    }
    for (int i = 0; i < user_count; i++) {
        printf("User Directory: %s\n", users[i].username);
        if (users[i].file_count == 0) {
            printf("  No files.\n");
        } else {
            for (int j = 0; j < users[i].file_count; j++) {
                printf("  File: %s\n", users[i].files[j].filename);
            }
        }
    }
}

int main() {
    int choice;
    while (1) {
        printf("\nTwo-Level Directory Simulation\n");
        printf("1. Create User Directory\n");
printf("2. Create File\n");
printf("3. Display Directories\n");
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
    case 1:
        create_user_directory();
        break;
    case 2:
        create_file();
        break;
    case 3:
        display_directories();
        break;
    case 4:
        return 0;
    default:
        printf("Invalid choice. Please try again.\n");
        }
    }
}
```

OUTPUT

```
 Two-Level Directory Simulation

 1. Create User Directory

 2. Create File

 3. Display Directories

 4. Exit

 Enter your choice: 1

 Enter username: alice

 User directory 'alice' created.

Enter your choice: 2

 Enter username: alice

 Enter filename: report.txt

 File 'report.txt' created in user directory 'alice'.

Enter your choice: 3

 User Directory: alice

   File: report.txt

 Enter your choice: 4
```

6 Write a C program to implement the Producer & consumer Problem (Semaphore)

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

#define BUFFER_SIZE 5
#define MAX_ITEMS 10

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

sem_t empty;
sem_t full;
pthread_mutex_t mutex;

void* producer(void* arg) {
    for (int i = 0; i < MAX_ITEMS; i++) {
        int item = rand() % 100;
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        buffer[in] = item;
        printf("Produced: %d\n", item);
        in = (in + 1) % BUFFER_SIZE;
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
        sleep(1); // Simulate production time
    }
    return NULL;
}

void* consumer(void* arg) {
    for (int i = 0; i < MAX_ITEMS; i++) {
        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(1); // Simulate consumption time
    }
    return NULL;
}

int main() {
    pthread_t prod, cons;

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

    pthread_create(&prod, NULL, producer, NULL);
    pthread_create(&cons, NULL, consumer, NULL);
    pthread_join(prod, NULL);
    pthread_join(cons, NULL);

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

    return 0;
}
```

OUTPUT

```
Produced: 42
Consumed: 42
Produced: 17
Consumed: 17
...
```

7. Write a C program to implement File Allocation concept using the technique indexed allocation technique.

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

#define MAX_BLOCKS 50

int main() {
    int total_blocks, index_block, num_blocks, i, j, choice;
    int blocks[MAX_BLOCKS] = {0}; // 0: free, 1: allocated

    printf("Enter total number of blocks: ");
    scanf("%d", &total_blocks);

    do {
        printf("\nEnter index block number (0 to %d): ", total_blocks - 1);
        scanf("%d", &index_block);

        if (index_block >= total_blocks || blocks[index_block]) {
```

```
    printf("Invalid or already allocated index block.\n");
    continue;
}

printf("Enter number of blocks needed for the file: ");
scanf("%d", &num_blocks);

int data_blocks[num_blocks];
int allocated = 0;

printf("Enter block numbers:\n");
for (i = 0; i < num_blocks; i++) {
    scanf("%d", &data_blocks[i]);
    if (data_blocks[i] >= total_blocks || blocks[data_blocks[i]]) {
        printf("Block %d is invalid or already allocated.\n", data_blocks[i]);
        break;
    }
}
```

```
        if (i == num_blocks) {
            blocks[index_block] = 1;
            for (i = 0; i < num_blocks; i++)
                blocks[data_blocks[i]] = 1;

            printf("File allocated successfully.\nIndex Block: %d\nBlocks: ", index_block);
            for (i = 0; i < num_blocks; i++)
                printf("%d ", data_blocks[i]);
            printf("\n");
        } else {
            printf("Allocation failed. Try again.\n");
        }

        printf("Do you want to allocate another file? (1: Yes, 0: No): ");
        scanf("%d", &choice);
    } while (choice == 1);

    return 0;
}
```

OUTPUT

```
Enter total number of blocks: 10



Enter index block number (0 to 9): 2

Enter number of blocks needed for the file: 3

Enter block numbers:

3 4 5

File allocated successfully.

Index Block: 2

Blocks: 3 4 5



Do you want to allocate another file? (1: Yes, 0: No): 0
```

8 Write a C program to implement the first come first serve without arrival time CPU scheduling algorithm.

```
#include <stdio.h>


int main() {

    int n, bt[10], wt[10], tat[10], i;

    float avg_wt = 0, avg_tat = 0;


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

    scanf("%d", &n);


    printf("Enter burst times for each process:\n");

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

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

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

    }


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

    for (i = 1; i < n; i++) {
        wt[i] = bt[i - 1] + wt[i - 1]; // Waiting time = sum of burst times of previous processes
    }

    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time");
    for (i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i]; // Turnaround time = burst time + waiting time
        avg_wt += wt[i];
        avg_tat += tat[i];
        printf("\nP%d\t\t%d\t\t%d\t\t%d", i + 1, bt[i], wt[i], tat[i]);
    }

    avg_wt /= n;
    avg_tat /= n;

    printf("\n\nAverage Waiting Time: %.2f", avg_wt);
    printf("\nAverage Turnaround Time: %.2f", avg_tat);

    return 0;
}
```

OUTPUT

```
Enter the number of processes: 3
Enter burst times for each process:
P1: 4
P2: 3
P3: 5

Process  Burst Time  Waiting Time  Turnaround Time
P1      4       0        4
P2      3       4        7
P3      5       7        12

Average Waiting Time: 3.67
Average Turnaround Time: 7.67
```

9. Write a C program to implement the CPU scheduling algorithm for shortest job first.

```
#include <stdio.h>



void sort_by_burst_time(int n, int bt[], int p[]) {

    int temp, pos;

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

        pos = i;

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

            if (bt[j] < bt[pos]) pos = j;

        }

        // Swap burst times

        temp = bt[i];

        bt[i] = bt[pos];

        bt[pos] = temp;

        // Swap process IDs

        temp = p[i];

        p[i] = p[pos];

        p[pos] = temp;

    }
```

```
}


int main() {
    int n, total_wt = 0, total_tat = 0;
    float avg_wt, avg_tat;

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

    int bt[n], wt[n], tat[n], p[n];
    for (int i = 0; i < n; i++) {
        p[i] = i + 1; // Process IDs
        printf("Enter burst time for P%d: ", p[i]);
        scanf("%d", &bt[i]);
    }

    sort_by_burst_time(n, bt, p);

    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];
    total_wt += wt[i];
}

printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time");
for (int i = 0; i < n; i++) {
    tat[i] = bt[i] + wt[i];
    total_tat += tat[i];
    printf("\nP%d\t\t%d\t\t%d\t\t%d", p[i], bt[i], wt[i], tat[i]);
}

avg_wt = (float)total_wt / n;
avg_tat = (float)total_tat / n;

printf("\n\nAverage Waiting Time: %.2f", avg_wt);
printf("\nAverage Turnaround Time: %.2f", avg_tat);

return 0;
```

```
}
```

OUTPUT

```
Enter number of processes: 3

Enter burst time for P1: 6

Enter burst time for P2: 2

Enter burst time for P3: 8



Process  Burst Time  Waiting Time  Turnaround Time

P2         2           0             2

P1         6           2             8

P3         8           8             16



Average Waiting Time: 3.33

Average Turnaround Time: 8.67
```

i. Write a shell program to solve arithmetic operation.

10 ii. Write a shell program to check whether the number is odd or even

11 Write a C program to implement File Organization concept using the technique Single level directory

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

#define MAX_FILES 10
#define FILENAME_LENGTH 20

struct Directory {
    char files[MAX_FILES][FILENAME_LENGTH];
    int fileCount;
};

void createFile(struct Directory *dir) {
    if (dir->fileCount < MAX_FILES) {
        printf("Enter file name: ");
        scanf("%s", dir->files[dir->fileCount]);
        dir->fileCount++;
        printf("File created successfully.\n");
    } else {
        printf("Directory is full. Cannot create more files.\n");
    }
}

void deleteFile(struct Directory *dir) {
    char fileName[FILENAME_LENGTH];
    printf("Enter file name to delete: ");
    scanf("%s", fileName);
    for (int i = 0; i < dir->fileCount; i++) {
        if (strcmp(dir->files[i], fileName) == 0) {
            for (int j = i; j < dir->fileCount - 1; j++) {
                strcpy(dir->files[j], dir->files[j + 1]);
            }
            dir->fileCount--;
            printf("File deleted successfully.\n");
            return;
        }

    }

    printf("File not found.\n");

}



void searchFile(struct Directory *dir) {

    char fileName[FILENAME_LENGTH];

    printf("Enter file name to search: ");

    scanf("%s", fileName);

    for (int i = 0; i < dir->fileCount; i++) {

        if (strcmp(dir->files[i], fileName) == 0) {

            printf("File found: %s\n", dir->files[i]);

            return;

        }

    }

    printf("File not found.\n");

}



void displayFiles(struct Directory *dir) {
    if (dir->fileCount == 0) {
        printf("No files in the directory.\n");
    } else {
        printf("Files in the directory:\n");
        for (int i = 0; i < dir->fileCount; i++) {
            printf("%s\n", dir->files[i]);
        }
    }
}


int main() {
    struct Directory dir = {.fileCount = 0};
    int choice;
    do {
        printf("\n1. Create File\n2. Delete File\n3. Search File\n4. Display Files\n5. Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: createFile(&dir); break;
            case 2: deleteFile(&dir); break;
            case 3: searchFile(&dir); break;

            case 4: displayFiles(&dir); break;

            case 5: printf("Exiting...\n"); break;

            default: printf("Invalid choice. Please try again.\n");

        }

    } while (choice != 5);

    return 0;

}
```

OUTPUT

```
1. Create File

2. Delete File

3. Search File

4. Display Files

5. Exit

Enter your choice: 1

Enter file name: file1.txt

File created successfully.



1. Create File

2. Delete File

3. Search File
```

```
4. Display Files
5. Exit
Enter your choice: 4
Files in the directory:
file1.txt

1. Create File
2. Delete File
3. Search File
4. Display Files
5. Exit
Enter your choice: 2
Enter file name to delete: file1.txt
File deleted successfully.

1. Create File
2. Delete File
3. Search File
4. Display Files
5. Exit
Enter your choice: 4
No files in the directory.
```

12 Write a C program to implement sequential file for processing the student information.

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

struct Student {
    int usn;
    char name[50];
    int marks[3];
};

void writeStudent(FILE *file) {
    struct Student s;
    printf("Enter USN: ");
    scanf("%d", &s.usn);
    printf("Enter Name: ");
    getchar(); // to consume the newline character left by previous scanf
    fgets(s.name, sizeof(s.name), stdin);
    s.name[strcspn(s.name, "\n")] = '\0'; // remove newline character
    printf("Enter marks for 3 subjects: ");
    for (int i = 0; i < 3; i++) {
        scanf("%d", &s.marks[i]);
    }
    fwrite(&s, sizeof(s), 1, file);
}

void displayStudents(FILE *file) {
    struct Student s;
    rewind(file);
    printf("\nUSN\tName\t\tMarks\n");
    while (fread(&s, sizeof(s), 1, file)) {
        printf("%d\t%s\t", s.usn, s.name);
        for (int i = 0; i < 3; i++) {
            printf("%d ", s.marks[i]);
        }
        printf("\n");
    }
}

void searchStudent(FILE *file, int usn) {
    struct Student s;
    rewind(file);
    while (fread(&s, sizeof(s), 1, file)) {
        if (s.usn == usn) {
            printf("\nRecord Found: %d\t%s\t", s.usn, s.name);
            for (int i = 0; i < 3; i++) {
                printf("%d ", s.marks[i]);
            }
            printf("\n");
            return;
        }
    }
    printf("\nRecord with USN %d not found.\n", usn);
}

int main() {
    FILE *file = fopen("students.dat", "a+b");
    if (!file) {
        printf("Error opening file.\n");
        return 1;
    }

    int choice, usn;
    do {
        printf("\n1. Add Student Record\n2. Display All Records\n3. Search Record by USN\n4. Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                writeStudent(file);
                break;
            case 2:
                displayStudents(file);
                break;
            case 3:
                printf("Enter USN to search: ");
                scanf("%d", &usn);
                searchStudent(file, usn);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 4);

    fclose(file);
    return 0;
}
```

OUTPUT

```
1. Add Student Record
2. Display All Records
3. Search Record by USN
4. Exit
Enter your choice: 1
Enter USN: 1
Enter Name: John Doe
Enter marks for 3 subjects: 85 90 88

1. Add Student Record
2. Display All Records
3. Search Record by USN
4. Exit
Enter your choice: 2

USN    Name          Marks
1      John Doe      85 90 88

1. Add Student Record
2. Display All Records
3. Search Record by USN
4. Exit
Enter your choice: 3
Enter USN to search: 1

Record Found: 1   John Doe    85 90 88

1. Add Student Record
2. Display All Records
3. Search Record by USN
4. Exit
Enter your choice: 4
Exiting...
```

13 Write a C program to implement Deadlock avoidance by using Banker’s algorithm.

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

#define MAX_PROCESSES 5
#define MAX_RESOURCES 3

int available[MAX_RESOURCES];
int max[MAX_PROCESSES][MAX_RESOURCES];
int allocation[MAX_PROCESSES][MAX_RESOURCES];
int need[MAX_PROCESSES][MAX_RESOURCES];

bool isSafe(int processes[], int avail[], int max[][MAX_RESOURCES], int
allot[][MAX_RESOURCES], int n, int m) {
    int work[m];
    bool finish[n];
    int safeSeq[n];
    int count = 0;
    for (int i = 0; i < m; i++) work[i] = avail[i];
    for (int i = 0; i < n; i++) finish[i] = false;
```

```
while (count < n) {
    bool found = false;
    for (int p = 0; p < n; p++) {
        if (!finish[p]) {
            int j;
            for (j = 0; j < m; j++)
                if (need[p][j] > work[j])
                    break;
            if (j == m) {
                for (int k = 0; k < m; k++)
                    work[k] += allot[p][k];
                safeSeq[count++] = p;
                finish[p] = true;
                found = true;
            }
        }
    }
    if (!found) {
        printf("System is in an unsafe state.\n");
```

```
            return false;
        }
    }
    printf("System is in a safe state.\nSafe sequence is: ");
    for (int i = 0; i < n; i++)
        printf("P%d ", safeSeq[i]);
    printf("\n");
    return true;
}


int main() {
    int n = 5; // Number of processes
    int m = 3; // Number of resources
    int processes[MAX_PROCESSES] = {0, 1, 2, 3, 4};


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

```
// Maximum demand of each process
printf("Enter maximum demand of each process:\n");
for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
        scanf("%d", &max[i][j]);

// Allocation of resources to processes
printf("Enter allocation of resources to processes:\n");
for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
        scanf("%d", &allocation[i][j]);

// Calculate need matrix
for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
        need[i][j] = max[i][j] - allocation[i][j];

// Check system's safety
```

```
    isSafe(processes, available, max, allocation, n, m);


    return 0;
}
```

OUTPUT

```
Enter available instances of resources: 3 3 2
Enter maximum demand of each process:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter allocation of resources to processes:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
System is in a safe state.
Safe sequence is: P0 P1 P3 P4 P2
```

14 Write a C program to implement shared memory and inter process communication.

PROUDCER

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>



#define SHM_SIZE 1024

#define SHM_KEY 1234



int main() {

  int shmid;
```

```
char *shm_ptr;

// Create shared memory segment
shmid = shmget(SHM_KEY, SHM_SIZE, 0666 | IPC_CREAT);
if (shmid == -1) {
    perror("shmget failed");
    exit(1);
}

// Attach to shared memory
shm_ptr = shmat(shmid, NULL, 0);
if (shm_ptr == (char *)-1) {
    perror("shmat failed");
    exit(1);
}

// Write data to shared memory
printf("Enter a message: ");
fgets(shm_ptr, SHM_SIZE, stdin);

// Detach from shared memory
if (shmdt(shm_ptr) == -1) {
    perror("shmdt failed");
    exit(1);
}
```

```
    return 0;
}
```

CONSUMER

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SHM_SIZE 1024
#define SHM_KEY 1234

int main() {
    int shmid;
```

```
char *shm_ptr;

// Access shared memory segment
shmid = shmget(SHM_KEY, SHM_SIZE, 0666);
if (shmid == -1) {
    perror("shmget failed");
    exit(1);
}

// Attach to shared memory
shm_ptr = shmat(shmid, NULL, 0);
if (shm_ptr == (char *)-1) {
    perror("shmat failed");
    exit(1);
}

// Read and display data from shared memory
printf("Message from shared memory: %s\n", shm_ptr);

// Detach from shared memory
if (shmdt(shm_ptr) == -1) {
    perror("shmdt failed");
    exit(1);
}

return 0;
```

```
}
```

OUTPUT

```
Enter a message: Hello from producer!

Message from shared memory: Hello from producer!
```

15 Write programs using the following system calls of UNIX operating system: fork, exec, getpid, exit, wait, close, stat, opendir, readdir.

16. Write a C program to implement the concept of Segmentation.

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

#define NUM_SEGMENTS 3

// Structure to represent a segment
typedef struct {
    char name[20];
    int size;
    int base;
    int limit;
} Segment;

// Function to display segment table
void displaySegmentTable(Segment segments[], int num) {
    printf("\nSegment Table:\n");
    printf("Segment Name | Base Address | Limit | Size\n");
    for (int i = 0; i < num; i++) {
        printf("%-13s | %-12d | %-5d | %-4d\n", segments[i].name, segments[i].base, segments[i].limit, segments[i].size);
    }
}

int main() {
    Segment segments[NUM_SEGMENTS] = {
        {"Code", 100, 0, 100},
        {"Data", 200, 100, 300},
        {"Stack", 150, 300, 450}
    };

    // Display the segment table
    displaySegmentTable(segments, NUM_SEGMENTS);

    // Simulate accessing an address within the Data segment
    int segmentNumber = 1; // Data segment
    int offset = 50; // Address within the Data segment
    if (offset < segments[segmentNumber].size) {

        printf("\nAccessing address %d within %s segment: Success\n", offset,
segments[segmentNumber].name);

    } else {

        printf("\nAccessing address %d within %s segment: Error (Out of bounds)\n", offset,
segments[segmentNumber].name);

    }



    return 0;

}
```

OUTPUT

```
Segment Table:

Segment Name | Base Address | Limit | Size

Code         | 0            | 100   | 100

Data         | 100          | 300   | 200

Stack        | 300          | 450   | 150



Accessing address 50 within Data segment: Success
```

17. Write a C program to simulate UNIX commands like cp, ls, grep.

18. i.Write a Shell program to find the factorial of a number.

ii. Write a Shell program to check the given year is leap year or not.

19. Write a C program to implement random access file for processing the employee details.

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

struct Employee {
    int id;
    char name[50];
    int age;
    float salary;
};

void addEmployee(FILE *file, int index) {
    struct Employee emp;
    fseek(file, index * sizeof(struct Employee), SEEK_SET);
    printf("Enter ID: ");
    scanf("%d", &emp.id);
    printf("Enter Name: ");
    scanf("%s", emp.name);
    printf("Enter Age: ");

    scanf("%d", &emp.age);

    printf("Enter Salary: ");

    scanf("%f", &emp.salary);

    fwrite(&emp, sizeof(struct Employee), 1, file);

}



void displayEmployee(FILE *file, int index) {

    struct Employee emp;

    fseek(file, index * sizeof(struct Employee), SEEK_SET);

    if (fread(&emp, sizeof(struct Employee), 1, file)) {

        printf("ID: %d\n", emp.id);

        printf("Name: %s\n", emp.name);

        printf("Age: %d\n", emp.age);

        printf("Salary: %.2f\n", emp.salary);

    } else {

        printf("No record found at index %d.\n", index);

    }

}



int main() {
```

```
FILE *file = fopen("employees.dat", "rb+");
if (!file) {
    file = fopen("employees.dat", "wb+");
    if (!file) {
        perror("Unable to open file");
        return EXIT_FAILURE;
    }
}

int choice, index;
while (1) {
    printf("\nMenu:\n");
    printf("1. Add Employee\n");
    printf("2. Display Employee\n");
    printf("3. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter index to add employee: ");
```

```
                scanf("%d", &index);
                addEmployee(file, index);
                break;
            case 2:
                printf("Enter index to display employee: ");
                scanf("%d", &index);
                displayEmployee(file, index);
                break;
            case 3:
                fclose(file);
                return EXIT_SUCCESS;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
}
```

OUTPUT

```
Menu:
1. Add Employee
2. Display Employee
3. Exit
```

```
Enter your choice: 1
Enter index to add employee: 0
Enter ID: 101
Enter Name: John
Enter Age: 30
Enter Salary: 50000

Menu:
1. Add Employee
2. Display Employee
3. Exit
Enter your choice: 2
Enter index to display employee: 0
ID: 101
Name: John
Age: 30
Salary: 50000.00
```

20 Write a C program to implement page replacement LRU (Least Recently Used) algorithm.

```
#include <stdio.h>


void LRU(int pages[], int n, int frames) {

    int frame[frames];

    int time[frames];

    int page_faults = 0, counter = 0;


    // Initialize frames and time

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

        frame[i] = -1;

        time[i] = -1;

    }


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

        int page = pages[i];

        int found = 0;


        // Check if page is already in frame
```

```
for (int j = 0; j < frames; j++) {
    if (frame[j] == page) {
        found = 1;
        time[j] = counter++;
        break;
    }
}

// If page is not found, replace the least recently used page
if (!found) {
    int lru = 0;
    for (int j = 1; j < frames; j++) {
        if (time[j] < time[lru]) {
            lru = j;
        }
    }
    frame[lru] = page;
    time[lru] = counter++;
    page_faults++;
```

```
        }

        // Display current frames
        printf("Page %d: ", page);
        for (int j = 0; j < frames; j++) {
            if (frame[j] != -1)
                printf("%d ", frame[j]);
            else
                printf("- ");
        }
        printf("\n");
    }

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

int main() {
    int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3};
    int n = sizeof(pages) / sizeof(pages[0]);
    int frames = 3;



    LRU(pages, n, frames);

    return 0;

}
```

OUTPUT

```
Page 7: 7 - -

Page 0: 7 0 -

Page 1: 7 0 1

Page 2: 2 0 1

Page 0: 2 0 1

Page 3: 2 3 1

Page 0: 0 3 1

Page 4: 0 3 4

Page 2: 0 3 4

Page 3: 0 3 4

Page 0: 0 3 4

Page 3: 0 3 4

Page 2: 0 3 4



Total Page Faults: 9
```