Program 2.2: Develop a C program to implement SJF Scheduling

```
#include <stdio.h>

int main() {
    int bt[20], p[20], wt[20], tat[20], i, j, n, total_tat = 0, total_wt = 0,
temp;
    float avg_wt, avg_tat;

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

    printf("Enter Burst Time:\n");
    for(i = 0; i < n; i++) {
        printf("P%d: ", i + 1);
        scanf("%d", &bt[i]);
        p[i] = i + 1;
    }

    // Sorting of burst times
    for(i = 0; i < n; i++) {
        for(j = i + 1; j < n; j++) {
            if(bt[j] < bt[i]) {
                temp = bt[i];
                bt[i] = bt[j];
                bt[j] = temp;

                temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }

    wt[0] = 0;
    for(i = 1; i < n; i++) {
        wt[i] = 0;
```

```
        for(j = 0; j < i; j++)
            wt[i] += bt[j];
        total_wt += wt[i]; // Calculate total waiting time separately
    }

    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for(i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        total_tat += tat[i]; // Calculate total turnaround time
        printf("P%d\t%d\t\t%d\t\t%d\n", p[i], bt[i], wt[i], tat[i]);
    }

    avg_wt = (float)total_wt / n; // Calculate average waiting time
    avg_tat = (float)total_tat / n; // Calculate average turnaround time

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

    return 0;
}
```

Program 3

C program to simulate producer Consumer

```
#include<stdio.h>
int main()
{
int buffer[10], bufsize, in, out, produce, consume,
choice=0; in = 0;
out = 0;
bufsize = 10;
while(choice !=3)
{
printf("\n1. Produce \t 2. Consume \t3. Exit");
printf("\nEnter your choice:");
scanf("%d",&choice);
```

```
switch(choice) {
case 1: if((in+1)%bufsize==out)
printf("\nBuffer is Full");
else
{
printf("\nEnter the value:");
scanf("%d", &produce);
buffer[in] = produce;
in = (in+1)%bufsize;
}
break;
case 2: if(in == out)
printf("\nBuffer is Empty");
else
{
consume = buffer[out];
printf("\nThe consumed value is %d", consume);
out = (out+1)%bufsize;
}
break;
return 0;
} } }
```

Experiment 4

Develop a C program which demonstrates interprocess communication between a reader process and a writer process. Use mkfifo, open, read, write and close APIs in your program.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

#define FIFO_PATH "myfifo"
```

```
void writerProcess() {
    int fd;
    char buffer[] = "Hello, Reader!";
    // Create a FIFO (named pipe)
    mkfifo(FIFO_PATH, 0666);
    // Open the FIFO for writing
    fd = open(FIFO_PATH, O_WRONLY);
    if (fd == -1) {
        perror("Error opening FIFO for writing");
        exit(EXIT_FAILURE);
    }
    // Write data to the FIFO
    write(fd, buffer, sizeof(buffer));
    // Close the FIFO
    close(fd);
    // Remove the FIFO
    unlink(FIFO_PATH);
}

void readerProcess() {
    int fd;
    char buffer[50];
    // Open the FIFO for reading
    fd = open(FIFO_PATH, O_RDONLY);
    if (fd == -1) {
        perror("Error opening FIFO for reading");
        exit(EXIT_FAILURE);
    }
    // Read data from the FIFO
    read(fd, buffer, sizeof(buffer));
    // Display the read data
    printf("Reader Process: Received message - %s\n", buffer);
    // Close the FIFO
    close(fd);
}

int main() {
    pid_t pid;
```

```
    // Fork a child process
    pid = fork();
    if (pid < 0) {
        perror("Fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Child process (writer)
        writerProcess();
    } else {
        // Parent process (reader)
        // Add a delay to ensure the writer process has created the FIFO
        sleep(1);
        readerProcess();
    }
    return 0;
}
```

5 Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance.

PROGRAM

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

int main() {
    int alloc[10][10], max[10][10];
    int avail[10], work[10], total[10];
    int i, j, k, n, need[10][10];
    int m;
    int count = 0;
    char finish[10];

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

    for (i = 0; i < n; i++) {
        finish[i] = 'n';
```

```
}

printf("Enter the maximum matrix:\n");
for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++) {
        scanf("%d", &max[i][j]);
    }
}

printf("Enter the allocation matrix:\n");
for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++) {
        scanf("%d", &alloc[i][j]);
    }
}

printf("Enter the available vector: ");
for (i = 0; i < m; i++) {
    scanf("%d", &total[i]);
}

// Initializing avail to total
for (i = 0; i < m; i++) {
    avail[i] = total[i];
}

// Subtracting alloc from avail
for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++) {
        avail[j] -= alloc[i][j];
    }
}

for (i = 0; i < m; i++) {
    work[i] = avail[i];
}

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

```
        for (j = 0; j < m; j++) {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }

A:
    for (i = 0; i < n; i++) {
        int c = 0;
        for (j = 0; j < m; j++) {
            if (need[i][j] <= work[j] && finish[i] == 'n') {
                c++;
            }
        }

        if (c == m) {
            printf("All the resources can be allocated to Process %d\n", i +
1);
            printf("Available resources are: ");
            for (k = 0; k < m; k++) {
                work[k] += alloc[i][k];
                printf("%4d", work[k]);
            }
            printf("\n");
            finish[i] = 'y';
            printf("Process %d executed?: %c \n", i + 1, finish[i]);
            count++;
        }
    }

    if (count != n) {
        goto A;
    } else {
        printf("\nSystem is in a safe state\n");
    }

    printf("\nThe given state is a safe state\n");
    return 0;
}
```

6. Develop a C program to simulate the following contiguous memory allocation Techniques: a) Worst fit b) Best fit c) First fit.

a) WORST-FIT

PROGRAM

```
#include<stdio.h>
#define max 25

int main() {
   int frag[max], b[max], f[max], i, j, nb, nf, temp;
   static int bf[max], ff[max];

   printf("\n\tMemory Management Scheme - Worst Fit\n");
   printf("Enter the number of blocks:");
   scanf("%d", &nb);
   printf("Enter the number of files:");
   scanf("%d", &nf);
   printf("\nEnter the size of the blocks:-\n");
   for (i = 0; i < nb; i++) {
      printf("Block %d:", i + 1);
      scanf("%d", &b[i]);
   }
   printf("Enter the size of the files :-\n");
   for (i = 0; i < nf; i++) {
      printf("File %d:", i + 1);
      scanf("%d", &f[i]);
   }

   for (i = 0; i < nf; i++) {
      int index = -1; // Use -1 to indicate no block has been found yet
      int maxFrag = -1; // Initialize maxFrag to -1 to find the worst fit
      for (j = 0; j < nb; j++) {
         if (bf[j] != 1) {
            temp = b[j] - f[i];
            if (temp >= 0 && temp > maxFrag) { // Check if it's a worse
```

```
fit
                maxFrag = temp;
                index = j;
            }
        }
    }
    if (index != -1) { // If a block was found
        ff[i] = index;
        frag[i] = maxFrag;
        bf[index] = 1; // Mark this block as filled
    } else {
        // If no suitable block is found, you could set ff[i] and frag[i] to
indicate failure
    }
  }

  printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment");
  for (i = 0; i < nf; i++) {
    if (ff[i] != -1) // Check if file was allocated
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i + 1, f[i], ff[i] + 1,
b[ff[i]], frag[i]);
    else
        printf("\n%d\t\t%d\t\tNot Allocated", i + 1, f[i]);
  }

  return 0;
}
```

b) BEST-FIT

PROGRAM

```
#include<stdio.h>
#define max 25

int main() {
  int frag[max], b[max], f[max], i, j, nb, nf, temp;
  static int bf[max], ff[max];

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

```
    printf("Enter the number of blocks:");
    scanf("%d", &nb);
    printf("Enter the number of files:");
    scanf("%d", &nf);
    printf("\nEnter the size of the blocks:-\n");
    for (i = 0; i < nb; i++) {
        printf("Block %d:", i + 1);
        scanf("%d", &b[i]);
    }
    printf("Enter the size of the files :-\n");
    for (i = 0; i < nf; i++) {
        printf("File %d:", i + 1);
        scanf("%d", &f[i]);
    }

    for (i = 0; i < nf; i++) {
        int index = -1; // Use -1 to indicate no suitable block has been
found yet
        int minFrag = 1e9; // Initialize minFrag to a large number to find
the best fit
        for (j = 0; j < nb; j++) {
            if (bf[j] != 1) {
                temp = b[j] - f[i];
                if (temp >= 0 && temp < minFrag) { // Check if it's a better
fit
                    minFrag = temp;
                    index = j;
                }
            }
        }
        if (index != -1) { // If a suitable block is found
            ff[i] = index;
            frag[i] = minFrag;
            bf[index] = 1; // Mark this block as filled
        } else {
            // If no suitable block is found, you might want to indicate this
differently
            // For example, setting ff[i] to -1 (or another sentinel value) to
```

```
signify allocation failure
        }
    }

    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment");
    for (i = 0; i < nf; i++) {
        if (ff[i] != -1) // Check if file was allocated
            printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i + 1, f[i], ff[i] + 1,
b[ff[i]], frag[i]);
        else
            printf("\n%d\t\t%d\t\tNot Allocated", i + 1, f[i]);
    }

    return 0;
}
```

c) FIRST-FIT

PROGRAM

```
#include<stdio.h>
#define max 25

int main() {
    int frag[max], b[max], f[max], i, j, nb, nf, temp;
    static int bf[max], ff[max];

    printf("\n\tMemory Management Scheme - First Fit");
    printf("\nEnter the number of blocks:");
    scanf("%d", &nb);
    printf("Enter the number of files:");
    scanf("%d", &nf);
    printf("\nEnter the size of the blocks:-\n");
    for (i = 0; i < nb; i++) {
        printf("Block %d:", i + 1);
        scanf("%d", &b[i]);
    }
    printf("Enter the size of the files :-\n");
```

```
    for (i = 0; i < nf; i++) {
        printf("File %d:", i + 1);
        scanf("%d", &f[i]);
    }

    for (i = 0; i < nf; i++) {
        for (j = 0; j < nb; j++) {
            if (bf[j] == 0) { // if block[j] is not allocated
                temp = b[j] - f[i];
                if (temp >= 0) { // if file fits in block
                    ff[i] = j; // allocate block j to file i
                    bf[j] = 1; // mark block as allocated
                    frag[i] = temp; // fragmentation for this allocation
                    break; // exit loop after first fit found
                }
            }
        }
    }

    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment");
    for (i = 0; i < nf; i++) {
        if (bf[ff[i]] == 1) { // If the file got a block
            printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i + 1, f[i], ff[i] + 1,
b[ff[i]], frag[i]);
        } else { // If the file didn't get a block
            printf("\n%d\t\t%d\t\tNot Allocated", i + 1, f[i]);
        }
    }

    return 0;
}
```

Program 7

FIFO Page replacement Algorithm

```
#include<stdio.h>
int fr[3];
```

```
void display() {
    int i;
    printf("\n");
    for (i = 0; i < 3; i++) {
        printf("%d\t", fr[i]);
    }
}

int main() {
    int i, j, page[12] = {2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2};
    int flag1 = 0, flag2 = 0, pf = 0, frsize = 3, top = 0;

    for (i = 0; i < 3; i++) {
        fr[i] = -1;
    }

    for (j = 0; j < 12; j++) {
        flag1 = 0;
        flag2 = 0;

        for (i = 0; i < frsize; i++) {
            if (fr[i] == page[j]) {
                flag1 = 1;
                flag2 = 1;
                break;
            }
        }

        if (flag1 == 0) {
            for (i = 0; i < frsize; i++) {
                if (fr[i] == -1) {
                    fr[i] = page[j];
                    flag2 = 1;
                    pf++; // Increment page fault count for initial fills
                    break;
                }
            }
        }
```

```
        if (flag2 == 0) {
            fr[top] = page[j];
            top++;
            pf++; // Increment page fault count for replacements
            if (top >= frsize) {
                top = 0;
            }
        }
        display();
    }

    printf("\nNumber of page faults : %d", pf);
    return 0;
}
```

LRU Page replacemnt Algorithm

```
#include<stdio.h>

int fr[3];

void display() {
    int i;
    for (i = 0; i < 3; i++) {
        printf("\t%d", fr[i]);
    }
    printf("\n");
}

int findLRU(int time[], int n){
    int i, minimum = time[0], pos = 0;
    for(i = 1; i < n; ++i){
        if(time[i] < minimum){
            minimum = time[i];
            pos = i;
        }
    }
```

```
    return pos;
}

int main() {
    int p[12] = {2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2}, i, j;
    int fs[3], index, pf = 0, frsize = 3, counter = 0;
    int time[3];

    for(i = 0; i < frsize; i++) {
        fr[i] = -1;
    }

    for(j = 0; j < 12; j++) {
        int flag1 = 0, flag2 = 0;
        for(i = 0; i < frsize; i++) {
            if(fr[i] == p[j]) {
                counter++;
                time[i] = counter;
                flag1 = flag2 = 1;
                break;
            }
        }
        if(flag1 == 0) {
            for(i = 0; i < frsize; i++) {
                if(fr[i] == -1) {
                    counter++;
                    fr[i] = p[j];
                    time[i] = counter;
                    flag2 = 1;
                    pf++;
                    break;
                }
            }
        }
        if(flag2 == 0) {
            int pos = findLRU(time, frsize);
            counter++;
            fr[pos] = p[j];
```

```
            time[pos] = counter;
            pf++;
        }
        display();
    }

    printf("\nNumber of page faults: %d\n", pf);
    return 0;
}
```

Program 8

a) SINGLE LEVEL DIRECTORY FILE ORGANIZATION TECHNIQUE

```
#include <stdio.h>
#include <string.h> // For strcmp and strcpy
#include <stdlib.h> // For exit

// Define a structure for the directory
struct
{
    char dname[10];
    char fname[10][10];
    int fcnt;
} dir;

int main() // Changed return type to int for standard compliance
{
    int i, ch;
    char f[30];

    dir.fcnt = 0;
    printf("\nEnter name of directory -- ");
    scanf("%s", dir.dname);

    while (1)
    {
        printf("\n\n1. Create File\t2. Delete File\t3. Search File \n4.
```

```
Display Files\t5. Exit\nEnter your choice -- ");
        scanf("%d", &ch);

        switch (ch)
        {
            case 1:
                if(dir.fcnt < 10) { // Check if the directory is not full
                    printf("\nEnter the name of the file -- ");
                    scanf("%s", dir.fname[dir.fcnt]);
                    dir.fcnt++;
                } else {
                    printf("Directory is full, cannot add more files.\n");
                }
                break;
            case 2:
                printf("\nEnter the name of the file -- ");
                scanf("%s", f);
                for (i = 0; i < dir.fcnt; i++)
                {
                    if (strcmp(f, dir.fname[i]) == 0)
                    {
                        printf("File %s is deleted ", f);
                        strcpy(dir.fname[i], dir.fname[dir.fcnt - 1]);
                        dir.fcnt--;
                        break;
                    }
                }
                if (i == dir.fcnt)
                    printf("File %s not found", f);
                break;
            case 3:
                printf("\nEnter the name of the file -- ");
                scanf("%s", f);
                for (i = 0; i < dir.fcnt; i++)
                {
                    if (strcmp(f, dir.fname[i]) == 0)
                    {
                        printf("File %s is found ", f);
```

```
                    break;
                }
            }
            if (i == dir.fcnt)
                printf("File %s not found", f);
            break;
        case 4:
            if (dir.fcnt == 0)
                printf("\nDirectory Empty");
            else
            {
                printf("\nThe Files are -- ");
                for (i = 0; i < dir.fcnt; i++)
                    printf("\t%s", dir.fname[i]);
            }
            break;
        case 5:
            exit(0); // Correct use to exit the program
        default:
            printf("Invalid choice. Please enter a valid option.\n");
      }
   }
   // Removed getch(); as it's not standard C and not necessary here
   return 0; // Added return statement for compliance with 'int main'
}
```

b) TWO LEVEL DIRECTORY

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

struct {
   char dname[10];
   char fname[10][10];
   int fcnt;
} dir[10];

int main() {
```

```
   int i, ch, dcnt = 0, k;
   char f[30], d[30];

   while(1) {
      printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");
      printf("\n4. Search File\t\t5. Display\t6. Exit\nEnter your choice --
");
      scanf("%d", &ch);

      switch(ch) {
         case 1:
            if(dcnt < 10) { // Check if there is room for a new directory
               printf("\nEnter name of directory -- ");
               scanf("%s", dir[dcnt].dname);
               dir[dcnt].fcnt = 0;
               dcnt++;
               printf("Directory created");
            } else {
               printf("Maximum directory limit reached.");
            }
            break;
         case 2:
            printf("\nEnter name of the directory -- ");
            scanf("%s", d);
            for(i = 0; i < dcnt; i++) {
               if(strcmp(d, dir[i].dname) == 0) {
                  if(dir[i].fcnt < 10) { // Check if there is room for a new
file
                     printf("Enter name of the file -- ");
                     scanf("%s", dir[i].fname[dir[i].fcnt]);
                     dir[i].fcnt++;
                     printf("File created");
                  } else {
                     printf("Maximum file limit in directory reached.");
                  }
                  break; // Exit the loop once the directory is found and
file is added
               }
```

```
            }
            if(i == dcnt)
                printf("Directory %s not found", d);
            break;
        case 3:
            printf("\nEnter name of the directory -- ");
            scanf("%s", d);
            for(i = 0; i < dcnt; i++) {
                if(strcmp(d, dir[i].dname) == 0) {
                    printf("Enter name of the file -- ");
                    scanf("%s", f);
                    for(k = 0; k < dir[i].fcnt; k++) {
                        if(strcmp(f, dir[i].fname[k]) == 0) {
                            printf("File %s is deleted ", f);
                            dir[i].fcnt--;
                            strcpy(dir[i].fname[k], dir[i].fname[dir[i].fcnt]);
                            break; // Exit the loop once the file is found and
deleted
                        }
                    }
                    if(k == dir[i].fcnt)
                        printf("File %s not found", f);
                    break; // Exit the loop once the directory is found
                }
            }
            if(i == dcnt)
                printf("Directory %s not found", d);
            break;
        case 4:
            printf("\nEnter name of the directory -- ");
            scanf("%s", d);
            for(i = 0; i < dcnt; i++) {
                if(strcmp(d, dir[i].dname) == 0) {
                    printf("Enter the name of the file -- ");
                    scanf("%s", f);
                    for(k = 0; k < dir[i].fcnt; k++) {
                        if(strcmp(f, dir[i].fname[k]) == 0) {
                            printf("File %s is found ", f);
```

```
                    break; // Exit the loop once the file is found
                }
            }
            if(k == dir[i].fcnt)
                printf("File %s not found", f);
            break; // Exit the loop once the directory is found
        }
    }
    if(i == dcnt)
        printf("Directory %s not found", d);
    break;
case 5:
    if(dcnt == 0)
        printf("\nNo Directories");
    else {
        printf("\nDirectory\tFiles");
        for(i = 0; i < dcnt; i++) {
            printf("\n%s\t\t", dir[i].dname);
            for(k = 0; k < dir[i].fcnt; k++)
                printf("\t%s", dir[i].fname[k]);
        }
    }
    break;
default:
    exit(0);
}
}

return 0; // Correct program termination with a return statement
}
```