OS Practicals

System Programming and Operating System (Savitribai Phule Pune University)

1. Implement the C Program to create a child process using fork(), display parent and child process id. Child process will display the message “I am Child Process” and the parent process should display “I am Parent Process”.

```c
#include <stdio.h>
void childprocess()
{
    printf("\n I am Child Process");
}
void parentprocess()
{
    printf("\n I am Parent Process ");
}
main()
{
    int pid;
    pid = fork();
    if (pid == 0)
    {
        printf("Child process id=%d", pid);
        childprocess();
    }
    else
    {
        printf("\n Present Process ID=%d", pid);
        parentprocess();
    }
}
```

2. Write a program that demonstrates the use of nice() system call. After a child process is started using fork(), assign higher priority to the child using nice() system call.

```c
#include <stdio.h>
main()
{
    int pid, retnice, i;
    pid = fork();
    for (i = 0; i < 3; i++)
    {
        if (pid == 0)
        {
            retnice = nice(-5);
            printf("Child gets higher CPU priority%d\n", retnice);
            sleep(1);
        }
        else
        {
            retnice = nice(4);
            printf("Parent gets lower CPU Priority%d\n", retnice);
            sleep(1);
        }
    }
}
```

3. Implement the C program to accept n integers to be sorted. Main function creates child process using fork system call. Parent process sorts the integers using bubble sort and waits for child process using wait system call. Child process sorts the integers using insertion sort.

```c
#include <stdio.h>
int n;
void display(int a[20])
{
    int i;
    printf("\n Array Element:");
    for (i = 0; i < n; i++)
        printf("\t%d", a[i]);
}
void insertionsort(int a[20])
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = a[i];
        for (j = i - 1; j >= 0 && a[j] > key; j--)
        {
            a[j + 1] = a[j];
        }
        a[j + 1] = key;
    }
}
void bubblesort(int a[20])
{
    int pass, i, temp;
    for (pass = 1; pass < n; pass++)
    {
        for (i = 0; i < n - pass; i++)
        {
            if (a[i] > a[i + 1])
            {
                temp = a[i];
                a[i] = a[i + 1];
                a[i + 1] = temp;
            }
        }
    }
}
main()
{
    int i, a[20], pid, Barr[20], Iarr[20];
    printf("\n Enter value of n");
    scanf("%d", &n);
    printf("\n Enter arrray Elements");
    for (i = 0; i < n; i++)
    {
        printf("\n Enter Element:", i);
        scanf("%d", &a[i]);
        Barr[i] = a[i];
        Iarr[i] = a[i];
    }
    display(a);
    pid = fork();
    if (pid == 0)
    {
        printf("\n Child Process id=%d", getpid());
        insertionsort(Iarr);
        printf("\n Insertion Sort");
        display(Iarr);
    }
    else if (pid > 0)
    {
        wait(NULL);
        sleep(5);
        printf("\n Parent Process id=%d", getpid());
        bubblesort(Barr);
        printf("\nBubbleSort");
        display(Barr);
    }
}
```

4. Write a C program to illustrate the concept of orphan process. Parent process creates a child and terminates before child has finished its task. So child process becomes orphan process. (Use fork(), sleep(), getpid(), getppid()).

```c
#include <stdio.h>
main()
{
    int pid, a;
    printf("\n the process id is%d", getpid());
    pid = fork();
    if (pid < 0)
        printf("\n Fork failed!");
    else if (pid == 0)
    {
        printf("\n I am Child process");
        printf("\n Child process ID=%d", getpid());
    }
    else
    {
        wait(NULL);
        sleep(5);
        printf("\n I am Parent process");
        printf("\n Child Parent ID=%d", getpid());
    }
}
```

5. Write a C program that behaves like a shell which displays the command prompt ‘myshell$’. It accepts the command, tokenize the command line and execute it by creating the child process. Also implement the additional command ‘count’ as
   myshell$ count c filename: It will display the number of characters in given file
   myshell$ count w filename: It will display the number of words in given file
   myshell$ count l filename: It will display the number of lines in given file

```
#include <stdio.h>
#include<stdlib.h>
char *buff, *t1, *t2, *t3, ch;
FILE *fp;
int pid;
void count(char *t2, char *t3)
{
    int charcount = 0, wordcount = 0, linecount = 0;
    fp = fopen(t3, "r");
    if (fp == NULL)
        printf("\nError in opening the file");
    else
    {
        while ((ch = fgetc(fp)) != EOF)
        {
            if (ch == ' ')
                wordcount++;
            else if (ch == '\n')
            {
                linecount++;
                wordcount++;
            }
            else
                charcount++;
        }
```

```c
    }
    fclose(fp);
    if (strcmp(t2, "c") == 0)
        printf("\nTotal no of characters: %d", charcount);
    else if (strcmp(t2, "w") == 0)
        printf("\nTotal no of Words: %d", wordcount);
    else if (strcmp(t2, "l") == 0)
        printf("\nTotal no of lines: %d", linecount);
    else
        printf("Command not found");
}
main()
{
    while (1)
    {
        printf("\nMyshell$");
        buff = (char *)malloc(80);
        t1 = (char *)malloc(80);
        t2 = (char *)malloc(80);
        t3 = (char *)malloc(80);
        fgets(buff, 80, stdin);
        sscanf(buff, "%s %s %s", t1, t2, t3);
        if (strcmp(t1, "pause") == 0)
            exit(0);
        if (strcmp(t1, "count") == 0)
            count(t2, t3);
        else
        {
            pid = fork();
            if (pid < 0)
                printf("\nchild Process is not craeted\n");
            else if (pid == 0)
            {
                execlp("/bin", NULL);
                system(buff);
            }
            else
            {
                wait(NULL);
                exit(0);
            }
        }
    }
}
```

6. Write a C program that behaves like a shell which displays the command prompt ‘myshell$’. It accepts the command, tokenize the command line and execute it by creating the child process. Also implement the additional command ‘list’ as
myshell$ list f dirname: It will display filenames in a given directory.
myshell$ list n dirname: It will count the number of entries in a given directory.
myshell$ list i dirname: It will display filenames and their inode number for the files in a given directory

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <dirent.h>
char *buff, *t1, *t2, *t3, ch;
int pid;
void list(char t2, char *t3)
{
    DIR *dir;
    struct dirent *entry;
    int cnt = 0;
    dir = opendir(t3);
    if (dir == NULL)
    {
        printf("Directory %s not found", t3);
        return;
    }
    switch (t2)
    {
    case 'f':
        while ((entry = readdir(dir)) != NULL)
        {
            printf("%s\n", entry->d_name);
        }
        break;
    case 'n':
        while ((entry = readdir(dir)) != NULL)
            cnt++;
        printf("Total No of Entries: %d\n", cnt);
        break;
    case 'i':
        while ((entry = readdir(dir)) != NULL)
        {
            printf("\n%s\t %d", entry->d_name, entry->d_ino);
        }
        break;
    default:
        printf("Invalid argument");
    }
    closedir(dir);
}
```

```
main()
{
    while (1)
    {
        printf("myshell$");
        fflush(stdin);
        t1 = (char *)malloc(80);
        t2 = (char *)malloc(80);
        t3 = (char *)malloc(80);
        buff = (char *)malloc(80);
        fgets(buff, 80, stdin);
        sscanf(buff, "%s %s %s", t1, t2, t3);
        if (strcmp(t1, "pause") == 0)
            exit(0);
        else if (strcmp(t1, "list") == 0)
            list(t2[0], t3);
        else
        {
            pid = fork();
            if (pid < 0)
                printf("Child process is not created\n");
            else if (pid == 0)
            {
                execlp("/bin", NULL);
                if (strcmp(t1, "exit") == 0)
                    exit(0);
                system(buff);
            }
            else
            {
                wait(NULL);
                exit(0);
            }
        }
    }
}
```

7. Write a C program that behaves like a shell which displays the command prompt 'myshell$'. It accepts the command, tokenize the command line and execute it by creating the child process. Also implement the additional command 'typeline' as
myshell$ typeline n filename: It will display first n lines of the file.
myshell$ typeline -n filename: It will display last n lines of the file.
myshell$ typeline a filename: It will display all the lines of the file.

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

```c
#include <string.h>
char *buff, *t1, *t2, *t3, ch;
FILE *fp;
int pid;
void typeline(char *t2, char *t3)
{
    int i, n, count = 0, num;
    if ((fp = fopen(t3, "r")) == NULL)
        printf("File not found\n");
    if (strcmp(t2, "a") == 0)
    {
        while ((ch = fgetc(fp)) != EOF)
            printf("%c", ch);
        fclose(fp);
        return;
    }
    n = atoi(t2);
    if (n > 0)
    {
        i = 0;
        while ((ch = fgetc(fp)) != EOF)
        {
            if (ch == '\n')
                i++;
            if (i == n)
                break;
            printf("%c", ch);
        }
        printf("\n");
    }
    else
    {
        count = 0;
        while ((ch = fgetc(fp)) != EOF)
            if (ch == '\n')
                count++;
        fseek(fp, 0, SEEK_SET);
        i = 0;
        while ((ch = fgetc(fp)) != EOF)
        {
            if (ch == '\n')
                i++;
            if (i == count + n - 1)
                break;
        }
        while ((ch = fgetc(fp)) != EOF)
            printf("%c", ch);
    }
    fclose(fp);
}
main()
{
    while (1)
    {
        printf("myshell$");
        fflush(stdin);
        t1 = (char *)malloc(80);
        t2 = (char *)malloc(80);
        t3 = (char *)malloc(80);
        buff = (char *)malloc(80);
        fgets(buff, 80, stdin);
        sscanf(buff, "%s %s %s", t1, t2, t3);
        if (strcmp(t1, "pause") == 0)
            exit(0);
        else if (strcmp(t1, "typeline") == 0)
            typeline(t2, t3);
        else
        {
            pid = fork();
            if (pid < 0)
                printf("Child process is not created\n");
            else if (pid == 0)
            {
                execlp("/bin", NULL);
                if (strcmp(t1, "exit") == 0)
                    exit(0);
                system(buff);
            }
            else
            {
                wait(NULL);
                exit(0);
            }
        }
    }
}
```

8. Write a C program that behaves like a shell which displays the command prompt ‘myshell$’. It accepts the command, tokenize the command line and execute it by creating the child process. Also implement the additional command ‘search’ as
myshell$ search f filename pattern : It will search the first occurrence of pattern in the given file
myshell$ search a filename pattern : It will search all the occurrence of pattern in the given file
myshell$ search c filename pattern : It will count the number of occurrence of pattern in the given file

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
char *buff, *t1, *t2, *t3, *t4, ch;
FILE *fp;
int pid;
void search(char *t2, char *t3, char *t4)
{
    int i = 1, count = 0;
    char *p;
    if ((fp = fopen(t3, "r")) == NULL)
        printf("File not found\n");
    else
    {
        if (strcmp(t2, "f") == 0)
        {
            while (fgets(buff, 80, fp))
            {
                if ((strstr(buff, t4)) != NULL)
                {
                    printf("%d: %s\n", i, buff);
                    break;
                }
            }
            i++;
        }
        else if (strcmp(t2, "c") == 0)
        {
            while (fgets(buff, 80, fp))
            {
                if ((strstr(buff, t4)) != NULL)
                {
                    count++;
                }
            }
            printf("No of occurences of %s= %d\n", t3, count);
        }
        else if (strcmp(t2, "a") == 0)
        {
            while (fgets(buff, 80, fp))
            {
                if ((strstr(buff, t4)) != NULL)
                {
                    printf("%d: %s\n", i, buff);
                }
                i++;
            }
        }
        else
            printf("Command not found\n");
        fclose(fp);
    }
}
main()
{
    while (1)
    {
        printf("myshell$");
        fflush(stdin);
        t1 = (char *)malloc(80);
        t2 = (char *)malloc(80);
        t3 = (char *)malloc(80);
        t4 = (char *)malloc(80);
        buff = (char *)malloc(80);
        fgets(buff, 80, stdin);
        sscanf(buff, "%s %s %s %s", t1, t2, t3, t4);
        if (strcmp(t1, "pause") == 0)
            exit(0);
        else if (strcmp(t1, "search") == 0)
            search(t2, t3, t4);
        else
        {
            pid = fork();
            if (pid < 0)
                printf("Child process is not created\n");
            else if (pid == 0)
            {
                execlp("/bin", NULL);
                if (strcmp(t1, "exit") == 0)
                    exit(0);
                system(buff);
            }
            else
            {
                wait(NULL);
                exit(0);
            }
        }
    }
}
```

9. Write the program to simulate FCFS CPU-scheduling. The arrival time and first CPUburst for different n number of processes should be input to the algorithm. Assume that the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

```c
#include <stdio.h>
struct input
{
    char pname[10];
    int bt, at, tbt, ft;
} tab[10];
struct gantt
{
    int start, end;
    char pname[10];
} g[50], g1[10];
int n, i, k, time, prev;
void getinput()
{
    printf("\nEnter No of Processes: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("\nProcess Name: ");
        scanf("%s", tab[i].pname);
        printf("Burst Time: ");
        scanf("%d", &tab[i].bt);
        tab[i].tbt = tab[i].bt;
        printf("Arrival Time:");
        scanf("%d", &tab[i].at);
    }
}
void printinput()
{
    printf("\nPname\tBT\tAT");
    for (i = 0; i < n; i++)
        printf("\n%s\t%d\t%d", tab[i].pname, tab[i].tbt, tab[i].at);
}
void sort()
{
    struct input temp;
    int j;
    for (i = 1; i < n; i++)         // pass
        for (j = 0; j < n - 1; j++) // Comp
            if (tab[j].at > tab[j + 1].at)
            {
                temp = tab[j];
                tab[j] = tab[j + 1];
                tab[j + 1] = temp;
            }
}
int arrived(int time)
{
    for (i = 0; i < n; i++)
        if (tab[i].at <= time && tab[i].tbt != 0)
            return 1;
    return 0;
}
void processinput()
{
    int j, finish = 0;
    // time=tab[0].at;
    while (finish != n)
    {
        if (arrived(time))
        {
            for (j = 0; j < tab[i].bt; j++)
            {
                time++;
                tab[i].tbt--;
                g[k].start = prev;
                g[k].end = time;
                // printinput();
                prev = time;
                tab[i].ft = time;
                strcpy(g[k++].pname, tab[i].pname);
                if (tab[i].tbt == 0)
                {
                    finish++;
                    break;
                }
            }
        }
        else
        {
            time++;
            g[k].start = prev;
            g[k].end = time;
            prev = time;
            strcpy(g[k++].pname, "idle");
        }
    }
    // printinput();
}
void printoutput()
{
```

```c
    int TTAT = 0, TWT = 0;
    float ATAT, AWT;
    printf("\n******Final Table*****");
    printf("\nPname\tAT\tBT\tFT\tTAT\tWT");
    for (i = 0; i < n; i++)
    {
        printf("\n%s\t%d\t%d\t%d\t%d\t%d", tab[i].pname, tab[i].at, tab[i].bt,
tab[i].ft, tab[i].ft-tab[i].at, tab[i].ft - tab[i].at - tab[i].bt);
        TTAT = TTAT + (tab[i].ft - tab[i].at);
        TWT = TWT + (tab[i].ft - tab[i].at - tab[i].bt);
    }
    ATAT = (float)TTAT / n;
    AWT = (float)TWT / n;
    printf("\nTotal TAT=%d", TTAT);
    printf("\nTotal WT=%d", TWT);
    printf("\nAverage TAT=%f", ATAT);
    printf("\nAverage WT=%f", AWT);
}
void printganttchart()
{
    int j = 0;
    g1[0] = g[0];
    for (i = 1; i < k; i++)
    {
        if (strcmp(g1[j].pname, g[i].pname) == 0)
            g1[j].end = g[i].end;
        else
        {
            j++;
            g1[j] = g[i];
        }
    }
    printf("\n******Each unit Gantt chart******");
    for (i = 0; i < k; i++)
        printf("\n%d\t%s\t%d", g[i].start, g[i].pname, g[i].end);
    printf("\n******Final Gantt chart******");
    for (i = 0; i <= j; i++)
        printf("\n%d\t%s\t%d", g1[i].start, g1[i].pname, g1[i].end);
}
int main()
{
    getinput();
    printf("\nEntered data is: ");
    printinput();
    sort();
    printf("\nData after Sorting");
    printinput();
    processinput();
    printoutput();
    printganttchart();
}
```

10. Write the program to simulate Non-preemptive Shortest Job First (SJF) -scheduling. The arrival time and first CPU-burst for different n number of processes should be input to the algorithm. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

```
#include <stdio.h>
struct input
{
    char pname[10];
    int bt, at, tbt, ft;
} tab[10];
struct gantt
{
    int start, end;
    char pname[10];
} g[50], g1[10];
int n, i, k, time, prev;
void getinput()
{
    printf("\nEnter No of Processes: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("\nProcess Name: ");
        scanf("%s", tab[i].pname);
        printf("Burst Time: ");
        scanf("%d", &tab[i].bt);
        tab[i].tbt = tab[i].bt;
        printf("Arrival Time:");
        scanf("%d", &tab[i].at);
    }
}
void printinput()
{
    printf("\nPname\tBT\tAT");
    for (i = 0; i < n; i++)
        printf("\n%s\t%d\t%d", tab[i].pname, tab[i].tbt, tab[i].at);
}
void sort()
{
    struct input temp;
    int j;
    for (i = 1; i < n; i++)         // pass
```

```
        for (j = 0; j < n - 1; j++) // Comp
            if (tab[j].at > tab[j + 1].at)
            {
                temp = tab[j];
                tab[j] = tab[j + 1];
                tab[j + 1] = temp;
            }
}
int arrived(int time)
{
    for (i = 0; i < n; i++)
        if (tab[i].at <= time && tab[i].tbt != 0)
            return 1;
    return 0;
}
int getsmallburst(int time)
{
    int min = 99, mini;
    for (i = 0; i < n; i++)
    {
        if (tab[i].tbt < min && tab[i].at <= time && tab[i].tbt != 0)
        {
            min = tab[i].tbt;
            mini = i;
        }
    }
    return mini;
}
void processinput()
{
    int j, finish = 0;
    // time=tab[0].at;
    while (finish != n)
    {
        if (arrived(time))
        {
            i = getsmallburst(time);
            for (j = 0; j < tab[i].bt; j++)
            {
                time++;
                tab[i].tbt--;
                g[k].start = prev;
                g[k].end = time;
                // printinput();
                prev = time;
                tab[i].ft = time;
                strcpy(g[k++].pname, tab[i].pname);
                if (tab[i].tbt == 0)
                {
                    finish++;
                    break;
                }
            }
        }
        else
        {
            time++;
            g[k].start = prev;
            g[k].end = time;
            prev = time;
            strcpy(g[k++].pname, "idle");
        }
    }
    // printinput();
}
void printoutput()
{
    int TTAT = 0, TWT = 0;
    float ATAT, AWT;
    printf("\n******Final Table*****");
    printf("\nPname\tAT\tBT\tFT\tTAT\tWT");
    for (i = 0; i < n; i++)
    {
        printf("\n%s\t%d\t%d\t%d\t%d\t%d", tab[i].pname, tab[i].at, tab[i].bt,
tab[i].ft, tab[i].ft - tab[i].at, tab[i].ft - tab[i].at - tab[i].bt);
        TTAT = TTAT + (tab[i].ft - tab[i].at);
        TWT = TWT + (tab[i].ft - tab[i].at - tab[i].bt);
    }
    ATAT = (float)TTAT / n;
    AWT = (float)TWT / n;
    printf("\nTotal TAT=%d", TTAT);
    printf("\nTotal WT=%d", TWT);
    printf("\nAverage TAT=%f", ATAT);
    printf("\nAverage WT=%f", AWT);
}
void printganttchart()
{
    int j = 0;
    g1[0] = g[0];
    for (i = 1; i < k; i++)
    {
        if (strcmp(g1[j].pname, g[i].pname) == 0)
            g1[j].end = g[i].end;
        else
        {
            j++;
            g1[j] = g[i];
        }
    }
    printf("\n******Each unit Gantt chart******");
    for (i = 0; i < k; i++)
        printf("\n%d\t%s\t%d", g[i].start, g[i].pname, g[i].end);
    printf("\n******Final Gantt chart******");
    for (i = 0; i <= j; i++)
        printf("\n%d\t%s\t%d", g1[i].start, g1[i].pname, g1[i].end);
}
int main()
{
    getinput();
    printf("\nEntered data is: ");
    printinput();
    sort();
    printf("\nData after Sorting");
    printinput();
    processinput();
    printoutput();
    printganttchart();
}
```

11. Write the program to simulate Preemptive Shortest Job First (SJF) -scheduling. The arrival time and first CPU-burst for different n number of processes should be input to the algorithm. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

```
#include <stdio.h>
struct input
{
    char pname[10];
    int bt, at, tbt, ft;
} tab[10];
struct gantt
{
    int start, end;
    char pname[10];
} g[50], g1[10];
int n, i, k, time, prev;
void getinput()
{
    printf("\nEnter No of Processes: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("\nProcess Name: ");
```

```c
        scanf("%s", tab[i].pname);
        printf("Burst Time: ");
        scanf("%d", &tab[i].bt);
        tab[i].tbt = tab[i].bt;
        printf("Arrival Time:");
        scanf("%d", &tab[i].at);
    }
}
void printinput()
{
    printf("\nPname\tBT\tAT");
    for (i = 0; i < n; i++)
        printf("\n%s\t%d\t%d", tab[i].pname, tab[i].tbt, tab[i].at);
}
void sort()
{
    struct input temp;
    int j;
    for (i = 1; i < n; i++)         // pass
        for (j = 0; j < n - 1; j++) // Comp
            if (tab[j].at > tab[j + 1].at)
            {
                temp = tab[j];
                tab[j] = tab[j + 1];
                tab[j + 1] = temp;
            }
}
int arrived(int time)
{
    for (i = 0; i < n; i++)
        if (tab[i].at <= time && tab[i].tbt != 0)
            return 1;
    return 0;
}
int getsmallburst(int time)
{
    int min = 99, mini;
    for (i = 0; i < n; i++)
    {
        if (tab[i].tbt < min && tab[i].at <= time && tab[i].tbt != 0)
        {
            min = tab[i].tbt;
            mini = i;
        }
    }
    return mini;
}
void processinput()
```

```
{
    int j, finish = 0;
    // time=tab[0].at;
    while (finish != n)
    {
        if (arrived(time))
        {
            i = getsmallburst(time);
            time++;
            tab[i].tbt--;
            g[k].start = prev;
            g[k].end = time;
            // printinput();
            prev = time;
            tab[i].ft = time;
            strcpy(g[k++].pname, tab[i].pname);
            if (tab[i].tbt == 0)
            {
                finish++;
            }
        }
        else
        {
            time++;
            g[k].start = prev;
            g[k].end = time;
            prev = time;
            strcpy(g[k++].pname, "idle");
        }
    }
    // printinput();
}
void printoutput()
{
    int TTAT = 0, TWT = 0;
    float ATAT, AWT;
    printf("\n******Final Table*****");
    printf("\nPname\tAT\tBT\tFT\tTAT\tWT");
    for (i = 0; i < n; i++)
    {
        printf("\n%s\t%d\t%d\t%d\t%d\t%d", tab[i].pname, tab[i].at, tab[i].bt,
tab[i].ft, tab[i].ft - tab[i].at, tab[i].ft - tab[i].at - tab[i].bt);
        TTAT = TTAT + (tab[i].ft - tab[i].at);
        TWT = TWT + (tab[i].ft - tab[i].at - tab[i].bt);
    }
    ATAT = (float)TTAT / n;
    AWT = (float)TWT / n;
    printf("\nTotal TAT=%d", TTAT);
```

```c
    printf("\nTotal WT=%d", TWT);
    printf("\nAverage TAT=%f", ATAT);
    printf("\nAverage WT=%f", AWT);
}
void printganttchart()
{
    int j = 0;
    g1[0] = g[0];
    for (i = 1; i < k; i++)
    {
        if (strcmp(g1[j].pname, g[i].pname) == 0)
            g1[j].end = g[i].end;
        else
        {
            j++;
            g1[j] = g[i];
        }
    }
    printf("\n******Each unit Gantt chart******");
    for (i = 0; i < k; i++)
        printf("\n%d\t%s\t%d", g[i].start, g[i].pname, g[i].end);
    printf("\n******Final Gantt chart******");
    for (i = 0; i <= j; i++)
        printf("\n%d\t%s\t%d", g1[i].start, g1[i].pname, g1[i].end);
}
int main()
{
    getinput();
    printf("\nEntered data is: ");
    printinput();
    sort();
    printf("\nData after Sorting");
    printinput();
    processinput();
    printoutput();
    printganttchart();
}
```

12. Write the program to simulate Non-preemptive Priority scheduling. The arrival time and first CPU-burst and priority for different n number of processes should be input to the algorithm. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

```c
#include <stdio.h>
struct input
{
    char pname[10];
    int bt, at, tbt, ft, p;
```

```c
} tab[10];
struct gantt
{
    char pname[10];
    int start, end;
} g[30], g1[30];
int n, i, time, prev, k;
void getinput()
{
    printf("\nEnter No of Process: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("\nEnter Process Name: ");
        scanf("%s", tab[i].pname);
        printf("Arrival Time:");
        scanf("%d", &tab[i].at);
        printf("Burst Time: ");
        scanf("%d", &tab[i].bt);
        tab[i].tbt = tab[i].bt;
        printf("\nEnter the Priority:");
        scanf("%d", &tab[i].p);
    }
}
void printinput()
{
    // int TWT=0,TTAT=0;
    printf("\nPname\tAT\tBT");
    for (i = 0; i < n; i++)
        printf("\n%s\t%d\t%d", tab[i].pname, tab[i].at, tab[i].bt);
}
void printoutput()
{
    int TWT = 0, TTAT = 0;
    float ATAT, AWT;
    printf("\nPname\tAT\tBT\tFT\tWT\tTAT");
    for (i = 0; i < n; i++)
    {
        printf("\n%s\t%d\t%d\t%d\t%d\t%d", tab[i].pname, tab[i].at, tab[i].bt,
tab[i].ft, tab[i].ft-tab[i].at - tab[i].bt, tab[i].ft - tab[i].at);
        TWT = TWT + (tab[i].ft - tab[i].at - tab[i].bt);
        TTAT = TTAT + (tab[i].ft - tab[i].at);
    }
    printf("\nTotal WT: %d", TWT);
    printf("\nTotal TAT:%d", TTAT);
    AWT = (float)TWT / n;
    ATAT = (float)TTAT / n;
    printf("\nAverage WT: %f", AWT);
    printf("\nAverage TAT:%f", ATAT);
}
void sort()
{
    int pass;
    struct input temp;
    for (pass = 1; pass < n; pass++)
    {
        for (i = 0; i < n - pass; i++)
        {
            if (tab[i].at > tab[i + 1].at)
            {
                temp = tab[i];
                tab[i] = tab[i + 1];
                tab[i + 1] = temp;
            }
        }
    }
}
int arrived(int time)
{
    for (i = 0; i < n; i++)
    {
        if (tab[i].at <= time && tab[i].tbt != 0)
            return 1;
    }
    return 0;
}
int gethighpriority(int time)
{
    int processpos, min = 99;
    for (i = 0; i < n; i++) // i=0,1
    {                        // p1,p3 min=2
        if (tab[i].at <= time && tab[i].tbt != 0 && tab[i].p < min)
        {
            min = tab[i].p;
            processpos = i;
        }
    }
    return processpos;
}
void processinput()
{
    int finish = 0, j;
    k = 0;
    while (finish != n)
    {
        if (arrived(time))
        {
            i = gethighpriority(time);
            for (j = 0; j < tab[i].bt; j++)
            {
                time++;
                tab[i].tbt--;
                g[k].start = prev;
                g[k].end = time;
                prev = time;
                strcpy(g[k++].pname, tab[i].pname);
                tab[i].ft = time;
                if (tab[i].tbt == 0)
                {
                    finish++;
                    break;
                }
            }
        }
        else
        {
            time++;
            g[k].start = prev;
            g[k].end = time;
            strcpy(g[k++].pname, "idle");
            prev = time;
        }
        // i++;
    }
}
void ganttchart()
{
    int i, j = 0;
    printf("\n******Each Unit Gantt chart******");
    printf("\nStart\tpname\tEnd");
    for (i = 0; i < k; i++)
    {
        printf("\n%d\t%s\t%d", g[i].start, g[i].pname, g[i].end);
    }
    printf("\n********Final Gantt Chart*******");
    g1[0] = g[0];
    for (i = 1; i < k; i++)
    {
        if (strcmp(g[i].pname, g1[j].pname) == 0)
            g1[j].end = g[i].end;
        else
        {
            j++;
            g1[j] = g[i];
```

```c
        }
    }
    printf("\nStart\tpname\tEnd");
    for (i = 0; i <= j; i++)
    {
        printf("\n%d\t%s\t%d", g1[i].start, g1[i].pname, g1[i].end);
    }
}
int main()
{
    getinput();
    printinput();
    sort();
    printf("\nData After Sorting: ");
    printinput();
    processinput();
    printoutput();
    ganttchart();
    for (i = 0; i < n; i++)
    {
        tab[i].tbt = tab[i].bt = rand() % 10 + 1;
        tab[i].at = tab[i].ft + 2;
    }
    printinput();
    processinput();
    printoutput();
    ganttchart();
}
```

13. Write the program to simulate Preemptive Priority scheduling. The arrival time and first CPU-burst and priority for different n number of processes should be input to the algorithm. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

```c
#include <stdio.h>
struct input
{
    char pname[10];
    int bt, at, tbt, ft, p;
} tab[10];
struct gantt
{
    char pname[10];
    int start, end;
} g[30], g1[30];
int n, i, time, prev, k;
void getinput()
```

```
{
    printf("\nEnter No of Process: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("\nEnter Process Name: ");
        scanf("%s", tab[i].pname);
        printf("Arrival Time:");
        scanf("%d", &tab[i].at);
        printf("Burst Time: ");
        scanf("%d", &tab[i].bt);
        tab[i].tbt = tab[i].bt;
        printf("\nEnter the Priority:");
        scanf("%d", &tab[i].p);
    }
}
void printinput()
{
    // int TWT=0,TTAT=0;
    printf("\nPname\tAT\tBT");
    for (i = 0; i < n; i++)
        printf("\n%s\t%d\t%d", tab[i].pname, tab[i].at, tab[i].bt);
}
void printoutput()
{
    int TWT = 0, TTAT = 0;
    float ATAT, AWT;
    printf("\nPname\tAT\tBT\tFT\tWT\tTAT");
    for (i = 0; i < n; i++)
    {
        printf("\n%s\t%d\t%d\t%d\t%d\t%d", tab[i].pname, tab[i].at, tab[i].bt,
tab[i].ft, tab[i].ft-tab[i].at - tab[i].bt, tab[i].ft - tab[i].at);
        TWT = TWT + (tab[i].ft - tab[i].at - tab[i].bt);
        TTAT = TTAT + (tab[i].ft - tab[i].at);
    }
    printf("\nTotal WT: %d", TWT);
    printf("\nTotal TAT:%d", TTAT);
    AWT = (float)TWT / n;
    ATAT = (float)TTAT / n;
    printf("\nAverage WT: %f", AWT);
    printf("\nAverage TAT:%f", ATAT);
}
void sort()
{
    int pass;
    struct input temp;
    for (pass = 1; pass < n; pass++)
    {
```

```c
        for (i = 0; i < n - pass; i++)
        {
            if (tab[i].at > tab[i + 1].at)
            {
                temp = tab[i];
                tab[i] = tab[i + 1];
                tab[i + 1] = temp;
            }
        }
    }
}
int arrived(int time)
{
    for (i = 0; i < n; i++)
    {
        if (tab[i].at <= time && tab[i].tbt != 0)
            return 1;
    }
    return 0;
}
int gethighpriority(int time)
{
    int processpos, min = 99;
    for (i = 0; i < n; i++) // i=0,1
    {                       // p1,p3 min=2
        if (tab[i].at <= time && tab[i].tbt != 0 && tab[i].p < min)
        {
            min = tab[i].p;
            processpos = i;
        }
    }
    return processpos;
}
void processinput()
{
    int finish = 0, j;
    k = 0;
    while (finish != n)
    {
        if (arrived(time))
        {
            i = gethighpriority(time);
            time++;
            tab[i].tbt--;
            g[k].start = prev;
            g[k].end = time;
            prev = time;
            strcpy(g[k++].pname, tab[i].pname);
            tab[i].ft = time;
            if (tab[i].tbt == 0)
            {
                finish++;
            }
        }
        else
        {
            time++;
            g[k].start = prev;
            g[k].end = time;
            strcpy(g[k++].pname, "idle");
            prev = time;
        }
        // i++;
    }
}
void ganttchart()
{
    int i, j = 0;
    printf("\n******Each Unit Gantt chart******");
    printf("\nStart\tpname\tEnd");
    for (i = 0; i < k; i++)
    {
        printf("\n%d\t%s\t%d", g[i].start, g[i].pname, g[i].end);
    }
    printf("\n*******Final Gantt Chart*******");
    g1[0] = g[0];
    for (i = 1; i < k; i++)
    {
        if (strcmp(g[i].pname, g1[j].pname) == 0)
            g1[j].end = g[i].end;
        else
        {
            j++;
            g1[j] = g[i];
        }
    }
    printf("\nStart\tpname\tEnd");
    for (i = 0; i <= j; i++)
    {
        printf("\n%d\t%s\t%d", g1[i].start, g1[i].pname, g1[i].end);
    }
}
int main()
{
    getinput();
    printinput();
    sort();
    printf("\nData After Sorting: ");
    printinput();
    processinput();
    printoutput();
    ganttchart();
    for (i = 0; i < n; i++)
    {
        tab[i].tbt = tab[i].bt = rand() % 10 + 1;
        tab[i].at = tab[i].ft + 2;
    }
    printinput();
    processinput();
    printoutput();
    ganttchart();
}
```

14. Write the program to simulate Round Robin (RR) scheduling. The arrival time and first CPU-burst for different n number of processes should be input to the algorithm. Also give the time quantum as input. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

```
#include <stdio.h>
struct input
{
    char pname[10];
    int bt, at, tbt, ft;
} tab[10];
struct gantt
{
    char pname[10];
    int start, end;
} g[30], g1[30];
int n, time, prev, k, tq;
void getinput()
{
    int i;
    printf("\nEnter No of Process: ");
    scanf("%d", &n);
    printf("\nEnter Time quantum: ");
    scanf("%d", &tq);
    for (i = 0; i < n; i++)
    {
        printf("\nEnter Process Name: ");
        scanf("%s", tab[i].pname);
        printf("Arrival Time:");
        scanf("%d", &tab[i].at);
```

```c
        printf("Burst Time: ");
        scanf("%d", &tab[i].bt);
        tab[i].tbt = tab[i].bt;
    }
}
void printinput()
{
    // int TWT=0,TTAT=0;
    int i;
    printf("\nPname\tAT\tBT");
    for (i = 0; i < n; i++)
        printf("\n%s\t%d\t%d", tab[i].pname, tab[i].at, tab[i].tbt);
}
void printoutput()
{
    int TWT = 0, TTAT = 0, i;
    float ATAT, AWT;
    printf("\nPname\tAT\tBT\tFT\tWT\tTAT");
    for (i = 0; i < n; i++)
    {
        printf("\n%s\t%d\t%d\t%d\t%d\t%d", tab[i].pname, tab[i].at, tab[i].bt,
tab[i].ft, tab[i].ft - tab[i].at - tab[i].bt, tab[i].ft - tab[i].at);
        TWT = TWT + (tab[i].ft - tab[i].at - tab[i].bt);
        TTAT = TTAT + (tab[i].ft - tab[i].at);
    }
    printf("\nTotal WT: %d", TWT);
    printf("\nTotal TAT:%d", TTAT);
    AWT = (float)TWT / n;
    ATAT = (float)TTAT / n;
    printf("\nAverage WT: %f", AWT);
    printf("\nAverage TAT:%f", ATAT);
}
void sort()
{
    int pass, i;
    struct input temp;
    for (pass = 1; pass < n; pass++)
    {
        for (i = 0; i < n - pass; i++)
        {
            if (tab[i].at > tab[i + 1].at)
            {
                temp = tab[i];
                tab[i] = tab[i + 1];
                tab[i + 1] = temp;
            }
        }
    }
```

```c
}
int arrived(int time)
{
    int i;
    for (i = 0; i < n; i++)
    {
        if (tab[i].at <= time && tab[i].tbt != 0)
            return 1;
    }
    return 0;
}
void processinput()
{
    int finish = 0, j;
    int i = 0;
    k = 0;
    while (finish != n)
    {
        if (arrived(time))
        {
            if (tab[i].tbt != 0)
            {
                for (j = 0; j < tq; j++)
                {
                    time++;
                    tab[i].tbt--;
                    g[k].start = prev;
                    g[k].end = time;
                    prev = time;
                    strcpy(g[k++].pname, tab[i].pname);
                    tab[i].ft = time;
                    if (tab[i].tbt == 0)
                    {
                        finish++;
                        break;
                    }
                }
            }
        }
        else
        {
            time++;
            g[k].start = prev;
            g[k].end = time;
            strcpy(g[k++].pname, "idle");
            prev = time;
        }
        if (time < tab[(i + 1) % n].at)
```

```c
            i = 0;
        else
            i = (i + 1) % n;
    }
}
void ganttchart()
{
    int i, j = 0;
    printf("\n******Each Unit Gantt chart******");
    printf("\nStart\tpname\tEnd");
    for (i = 0; i < k; i++)
    {
        printf("\n%d\t%s\t%d", g[i].start, g[i].pname, g[i].end);
    }
    printf("\n*******Final Gantt Chart*******");
    g1[0] = g[0];
    for (i = 1; i < k; i++)
    {
        if (strcmp(g[i].pname, g1[j].pname) == 0)
            g1[j].end = g[i].end;
        else
        {
            j++;
            g1[j] = g[i];
        }
    }
    printf("\nStart\tpname\tEnd");
    for (i = 0; i <= j; i++)
    {
        printf("\n%d\t%s\t%d", g1[i].start, g1[i].pname, g1[i].end);
    }
}
int main()
{
    int i;
    getinput();
    printinput();
    sort();
    printf("\nData After Sorting: ");
    printinput();
    processinput();
    printoutput();
    ganttchart();
    for (i = 0; i < n; i++)
    {
        tab[i].tbt = tab[i].bt = rand() % 10 + 1;
        tab[i].at = tab[i].ft + 2;
    }
    printinput();
    processinput();
    printoutput();
    ganttchart();
}
```

15. Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames.
    Reference String : 12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8
    1) Implement FIFO

```
#include <stdio.h>
int nor, nof, refstring[30], F[10];
void accept()
{
    int i;
    printf("\nEnter the Reference String:\n ");
    for (i = 0; i < nor; i++)
    {
        printf("[%d]: ", i);
        scanf("%d", &refstring[i]);
    }
}
int search(int page)
{
    int i;
    for (i = 0; i < nof; i++)
    {
        if (page == F[i])
            return i;
    }
    return -1;
}
void FIFO()
{
    int i, j, k, fno = 0, fault = 0;
    for (i = 0; i < nor; i++)
    {
        printf("\n%d", refstring[i]);
        k = search(refstring[i]);
        if (k == -1)
        {
            F[fno] = refstring[i];
            for (j = 0; j < nof; j++)
            {
                if (F[j])
```

```
                    printf("\t%d", F[j]);
            }
            fault++;
            fno = (fno + 1) % nof;
        }
    }
    printf("\nTotal no of Page fault: %d", fault);
}
main()
{
    printf("\nEnter the Length of the string: ");
    scanf("%d", &nor);
    printf("\nEnter no. of Frames: ");
    scanf("%d", &nof);
    accept();
    FIFO();
}
```

16. Implement LRU

```
#include <stdio.h>
int nor, nof, refstring[30], F[10];
void accept()
{
    int i;
    printf("\nEnter the Reference String:\n ");
    for (i = 0; i < nor; i++)
    {
        printf("[%d]: ", i);
        scanf("%d", &refstring[i]);
    }
}
int search(int page)
{
    int i;
    for (i = 0; i < nof; i++)
    {
        if (page == F[i])
            return i;
    }
    return -1;
}
int getfno(int i)
{
    int fno, prev, pos = 99, fpos;
    for (fno = 0; fno < nof; fno++)
    {
        for (prev = i - 1; prev >= 0; prev--)
        {
            if (F[fno] == refstring[prev])
            {
                if (prev < pos)
                {
                    pos = prev;
                    fpos = fno;
                }
                break;
            }
        }
    };
    return fpos;
}
void LRU()
{
    int i, j, k, fno, fault = 0;
    for (fno = 0, i = 0; fno < nof && i < nor; i++)
    {
        printf("\n%d", refstring[i]);
        k = search(refstring[i]);

        if (k == -1)
        {
            F[fno] = refstring[i];
            for (j = 0; j < nof; j++)
            {
                if (F[j])
                    printf("\t%d", F[j]);
            }
            fault++;
            fno++;
        }
    }
    while (i < nor)
    {
        printf("\n%d", refstring[i]);
        k = search(refstring[i]);
        if (k == -1)
        {
            fno = getfno(i);
            F[fno] = refstring[i];
            for (j = 0; j < nof; j++)
            {
                if (F[j])
                    printf("\t%d", F[j]);
            }
            fault++;
        }
        i++;
    }
    printf("\nTotal no of Page fault: %d", fault);
}
main()
{
    printf("\nEnter the Length of the string: ");
    scanf("%d", &nor);
    printf("\nEnter no. of Frames: ");
    scanf("%d", &nof);
    accept();
    LRU();
}
```

17. Implement OPT

```
#include <stdio.h>
int nor, nof, refstring[30], F[10];
void accept()
{
    int i;
    printf("\nEnter the Reference String:\n ");
    for (i = 0; i < nor; i++)
    {
        printf("[%d]: ", i);
        scanf("%d", &refstring[i]);
    }
}
int search(int page)
{
    int i;
    for (i = 0; i < nof; i++)
    {
        if (page == F[i])
            return i;
    }
    return -1;
}
int getfno(int i)
{
    int fno, prev, pos = 0, fpos, flag;
    for (fno = 0; fno < nof; fno++)
    {
        flag = 0;
        for (prev = i + 1; prev < nor; prev++)
        {
            if (F[fno] == refstring[prev])
            {
                flag = 1;
                if (prev > pos)
                {
                    pos = prev;
                    fpos = fno;
                }
                break;
            }
        }
        if (flag == 0)
        {
            fpos = fno;
            break;
        }
    }
    return fpos;
}
void optimal()
{
    int i, j, k, fno, fault = 0;
    for (fno = 0, i = 0; fno < nof && i < nor; i++)
    {
        printf("\n%d", refstring[i]);
        k = search(refstring[i]);
        if (k == -1)
        {
            F[fno] = refstring[i];
            for (j = 0; j < nof; j++)
            {
                if (F[j])
                    printf("\t%d", F[j]);
            }
            fault++;
            fno++;
        }
    }
    while (i < nor)
    {
        printf("\n%d", refstring[i]);
        k = search(refstring[i]);
        if (k == -1)
        {
            fno = getfno(i);
            F[fno] = refstring[i];
            for (j = 0; j < nof; j++)
            {
                if (F[j])
                    printf("\t%d", F[j]);
            }
            fault++;
        }
        i++;
    }
    printf("\nTotal no of Page fault: %d", fault);
}
main()
{
    printf("\nEnter the Length of the string: ");
    scanf("%d", &nor);
    printf("\nEnter no. of Frames: ");
    scanf("%d", &nof);
    accept();
    optimal();
}
```

18. Implement MFU.

```
#include <stdio.h>
int RefString[20], PT[10], count[5], seq[4], nof, nor, frmno;
void Accept()
{
    int i;
    printf("\n enter the reference sting:");
    for (i = 0; i < nor; i++)
    {
        printf("[%d]=", i);
        scanf("%d", &RefString[i]);
    }
}
int search(int s)
{
    int i;
    for (i = 0; i < nof; i++)
        if (PT[i] == s)
            return (i);
    return (-1);
}
int checkcount(int e)
{
    int i, cnt = 0, Pos = 99, Posi, j, flag, cnt1 = 0;
    for (i = 0; i < nof; i++)
    {
        if (count[i] > cnt)
        {
            cnt = count[i];
        }
    }
    for (i = 0; i < nof; i++)
    {
        if (count[i] == cnt)
        {
            for (j = e - 1; j >= 0; j--)
            {
                if (PT[i] == RefString[j])
                {
                    if (j < Pos)
                    {
                        Pos = j;
                        Posi = i;
                    }
                    break;
                }
            }
        }
    }
    return Posi;
}
void MFU()
{
    int i, j, k, faults = 0, frameno, cnt;
    for (k = 0, i = 0; k < nof && i < nor; i++)
    {
        getch();
        cnt = 0;
        printf("%2d", RefString[i]);
        frameno = search(RefString[i]);
        if (frameno == -1)
        {
            PT[k] = RefString[i];
            count[k] = cnt;
            for (j = 0; j < nof; j++)
            {
                if (PT[j])
                    printf("%2d(%d)\t", PT[j], count[j]);
            }
            faults++;
            k++;
        }
        else
        {
            cnt = count[frameno];
            count[frameno] = ++cnt;
            for (j = 0; j < nof; j++)
            {
                if (PT[j])
                    printf("%2d(%d)\t", PT[j], count[j]);
            }
            printf("    No page fault");
        }
        printf("\n\n");
    }
    k = 0;
    while (i < nor)
    {
        getch();
        cnt = 0;
        printf("%2d", RefString[i]);
        frameno = search(RefString[i]);
        if (frameno == -1)
        {
            k = checkcount(i);
            PT[k] = RefString[i];
            count[k] = cnt;
            for (j = 0; j < nof; j++)
            {

                printf("%2d(%d)\t", PT[j], count[j]);
            }
            faults++;
        }
        else
        {
            cnt = count[frameno];
            count[frameno] = ++cnt;
            for (j = 0; j < nof; j++)
            {
                printf("%2d(%d)\t", PT[j], count[j]);
            }
            printf("    No page fault");
        }
        i++;
        printf("\n\n");
    }
    printf("total page faults:%d", faults);
}
int main()
```

```
{
    printf("\n enter the length of Reference string:");
    scanf("%d", &nor);
    printf("\n enter the number of frames:");
    scanf("%d", &nof);
    Accept();
    MFU();
}
```