OS LAB IA PROGRAMS

Program 1:

Develop a C program to implement the Process system calls (fork (), exec(), wait(), create process, terminate process)

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

int main()
{
    pid_t pid;
    pid = fork();

    if (pid < 0)
    {
        perror("Fork failed");
        exit(EXIT_FAILURE);
    }
    else if (pid == 0)
    {
        printf("Child process (PID: %d) created successfully.\n", getpid());

        execl("/bin/ls","ls",NULL);
        perror("excel");
        exit(EXIT_FAILURE);

    }
    else
    {
        printf("Parent process (PID: %d) waiting for child process to
complete.\n", getpid());

        int status;
        pid_t child_pid = wait(&status);

        if(child_pid < 0)
        {
            perror("wait");
            exit(EXIT_FAILURE);
        }

        if (WIFEXITED(status))
        {
            printf("Child process (PID: %d) terminated normally with exit
status: %d.\n",child_pid, WEXITSTATUS(status));
```

```
        }
        else
        {
            printf("Child process terminated abnormally.\n");
        }

        printf("Parent process (PID: %d) terminating.\n",getpid());
    }

    return 0;
}
```

Program 2 :

Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) FCFS b) SJF.

```
#include <stdio.h>
#define MAX 100

typedef struct
{
    int process_id;
    int burst_time;
    int waiting_time;
    int turnaround_time;
    int prp;
} Process;


void calculateFCFS(Process processes[], int n)
{
    processes[0].waiting_time = 0;

    printf("\n--- FCFS Scheduling ---\n");
    printf("Process\tBT\tTAT\tWT\n");

    for (int i =1; i < n; i++)
        processes[i].waiting_time = processes[i-1].burst_time + processes[i-
1].waiting_time;

    for(int i =0;i<n;i++)
        processes[i].turnaround_time = processes[i].waiting_time +
processes[i].burst_time;

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

```
    {
        printf("P%d\t%d\t%d\t%d\n", processes[i].process_id,
           processes[i].burst_time, processes[i].turnaround_time,
processes[i].waiting_time);
    }
}

void calculateSJF(Process processes[], int n)
{

    for(int i=0;i<n-1;i++)
    {
        for(int j=0;j<n-i-1;j++)
        {
            if(processes[j].burst_time > processes[j+1].burst_time)
            {
                Process temp = processes[j];
                processes[j] = processes[j+1];
                processes[j+1] = temp;
            }
        }
    }

    printf("\n--- SJF Scheduling ---\n");
    printf("Process\tBT\tTAT\tWT\n");

    processes[0].waiting_time = 0;

    for(int i=1;i<n;i++)
        processes[i].waiting_time = processes[i-1].waiting_time + processes[i-
1].burst_time;

    for(int i=0;i<n;i++)
        processes[i].turnaround_time = processes[i].waiting_time +
processes[i].burst_time;


    for(int i = 0;i<n;i++)
    {
        printf("P%d\t%d\t%d\t%d\n",
processes[i].process_id,processes[i].burst_time,
processes[i].turnaround_time,processes[i].waiting_time);
    }
}

int main()
{
    int n;
```

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

    Process processes[n];

    for (int i = 0; i < n; i++) {
        processes[i].process_id = i + 1;
        printf("Enter the Burst Time for Process P%d:\n",
processes[i].process_id);
        scanf("%d", &processes[i].burst_time);
    }

    calculateFCFS(processes, n);

    calculateSJF(processes, n);

    return 0;
}
```

Program 3:

Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a)Round Robin b) Priority.

```
#include <stdio.h>
#define MAX 100

typedef struct
{
    int process_id;
    int burst_time;
    int waiting_time;
    int turnaround_time;
    int priority;
} Process;

void calculatePriority(Process processes[], int n)
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (processes[j].priority > processes[j + 1].priority)
            {
                Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
```

```
    }

    printf("\n--- Priority Scheduling ---\n");
    printf("Process\tBT\tTAT\tWT\n");

    processes[0].waiting_time = 0;

    for (int i = 1; i < n; i++)
    {
        processes[i].waiting_time = processes[i - 1].waiting_time +
processes[i - 1].burst_time;
    }

    for (int i = 0; i < n; i++)
    {
        processes[i].turnaround_time = processes[i].waiting_time +
processes[i].burst_time;
        printf("P%d\t%d\t%d\t%d\n", processes[i].process_id,
               processes[i].burst_time, processes[i].turnaround_time,
processes[i].waiting_time);
    }
}

void calculateRR(Process processes[], int n, int quantum)
{
    int remaining_bt[MAX];
    for (int i = 0; i < n; i++) {
        remaining_bt[i] = processes[i].burst_time;
        processes[i].waiting_time = 0;
    }

    int t = 0;

    while (1)
    {
        int done = 1;

        for (int i = 0; i < n; i++)
        {
            if (remaining_bt[i] > 0)
            {
                done = 0;
                if (remaining_bt[i] > quantum)
                {
                    t += quantum;
                    remaining_bt[i] -= quantum;
                }
                else
```

```
                {
                    t += remaining_bt[i];
                    processes[i].waiting_time = t - processes[i].burst_time;
                    remaining_bt[i] = 0;
                }
                processes[i].turnaround_time = processes[i].waiting_time +
processes[i].burst_time;
            }
        }

        if (done)
            break;
    }

    printf("\n--- Round Robin Scheduling ---\n");
    printf("Process\tBT\tTAT\tWT\n");

    for (int i = 0; i < n; i++)
    {
        printf("P%d\t%d\t%d\t%d\n", processes[i].process_id,
               processes[i].burst_time, processes[i].turnaround_time,
processes[i].waiting_time);
    }
}

int main()
{
    int n;

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

    Process processes[n];

    for (int i = 0; i < n; i++)
    {
        processes[i].process_id = i + 1;
        printf("Enter the Burst Time for Process P%d: ",
processes[i].process_id);
        scanf("%d", &processes[i].burst_time);
    }

    for (int i = 0; i < n; i++)
    {
        printf("Enter the Priority for Process P%d (Lower number = Higher
priority): ", processes[i].process_id);
        scanf("%d", &processes[i].priority);
    }
```

```
    calculatePriority(processes, n);

    int quantum;
    printf("\nEnter time quantum for Round Robin: ");
    scanf("%d", &quantum);

    calculateRR(processes, n, quantum);

    return 0;
}
```

Program 4:

Develop a C program to simulate producer-consumer problem using semaphores.

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

int mutex = 1;
int full = 0;
int empty = 3, x=0;

void producer()
{
    --mutex;
    ++full;
    --empty;
    x++;
    printf("Producer produces item %d",x);
    ++mutex;
}

void consumer()
{
    --mutex;
    --full;
    ++empty;
    printf("Consumer consumes itme %d",x);
    x--;
    ++mutex;
}

int main()
{
    int n;
    printf("\n1.Press 1 for Producer"
```

```
            "\n2.Press 2 for Consumer"
            "\n3.Press 3 to exit");

    for(int i=1;i>0;i++)
    {
        printf("\nEnter your choice : ");
        scanf("%d",&n);
        switch(n)
        {
            case 1:
                    if((mutex == 1) && (empty!=0))
                    {
                        producer();
                    }
                    else
                    {
                        printf("Buffer is full!\n");
                    }
                    break;

            case 2:
                    if((mutex == 1) && (full!=0))
                        consumer();
                    else
                        printf("Buffer is empty!\n");
                    break;

            case 3: exit(0);
        }
    }
}
```

Program 5:

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.

//WRITER PROCESS

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

int main()
{
    int fd;
```

```
    fd = mkfifo("fifo",0777);
    printf("Named pipe created\n");
    return 0;
}
```

//SENDER PROCESS

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

int main()
{
    int fd;

    fd = open("fifo",O_WRONLY);
    write(fd,"Message" , 7);

    printf("Sender process having PID %d sent the data\n",getpid());
}
```

//READER PROCESS

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

int main()
{
    int n,fd;

    char buffer[100];

    fd = open("fifo",O_RDONLY);

    n = read(fd,buffer,100);

    printf("Reader process having PID %d sent the data\n",getpid());

    printf("Data received by sender %d is %s\n",getpid(), buffer);

    return 0;
}
```

Program 6:

Develop a C program to simulate the Linked file allocation strategies.

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

void main()
{
    int f[50],p,i,st,len,j,c,k,a;

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

    printf("Enter how many blocks already craeted : ");
    scanf("%d",&p);

    printf("Enter the blocks already allocated : ");

    for(i=0;i<p;i++)
    {
        scanf("%d",&a);
        f[a] = 1;
    }

    x: printf("Enter the index staring block and the length : "); scanf("%d
%d",&st,&len);

    k = len;

    if(f[st] == 0)
    {
        for(j=st;j<(st + k);j++)
        {
            if(f[j] == 0)
            {
                f[j] = 1;
                printf("%d -------> %d\n",j,f[j]);
            }
            else
            {
                printf("%d block is already allocated\n");
                k++;
            }
        }
    }
```

```
    else
    {
        printf("%d starting block is already allocated",st);
    }

    printf("\nDo you want to enter more files (YES - 1)/(NO - 0) : ");

    scanf("%d",&c);

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