Operating System and System Programming Lab (15B17CI472)

Lab Test 1 : Odd 2021 (23/10/2021)

Patil Amit Gurusidhappa

19104004

B11

Q1.

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

struct args

{

    int N;

};

// Function For finding Prime numbers upto N

void \*findprimeNoUptoN(void \*input)

{

    int low = 0, high = ((struct args \*)input)->N, i, flag;

    while (low < high)

    {

        flag = 0;

        if (low <= 1)

        {

            ++low;

            continue;

        }

        for (i = 2; i <= low / 2; ++i)

        {

            if (low % i == 0)

            {

                flag = 1;

                break;

            }

        }

        if (flag == 0)

            printf("%d ", low);

        ++low;

    }

}

// Function For finding Fabonacci numbers upto N

void \*findFabonacciNoUptoN(void \*input)

{

    int sum = 0, n = ((struct args \*)input)->N;

    int a = 0;

    int b = 1;

    while (sum <= n)

    {

        printf("%d ", sum);

        a = b;

        b = sum;

        sum = a + b;

    }

}

int main()

{

    int N;

    printf("Enter Number N\n");

    scanf("%d", &N);

    struct args \*Argument = (struct args \*)malloc(sizeof(struct args));

    Argument->N = N;

    printf("Print Prime numbers till %d\n",N);

    pthread\_t tid;

    pthread\_create(&tid, NULL, findprimeNoUptoN, (void \*)Argument);

    pthread\_join(tid, NULL);

    printf("\nPrint Fabonacci upto %d\n",N);

    pthread\_t tid\_1;

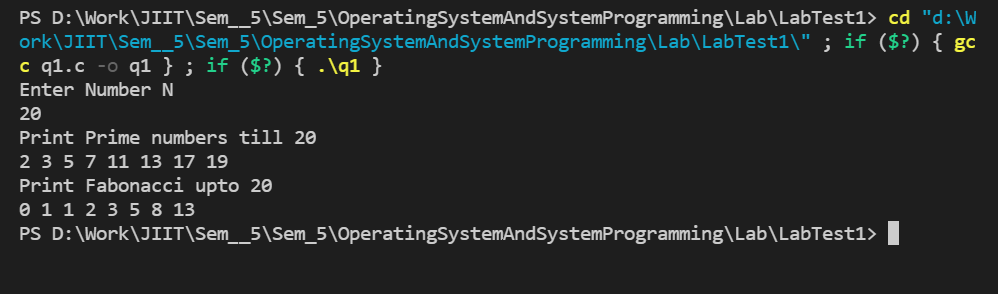
    pthread\_create(&tid\_1, NULL, findFabonacciNoUptoN, (void \*)Argument);

    pthread\_join(tid\_1, NULL);

    return 0;

}

Output



Q2.

#include <stdio.h>

struct ProcessControlBlock

{

    int pid, wait, burst, turnaround, arrival;

};

void pline(int x)

{

    for (int i = 0; i < x; i++)

    {

        printf("-");

    }

    printf("\n");

}

void main()

{

    int i, num, j;

    int w\_total = 0; //waiting total

    int t\_total = 0; //turnaround total

    float w\_avg = 0.0, t\_avg = 0.0;

    float sum = 0.0;

    struct ProcessControlBlock p[10], temp;

    printf("SJF Algorithms\n");

    // taking the process data input

    printf("Enter the total number of Processes: ");

    scanf("%d", &num);

    for (i = 0; i < num; i++)

    {

        printf("Enter the process id, arrival time and burst time for process %d: \n", i + 1);

        scanf("%d %d %d", &p[i].pid, &p[i].arrival, &p[i].burst);

    }

    // sorting of processes according to burst time

    for (i = 0; i < num - 1; i++)

    {

        for (j = 0; j < num - i - 1; j++)

        {

            if (p[j].burst > p[j + 1].burst)

            {

                temp = p[j];

                p[j] = p[j + 1];

                p[j + 1] = temp;

            }

        }

    }

    // calculate the turnaround time and waiting time of each process

    for (i = 0; i < num; i++)

    {

        p[i].wait = sum;

        sum = sum + p[i].burst;

        p[i].turnaround = sum;

    }

    // draw lines

    pline(44);

    printf("PID\tArrival\tBurst\tWaiting\tTurnaround\n");

    pline(44);

    // print all the processes

    for (i = 0; i < num; i++)

    {

        printf("%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrival, p[i].burst, p[i].wait, p[i].turnaround);

        // sum of all the turnaround time of processes

        t\_total += p[i].turnaround;

        // total waiting time

        w\_total += p[i].wait;

    }

    pline(44);

    w\_avg = w\_total / (float)num;

    t\_avg = t\_total / (float)num;

    printf("\nTotal Turnaround time: %d", t\_total);

    printf("\nAverage Turnaround time: %.3f", t\_avg);

    printf("\nTotal Waiting time: %d", w\_total);

    printf("\nAverage Waiting time: %.3f", w\_avg);

}

Output

