|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Family Name(s)** | | Wulon | | Module No.  Faculty Date Stamp  (or signature and date) |
| **Forename(s)** | | **Yussif Muniru** | |
| **ID Number(s) (from your student card)** | | **050916006/8356015** | |
| Estimated Time taken (hrs) (per student for group coursework) | |  | |
| **Lecturer** | **Prof. Sangeetha** | | Lab group / Tutorial group / Tutor (if applicable) | |
| **Module Code and Title** | **Operating Systems and Networks** | |  | |
| **Assignment No. / Title** |  | | Extensions & late submissions allowed: | |
| Estimated Time | Assignment type: (Individual/Group) | | % of Module | Hand out date: |
| Penalties: Marks will be reduced by 10% of the original mark for every week late. No work will be accepted that is more than two weeks late. If you are unable to submit coursework on time due to extenuating circumstances you may be eligible for an extension. | | | | |
| Declaration: I/we the undersigned confirm that I/we have read and agree to abide by the University regulations on plagiarism and cheating and Faculty coursework policies and procedures. I/we confirm that this piece of work is my/our own. I/we consent to appropriate storage of our work for checking to ensure that there is no plagiarism/ academic cheating.  Signature(s): -------------------------------------- -------------------------------------- -------------------------------------- -------------------------------------- | | | | |



APPENDIX A: Coursework Front Sheet

GHANA TECHNOLOGY UNIVERSTY COLLEGE/COVENTRYUNIVERSTY

FACULTY OF COMPUTING & INFORMATION SYSTEMS

*Section A – To be completed by the student - PLEASE PRINT CLEARLY*

##### Section B - To be completed by the assessor

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Marks breakdown and Assessor’s Feedback** | | | | | Max | Awarded |
|  | | | | |  |  |
|  |  |
|  |  |
|  |  |
| *Please continue on the reverse of this sheet if needed.* | | | | | | |
| Assessor’s name | | Assessor’s signature | | Total | | |
| Coursework Mark | Extension Agreed until: (Yes / No) | | Deduction for Late Submission (Penalty) | Final Mark | | |
|  |  | |  |  | | |
| Date:  Signed internal moderator: | | | Date:  Programme Leader Signature: | | | |

This work may have been moderated. You may find additional comments in the work.

Note: The results on this script are provisional, subject to SAB/PAB review.

:

**CPU scheduling** in a nut shell the process that permits a single process to use the resources of the computer CPU (thus the computation power of the CPU) for a particular time period while maybe some other processes are in a queue ( in what is referred to as being in the waiting state). This maybe due to occupation of some systems resources like the input and output devices. The goal of CPU scheduling is just to provide a fair, optimized, efficient and fast system. This scheduling is usually done through the use of algorithms referred to as **scheduling algorithms**. Some of such algorithms are listed below.

* Shortest Job next scheduling algorithm.
* Priority scheduling algorithm.
* Round robin scheduling algorithm.
* First-come First-served scheduling algorithm.

These algorithms can either be **preemptive** or **non-preemptive**. The preemptive ones can remove a process from the CPU if another process comes into the waiting area with a higher priority whether the former process has completed its burst time not. But with the non-preemptive ones, the former process has to complete its burst or execution time before it leaves the CPU.

**Shortest Job next scheduling algorithm.**

**INTRODUCTION:**

Shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJN is a non-preemptive algorithm.

**HOW IT WORKS.**

For example lets take four processes, P1,P2,P3,P4,P5. They have arrival times as 0, 2, 4, 5, 7 respectively. They happen to arrive in the ready queue at these times. Below is a chart providing more information about these processes.

1. Completion Time: The Completion time is the time taken for the process to finish it’s execution.
2. Turn Around Time: The turn around time is the difference between the completion time and the arrival time.
3. Waiting Time(W.T): The waiting is the difference between turn around time and the burst time.

Formulae for these times.

Turn around time = completion time – arrival time.   
Waiting Time = Turn Around Time – Burst Time.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Process ID** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turn Around Time** | **Waiting Time** |
| 1 | 0 | 2 | 2 | 2 | 0 |
| 2 | 2 | 2 | 4 | 2 | 1 |
| 3 | 3 | 3 | 6 | 3 | 4 |
| 4 | 5 | 1 | 7 | 2 | 9 |
| 5 | 7 | 1 | 8 | 1 | 17 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 |  | P2 |  | P3 |  |  | P4 | P5 |
|  |  |  |  |  |  |  |  |  |

0 2 4 6 7 8

Avg. waiting time = (0 + 0 +0 +1 + 0) / 5 = 0.2

With the shortest job next algorithm, if there are two or more processes in the ready queue, these processes are executed based on their burst times rather than their arrival times. Thus, the one with the smaller burst time is executed earlier than the one with the larger burst time irrespective of their arrival times. It is also known as SJN. And it has the minimum average waiting time as compared to all the other scheduling algorithms.

**C++ implementation of the shortest job first algorithm.**

**How it works.**

1- Sort all the process in ascending order of burst times.

2- Then simply, process time according to the one that comes first. Then the next and the next.

3- Find the waiting time of all the processes.

4- Find the turn-around time for the all the processes.

5- Find the average waiting time for all the processes.

C++ program to implement Shortest Job first

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

    int et[20],at[10],n,i,j,temp,st[10],ft[10],wt[10],ta[10];

    int totwt=0,totta=0;

    float awt,ata;

    char pn[10][10],t[10];

    //clrscr();

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

    scanf("%d",&n);

    for(i=0; i<n; i++)

    {

        printf("Enter process name, arrival time& execution time:");

        //flushall();

        scanf("%s%d%d",pn[i],&at[i],&et[i]);

    }

    for(i=0; i<n; i++)

        for(j=0; j<n; j++)

        {

            if(et[i]<et[j])

            {

                temp = at[i];

                at[i] = at[j];

                at[j] = temp;

                temp = et[i];

                et[i] = et[j];

                et[j] = temp;

                strcpy(t,pn[i]);

                strcpy(pn[i],pn[j]);

                strcpy(pn[j],t);

            }

        }

    for(i=0; i<n; i++)

    {

        if(i==0)

            st[i]=at[i];

        else

            st[i]=ft[i-1];

        wt[i]=st[i]-at[i];

        ft[i]=st[i]+et[i];

        ta[i]=ft[i]-at[i];

        totwt+=wt[i];

        totta+=ta[i];

    }

    awt=(float)totwt/n;

    ata=(float)totta/n;

    printf("\nPname\tarrivaltime\texecutiontime\twaitingtime\ttatime");

    for(i=0; i<n; i++)

        printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],wt[i],ta[i]);

    printf("\nAverage waiting time is:%f",awt);

    printf("\nAverage turnaroundtime is:%f",ata);

    getch();

}

 OUTPUT:

Enter the number of process:3

Enter process name, arrival time ; execution time:1 0.0 5

Enter process name, arrival time; execution time:2 1 6

Enter process name, arrival time; execution time:3 2 7

Pname arrivaltime executiontime waitingtime tatime

2 0.0 5 1 6

4 7 6 5 11

3 2 7 10

17

Average waiting time = (1 + 5 + 10)/ 3 = 5.

**Advantages of SJN.**

* Shortest Job first has the advantage of having minimum average waiting time among all scheduling algorithms.

**Disadvantages OF SJN**

* Even though starvation can be solved using aging, one major disadvantage of the SJN algorithm is that it causes starvation.
* It is in reality, not practical to use the algorithm in operating systems since the burst time of the next process cannot be known before-hand.

# First Come First Serve Scheduling.

In the "First come first serve" scheduling algorithm, the process which comes first, gets executed first, or we can say that the process which asks for resources of the CPU first, gets the CPU resources and finishes all its processing before leaving the CPU.

**HOW IT WORKS.**

For example lets take four processes, P1,P2,P3,P4,P5

. They have arrival times as 0, 2, 4, 5, 7 respectively. They happen to arrive in the ready queue at these times. Below is a chart providing more information about these processes.

1. Completion Time: The Completion time is the time taken for the process to finish it’s execution.
2. Turn Around Time: The turn around time is the difference between the completion time and the arrival time.
3. Waiting Time(W.T): The waiting is the difference between turn around time and the burst time.

Formulae for these times.

Turn around time = completion time – arrival time.   
Waiting Time = Turn Around Time – Burst Time.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Process ID** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turn Around Time** | **Waiting Time** |
| 1 | 0 | 2 | 2 | 2 | 0 |
| 2 | 2 | 2 | 4 | 2 | 1 |
| 3 | 3 | 3 | 6 | 3 | 4 |
| 4 | 5 | 1 | 7 | 2 | 9 |
| 5 | 7 | 1 | 8 | 1 | 17 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 |  | P2 |  | P3 |  |  | P4 | P5 |
|  |  |  |  |  |  |  |  |  |

0 2 4 6 7 8

Avg. waiting time = (0 + 0 +0 +1 + 0) / 5 = 0.2

With the first-come first-served algorithm, if there are two or more processes in the ready queue, these processes are executed based on their arrival times. Thus, the one with the the smallest arrival time or came first will be fully executed before the rest. It is also known as FIFO. It is very simple and easy to implement.

**C++ implementation of the first-come first-served algorithm.**

**How it works.**

1- Input the processes with their individual burst times.

2- Find the waiting time for the processes.

3- Since the first process need not wait,its waiting time will be 0;

4- Find the turn-around time for the all the processes.

5- Find the average waiting time for all the processes.

#include<iostream>

using namespace std;

// function to find the waiting time for all processes

void findWaitingTime(int processes[], int n, int bt[], int wt[])

{

// waiting time for first process will be 0

wt[0] = 0;

// calculating waiting time

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

{

wt[i] = bt[i-1] + wt[i-1];

}

}

// function to calculate turn around time

void findTurnAroundTime( int processes[], int n, int bt[], int wt[], int tat[])

{

// calculating turnaround time by adding

// bt[i] + wt[i]

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

{

tat[i] = bt[i] + wt[i];

}

}

// function to calculate average time

void findAverageTime( int processes[], int n, int bt[])

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

// function to find waiting time of all processes

findWaitingTime(processes, n, bt, wt);

// function to find turn around time for all processes

findTurnAroundTime(processes, n, bt, wt, tat);

// display processes along with all details

cout << "Processes "<< " Burst time "<< " Waiting time " << " Turn around time\n";

// calculate total waiting time and total turn around time

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

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

cout << " " << i+1 << "\t\t" << bt[i] <<"\t "<< wt[i] <<"\t\t " << tat[i] <<endl;

}

cout << "Average waiting time = "<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "<< (float)total\_tat / (float)n;

}

// main function

int main()

{

// process ids

int processes[] = { 1, 2, 3, 4};

int n = sizeof processes / sizeof processes[0];

// burst time of all processes

int burst\_time[] = {21, 3, 6, 2};

findAverageTime(processes, n, burst\_time);

return 0;

}

OUTPUT:

Pname arrivaltime executiontime waitingtime tatime

2 0.0 5 1 6

4 7 6 5 11

3 2 7 10

17

Average waiting time = (1 + 5 + 10)/ 3 = 5.

**Advantages**

1. Easy and simple to implement.

**Disadvantages**

1. Not optimal average waiting time.
2. There is the tendency of the convoy effect.

**Portfolio 2.**

**File Management.**

With Unix file management commands are numerous and very helpful in navigating the file structure of your computer easily and efficiently. Using commands like “ ls ” to list the content of directories and “mkdir “ to create a directory. One of the most common and most used commands in most operating systems not just Unix is the “ cd ” command, which is used to change a navigate from one directory to another. Below is a list of some of the most commonly used linux:

1. **Ls** : The “ ls “ command is used to list the contents in a particular directory. Thus it can be used show or list the files and even other directories inside of any directory.

It is used like this: [Yussif@localhost ~]# ls.

1. **Mkdir**: The “ mkdir” command is used to create a new directory in the file system. It basically initiates and initializes an empty directory. Other files can be put in that directory alongside some other directories. The command used here is :

[Yussif@localhost ~]# mkdir.

1. **Cd**: Perhaps the most commonly used linux file management system command. It is used to navigate between different directories. It can be combined with the “ \” character to go back a directory or to go straight to the parent directory. The command used here is :

**cd/myFirstFolder/mySecondFolder**.

Now explaining this command, what it is essentially doing is that, it is first of all

Going into a folder called “myFirstFolder” then going into another folder which is inside the “myFirstFolder” called “mySecondFolder”.

1. **MV** : The MV command is used to move a file from one location to the next. This is mostly on the Graphical user interface simply by dragging and dropping a file from one folder to the next. The command is used like below:

mv/file/originalLocation/ file /newLocation/

What is essentially happening above is the the file named “file” is being move from original location called “originalLocation” to new location refered to as “newLocation”.

1. **MAN**: The “man” command also refered to as the manual command is used to show all or detailed information about another command. For example using the manual command to show the description or information about the list or “ls” command can be done like this:

**man ls**

The above command will print out a detailed information about the “ls” or list command.

1. **Locate**: The “locate” command is like the google on your computer searching for files. It functions as a file finder . It operates by finding files on your system which you might have difficulty searching for. Either you have forgotton where those files are located or if you don’t really know the full name of the file. For example:

**locate -i \*firstFile\*\*secondFile\*\*thirdFile\***

Using the “ I ” in the command means that the linux system should ignore the capitalization of the characters in the names of the files( the linux system operates on lowercase). The asterisk “\*” before the name of the file tells the system to search on the basis of a wildcard. Meaning the system should search for and bring back any file resembling or having a similar name criteria as the searched file.

1. **Touch** : The touch command is an amazing to use command. The purpose of it is to make new files in the system. It is very similar to the “mkdir” command since it makes empty files inside empty files inside directories. The command can be used as described below:

**touch myFile.doc**

The command above creates an empty file in the system called “myFile.doc”. The file is of type “doc” that means this file is a doc file. Since there can be other file types like text files with extension “.txt”.