Submitted By:

Youssef Khaled :1190324

Yasmin Hashem :420013

Mahynour Alaa:1190339

Yahia Ahmed :1190445

Submission Date: 11/5/2022

Algorithms:

1. **Shortest Job First:**

*Data structures used:*

A queue was used with type struct in order to hold the information of each process. It represents the ready queue.

struct Queue \*runtime\_process\_queue = createQueue();

Each node of the queue has the information of the queue “as struct” containing the pid, arrival time, runtime and priority which are meant to be taken from the generated input file. As well as some variables to keep track of start time, finish time, remaining time, waiting time, turnaround time, weighted turnaround time in order to print them to the output file. Those could be considered as the **PCB** of the process.

When a process arrives it gets sent as a message. When the message is received it gets the process and enqueue it in the runtime\_process\_queue which is supposed to be the ready queue.

If more than one message arrives at the same time, each process is enqueued to the queue by the priority of it’s runtime where the process with shortest runtime is put at the beginning of the queue.

This was done by **priority enqueue function**. The process gets enqueued by its information.

prienQueue(runtime\_process\_queue, message.p.id, message.p.arrival, message.p.runtime, message.p.runtime, message.p.pid, WAITING);

A while loop is done in order to loop around the ready queue and checks if a process’s time has come to be enqueued to the running queue. The process gets dequeued from the **ready** **queue** “**runtime\_process\_queue**” and gets enqueued to the **running** **queue** “**Print\_Process**”, which was defined at the beginning as a queue.

  struct QNode \*print\_process;

It then forks the process “creates it”, gets the start time “which is the clock time in the mean time” and waiting time and prints to the output file the details of this process.

After the process finishes “runtime=0” , it gets the finish time from the clock time and gets the rest of the calculations needed to be printed.

*Explanation & results:*

In this algorithm, if only one process exists in the ready queue it gets to the running queue regarding its runtime. However, if more than one process exists in the ready queue the one with the **smallest runtime** gets processed to the running queue first.

*Example of input file:*

*#id arrival runtime priority*

*1 2 10 5*

*2 12 7 1*

*3 16 4 0*

*4 25 11 6*

*5 25 3 4*

*Example of output file:*

*at time  2 process  1 started  arr   2  remain 10  wait  0*

*at time 11 process  1 finished  arr   2  remain  0  wait  0  TA  9  WTA  0*

*at time 12 process  2 started  arr  12  remain 7  wait  0*

*at time 18 process  2 finished  arr  12  remain  0  wait  0  TA  6  WTA  0*

*at time 18 process  3 started  arr  16  remain 4  wait  2*

*at time 22 process  3 finished  arr  16  remain  0  wait  2  TA  6  WTA  1*

*at time 25 process  5 started  arr  25  remain 3  wait  0*

*at time 28 process  5 finished  arr  25  remain  0  wait  0  TA  2  WTA  0*

*at time 28 process  4 started  arr  25  remain 11  wait  3*

*at time 38 process  4 finished  arr  25  remain  0  wait  3  TA 13  WTA  1*

After the start of each process the line

**“at time x process x started arrival x remaining x waiting x”**

Where it shows which process started and when and its details of arrival time and its runtime as well as how much time it waited to be directed from ready queue to running queue.

After the end of each process the line

**“at time x process x finished arrived x remaining x waiting x TA x WTA x”**

Where it shows which process finished and when and its details of arrival time and the remaining time for it which is supposed to be 0 at the end as well as the waiting time “that is same to the waiting time calculated at the start line”. Moreover, the turnaround time and weighted turnaround time are calculated

*Equations used:*

**Turnaround time:** *finish-arrival*

**Weighted turnaround time:** *turnaround time/ runtime*

**Waiting time:** *start-arrival* “as its not preemptive so the waiting time is only at the beginning”

**2.Permeative Highest priority First:**

*Data structures used:*

A priority queue was used with type struct to hold the data of the process and the timing variables are stored in the node itself.

*Explanation & results:*

First when we fork the first process on the top of the priority queue, we add the run time +the clk to a variable named timer ,we use a condition that we will not fork a new process till clk reaches timer or a new process with higher priority is on the front of the queue and if we stopped the process before we will continue it when it is on the front of the queue and the current process has finished .we will distinguish between ready and stopped processes with the id as it will appear negative when stopped.

*Example of input file:*

#id arrival runtime priority

1 2 10 5

2 12 7 4

3 16 4 3

4 25 3 2

5 25 11 1

*Example of output file:*

#PHPF

At time 2 process 1 started arr 2 total 10 remain 10 wait 0

At time 12 process 1 stopped arr 2 total 10 remain 1 wait 10

At time 12 process 2 started arr 12 total 7 remain 7 wait 0

At time 16 process 2 stopped arr 12 total 7 remain 4 wait 4

At time 16 process 3 started arr 16 total 4 remain 4 wait 0

At time 20 process 3 finished wait 4 TA 4 WTA 1

At time 25 process 5 started arr 25 total 11 remain 11 wait 0

At time 36 process 5 finished wait 4 TA 11 WTA 1

At time 37 process 4 started arr 25 total 3 remain 3 wait 12

At time 40 process 4 finished wait 23 TA 15 WTA 5

At time 41 process 2 resumed arr 12 total 7 remain 3 wait 29

At time 44 process 2 finished wait 28 TA 32 WTA 4

At time 49 process 1 resumed arr 2 total 10 remain 7 wait 47

At time 49 process 1 finished wait 44 TA 47 WTA 4

*Assumptions :*

If two process arrived at the same and the second one has higher priority, the first will start and stop immediately

**2.ROUND ROBIN:**

*Data structures used:*

A priority queue was used with type struct to hold the data of the process and the timing variables are stored in the node itself.

*Explanation & results:*

we recieve all the proccesses with status intialized as waiting which indicates the process has never ran before , so inside the algorithm itself we first check on the process status , if waiting then we fork it for the first time , save it’s start time and start and see if it’s runtime is gonna finish before the qunatnum .. if not we stop it using sigstop , if not then it’s status might be stopped , stopped is the status of the process that took it’s turn in run , but didn’t finsih till it’s end as it’s runtime is > the quantum , if yes then we re run this process using sigcont signal and then we have 2 options it’s either the left runtime is less than or equal to the quantum then we run it till it’s finished and change the status to finished , print out all the needed calculations , the other option is that the runtime is still > the quantum so we resume till end of the quantum then stop by sigstop and re enqueue the process into the proccesses queue to wait for it’s next turn with it’s status changed to stopped till all the processes are finsihed

*Example of input file:*

*#id arrival runtime priority*

*1 2 10 5*

*2 12 7 4*

*3 16 4 3*

*4 25 11 2*

*5 25 3 1*

*Example of output file:*

#at time: 2 process: 1 started , arr = 2 , remain= 10 , wait= 0

at time: 5 process: 1 stopped , arr = 2 , remain = 7 , wait= 0

at time: 12 process: 1 resumed , arr = 2 , remain= 7 , wait= 7

at time: 15 process: 1 stopped , arr = 2 , remain = 4 , wait= 7

at time: 16 process: 2 started , arr = 12 , remain= 7 , wait= 4

at time: 19 process: 2 stopped , arr = 12 , remain = 4 , wait= 4

at time: 25 process: 1 resumed , arr = 2 , remain= 4 , wait= 17

at time: 28 process: 1 stopped , arr = 2 , remain = 1 , wait= 17

at time: 28 process: 3 started , arr = 16 , remain= 4 , wait= 12

at time: 31 process: 3 stopped , arr = 16 , remain = 1 , wait= 12

at time: 31 process: 2 resumed , arr = 12 , remain= 4 , wait= 16

at time: 34 process: 2 stopped , arr = 12 , remain = 1 , wait= 16

at time: 34 process: 4 started , arr = 25 , remain= 11 , wait= 9

at time: 37 process: 4 stopped , arr = 25 , remain = 8 , wait= 9

at time: 37 process: 5 started , arr = 25 , remain= 3 , wait= 12

at time: 40 process: 5 finished , arr = 25 , remain= 0 , wait= 12 TA 15 WTA 5

at time: 40 process: 1 resumed , arr = 2 , remain= 1 , wait= 29

at time: 41 process: 1 finished , arr = 2 , remain= 0 , wait= 29 TA 39 WTA 3

at time: 41 process: 3 resumed , arr = 16 , remain= 1 , wait= 22

at time: 42 process: 3 finished , arr = 16 , remain= 0 , wait= 22 TA 26 WTA 6

at time: 42 process: 2 resumed , arr = 12 , remain= 1 , wait= 24

at time: 43 process: 2 finished , arr = 12 , remain= 0 , wait= 24 TA 31 WTA 4

at time: 43 process: 4 resumed , arr = 25 , remain= 8 , wait= 15

at time: 46 process: 4 stopped , arr = 25 , remain = 5 , wait= 15

at time: 46 process: 4 resumed , arr = 25 , remain= 5 , wait= 15

at time: 49 process: 4 stopped , arr = 25 , remain = 2 , wait= 15

at time: 49 process: 4 resumed , arr = 25 , remain= 2 , wait= 15

at time: 51 process: 4 finished , arr = 25 , remain= 0 , wait= 15 TA 26 WTA 2

*Assumptions :*

Assuming that

quantum = 3

The next process starts at the same second the one before it finishes or stops

Waiting = (current time since arrival)-(the time the process ran through the system)

**MLFQ:**

*Data structure used:*

one priority Queue with the priority to the process which has the least priority

*Explanation:*

The processes which have the highest priorities run as RR with quantum of 1 sec then they are enqueued again with a priority higher by one, to join the process with higher priority and they are all enqueued again with higher priority after each quantum. When all of the processes reach the priority of 10, they are all enqueued again with their original priority. Until their runtimes finish.

*Results:*

*at time 1 process 1 started  arr  1  remain 12  wait 0*

*at time 2 process 1 stopped  arr  1  remain 11  wait 0*

*at time 2 process 1 resumed  arr  1  remain 11  wait 0*

*at time 3 process 1 stopped  arr  1  remain 10  wait 0*

*at time 3 process 1 resumed  arr  1  remain 10  wait 0*

*at time 4 process 1 stopped  arr  1  remain 9  wait 0*

*at time 4 process 1 resumed  arr  1  remain 9  wait 0*

*at time 5 process 1 stopped  arr  1  remain 8  wait 0*

*at time 5 process 1 resumed  arr  1  remain 8  wait 0*

*at time 6 process 1 stopped  arr  1  remain 7  wait 0*

*at time 6 process 1 resumed  arr  1  remain 7  wait 0*

*at time 7 process 1 stopped  arr  1  remain 6  wait 0*

*at time 7 process 1 resumed  arr  1  remain 6  wait 0*

*at time 8 process 1 stopped  arr  1  remain 5  wait 0*

*at time 8 process 1 resumed  arr  1  remain 5  wait 0*

*at time 9 process 1 stopped  arr  1  remain 4  wait 0*

*at time 9 process 1 resumed  arr  1  remain 4  wait 0*

*at time 10 process 1 stopped  arr  1  remain 3  wait 0*

*at time 10 process 1 resumed  arr  1  remain 3  wait 0*

*at time 11 process 1 stopped  arr  1  remain 2  wait 0*

*at time 11 process 1 resumed  arr  1  remain 2  wait 0*

*at time 12 process 1 stopped  arr  1  remain 1  wait 0*

*at time 12 process 2 started  arr  12  remain 7  wait 0*

*at time 13 process 2 stopped  arr  12  remain 6  wait 0*

*at time 13 process 2 resumed  arr  12  remain 6  wait 0*

*at time 14 process 2 stopped  arr  12  remain 5  wait 0*

*at time 14 process 2 resumed  arr  12  remain 5  wait 0*

*at time 15 process 2 stopped  arr  12  remain 4  wait 0*

*at time 15 process 2 resumed  arr  12  remain 4  wait 0*

*at time 16 process 2 stopped  arr  12  remain 3  wait 0*

*at time 16 process 3 started  arr  16  remain 4  wait 0*

*at time 17 process 3 stopped  arr  16  remain 3  wait 0*

*at time 17 process 3 resumed  arr  16  remain 3  wait 0*

*at time 18 process 3 stopped  arr  16  remain 2  wait 0*

*at time 18 process 3 resumed  arr  16  remain 2  wait 0*

*at time 19 process 3 stopped  arr  16  remain 1  wait 0*

*at time 19 process 3 resumed  arr  16  remain 1  wait 0*

*at time 20 process 3 finished  arr  16  remain 0  wait 0  TA 4  WTA 1*

*at time 20 process 2 resumed  arr  12  remain 3  wait 4*

*at time 21 process 2 stopped  arr  12  remain 2  wait 4*

*at time 21 process 2 resumed  arr  12  remain 2  wait 4*

*at time 22 process 2 stopped  arr  12  remain 1  wait 4*

*at time 22 process 1 resumed  arr  1  remain 1  wait 10*

*at time 23 process 1 finished  arr  1  remain 0  wait 10  TA 22  WTA 1*

*at time 23 process 2 resumed  arr  12  remain 1  wait 5*

*at time 24 process 2 finished  arr  12  remain 0  wait 5  TA 12  WTA 1*

*at time 25 process 5 started  arr  25  remain 11  wait 0*

*at time 26 process 5 stopped  arr  25  remain 10  wait 0*

*at time 26 process 4 started  arr  25  remain 3  wait 0*

*at time 27 process 4 stopped  arr  25  remain 2  wait 0*

*at time 27 process 5 resumed  arr  25  remain 10  wait 1*

*at time 28 process 5 stopped  arr  25  remain 9  wait 1*

*at time 28 process 4 resumed  arr  25  remain 2  wait 2*

*at time 29 process 4 stopped  arr  25  remain 1  wait 2*

*at time 29 process 5 resumed  arr  25  remain 9  wait 2*

*at time 30 process 5 stopped  arr  25  remain 8  wait 2*

*at time 30 process 4 resumed  arr  25  remain 1  wait 3*

*at time 31 process 4 finished  arr  25  remain 0  wait 3  TA 6  WTA 2*

*at time 31 process 5 resumed  arr  25  remain 8  wait 3*

*at time 32 process 5 stopped  arr  25  remain 7  wait 3*

*at time 32 process 5 resumed  arr  25  remain 7  wait 3*

*at time 33 process 5 stopped  arr  25  remain 6  wait 3*

*at time 33 process 5 resumed  arr  25  remain 6  wait 3*

*at time 34 process 5 stopped  arr  25  remain 5  wait 3*

*at time 34 process 5 resumed  arr  25  remain 5  wait 3*

*at time 35 process 5 stopped  arr  25  remain 4  wait 3*

*at time 35 process 5 resumed  arr  25  remain 4  wait 3*

*at time 36 process 5 stopped  arr  25  remain 3  wait 3*

*at time 36 process 5 resumed  arr  25  remain 3  wait 3*

*at time 37 process 5 stopped  arr  25  remain 2  wait 3*

*at time 37 process 5 resumed  arr  25  remain 2  wait 3*

*at time 38 process 5 stopped  arr  25  remain 1  wait 3*

*at time 38 process 5 resumed  arr  25  remain 1  wait 3*

*at time 39 process 5 finished  arr  25  remain 0  wait 3  TA 14  WTA 1*

cpu uti: 94.871796

WTA= 1

Avg Waiting= 4

*Assumptions:* the quantum is not changed

**Work Load Distribution:**

|  |  |
| --- | --- |
| Mahinour Alaa | SJF/ process generator/ Scheduler |
| Yasmin Hashem | RR/ process generator/queue/ Scheduler |
| Yahia Ahmed | MLFQ/ process generator/ Scheduler |
| Youssef Khaled | PHPF/ process generator/process/ Scheduler/Structs |