Linnéuniversitetet
Kalmar Växjö

Linnaeus University

Faculty of Technology – Department of Computer Science

1DV512 – Operating Systems
Individual Assignment 2

Assignment Number: 2
Student Name: Mohammadali Rashidfarokhi
Student Email: mr223jp@student.lnu.se
Course Code: 1DV512
Semester: Autumn 2020

Operating Systems Individual Assignment2

Task 1: Comparison of FreeBSD ULE and Linux CFS Schedulers

1. Does ULE support threads, or does it support processes only? How about CFS?

When it comes to FreeBSD, ULE is considered as the default scheduler. On the other hand, the CFS is the Linux scheduler. Due to the transporting of the ULE to the Linux, its usage can be seen in scheduling all threads that are planned by the CFS. By the comparison of the large suite of applications on the adapted kernel running CFS. The outcome of the comparisons was the result of the scheduling decisions and they are not pointing to the differences in other sub-systems between FreeBSD and Linux.

An interesting point about the CFS is the more complicated the load balancing mechanism of CFS is, the more quickly it counters to the workload transformations. Also, ULE and CFS are both manufactured to schedule large numbers of threads on well-built multi-core machines.

Regarding the processes, one of the most distinguishable challenges in comparing ULE and CFS would be that the application functioning depends on the scheduler and other OS subsystems that are known as networking, file system, and memory management. For the separation of the CFS and ULE effects, ULE was moved to Linux and it has been used as the default scheduler to execute all threads on the machine. So, all in all, SFS will support both threads and processes.

2. How does CFS select the next task to be executed?

Regarding the process of selecting the next task for the CFS, threads with the same priority and same importance and similar “vruntime” have been executed with the same amount of time. What I mean by this is that the core resources have been split properly between the threads. For making sure about all “vruntime” of threads advance properly. As soon as the current executing thread is anticipated, CFS schedules the thread with the least “vruntime “to be executed next. Moving on, since the CFS schedules the thread with the littlest “vruntime”, CFC is required to avoid a thread from having a “vruntime” much less than the “vruntimes” of other threads waiting for being scheduled. In that case, the lowest vruntime of a thread can be executed for a long period. Moving on, when a thread has been created, the thread will initiate with a vruntime equal to the highest vruntime of the threads that are standing by in the runqueue. In case of the awakening of the threads after the sleeping, its vruntime will be modernized to be equal to the least vruntime of the threads that are standing by to

be scheduled. Also, by taking advantage of the least vruntime, it will guarantee that the threads that snooze too much are arranged fist.

3. What is a cgroup and how is it used by CFS? Does ULE support cgroups?

Back then before the evolution of the Linux 2.6.38, all the thread was referred to as a self-governing entity and it received the same distribution of resources as other threads in the system. What I mean by this is that a single application that has taken advantage of numerous threads has a bigger distribution of resources than a solitary thread application. When it comes to the up to date version of the kernel, the threads that belong to the same application are categorized into a structure known as cgroup.
Cgroup is the short version of the control groups. They are a Linux kernel feature that will permit the processes to be scheduled into hierarchical groups whose function can be restricted and be watched. The process of the grouping will be applied in the core cgroup kernel code. On the other hand, the resource chasing process and limits will occur in a set of pre-resource-type subsystems.
All in all, cgroup is known as a collection of processes that are restricted to a set of restrictions or parameters describe via the cgroup filesystem.

One cgroup has a vruntime that aggress to the sum of the vruntime of its all threads. Consequently,
Justice between groups of threads. When the execution of the selected cgroup has been organized, its thread with the least vruntime will be performed for making sure of the equality within a cgroup.

Regarding the ULE, it will not support the cgroup. What I mean by this is that different from the CFS, ULE will not categorize the threads into cgroups. On the contrary, it will mention each thread as a solitary entity.

4. How many queues in total ULE use? What is the purpose of each queue?

Regarding the ULE, it is using two runqueues to schedule threads. Firstly, interactive threads exist in one runqueue. The next usage can be seen in the batch threads. Lastly, a runqueue known as idle will be used when a core is idle and this runqueue contains the idle task.
The purpose of taking advantage of two runqueuse is to give importance to interactive threads. However, the execution of the Batch processes usually takes a place without user interaction. Therefore, the scheduling latency will not be considered valuable. For keeping an eye on the interactivity of a thread, ULE will use an interactivity penalty metric from 0 to 100. Regarding the idle queue, the priority threads will be executed when there is no sign of time distribution or current threads to be executed. So basically, the idle queue can save all the idle threads. All in all, by using these three queues, it will make it achievable to implement processor affinity in an SMP system.

5. How does ULE compute priority for various tasks?

To be able to categorize the threads, the first thing the ULE does is to calculate a mark known as interactivity penalty + niceness. That is, A thread is referred to as an interactive if its mark is less than a level 30 as default in FreeBSD 11.1. Since by having a niceness value of 0, it will resemble nearly dedicating more than 60% of the time sleeping. If that is not the case it will be referred to as batch.
A minus nice mark (high priority) makes it simpler for a thread to be interactive. At the creation of a thread, it will inherit the runtime and the time for sleeping of its parents. In case of dying a thread, its runtime in the final 5 seconds will be given back to its parents.

Regarding the interactive and batch run queues, threads have been saved by the priority. The way for setting the priority for interactive threads will be a linear interpolation of their mark (e.g. a thread with 0 penalties will be given the highest priority, and a thread having 30 penalties is having the lowest priority). Interactive run queues are having FIFO per one priority. The process of choosing a thread is done by running from the runqueue and it will be done by choosing the initial thread in the highest-priority non-empty FIFO. However, the importance of the batch threads is reliant on their runtime.
What I mean by this is that the more a thread runs, the lower its priority will be. For choosing the upcoming thread to be run, ULE initially seeks in the interactive runqueues. In case of a thread being ready to be organized, it will return that thread. If it was empty, ULE will look in the batch runqueue rather than interactive runqueue. But if both are not containing anything, the core will be idle, and the thread will be organized.

When it comes to the execution of a single thread by the core, the time slice will be 10 ticks (78ms).
But for the more than one thread the score will be divided by the number of threads that were limited to a lower range of 1 tick (1/127th of a second). Also, full preemption has been deactivated.

For making it clearer the following equation will be used.

$$scaling\ factor = m = 50$$

$$penalty(r,s) = \begin{cases} \frac{m}{\frac{s}{r}} & s > r \\ \frac{m}{\frac{r}{s}} + m & otherwise \end{cases}$$

Regarding the penalty, the definition of the penalty when it has a range of less than 51, means that a thread has dedicated more time voluntarily sleeping than executing. On the other hand, the situation for the penalty above 51 is different.

6. Do CFS and ULE support task preemption? Are there any limitations?

Regarding the preemption, both ULE and CFS will support task preemption. However, CFS's full preemption is activated so the CFS will preempt the running tasks. On the other hand, the ULE has no full preemption by default. Therefore, just the kernel threads can preempt others. As I have mentioned before, the full preemption in ULE is deactivated this will happen at the same time when CFS preempts the running thread exactly when the thread has been wakened up it contains vruntime which is tinier than the vruntime of the present time running thread. The ab in the CFS will be preempted 2 million times throughout the benchmark because it will never be preempted with ULE. What I mean by this is that ab will begin by spending 100 requests to the httpd server, then it will stand by for the server to respond. Once the ab is awakened, it will analyze which requests have been taken care of then it will send new requests to the server. Due to the reason that the ab is considered single-threaded, the entire requests that have been sent to the server have been sent sequentially. Regarding the ULE, ab has the privilege to send as many new requests as it has received an answer. Every request in CFS that has been sent by ab, will wake up a httpd thread that can preempt ab.

7. Did Bouron et al. discover large differences in per-core scheduling performance between CFS and ULE? Which definition of "performance" did they use in their benchmark, and why?

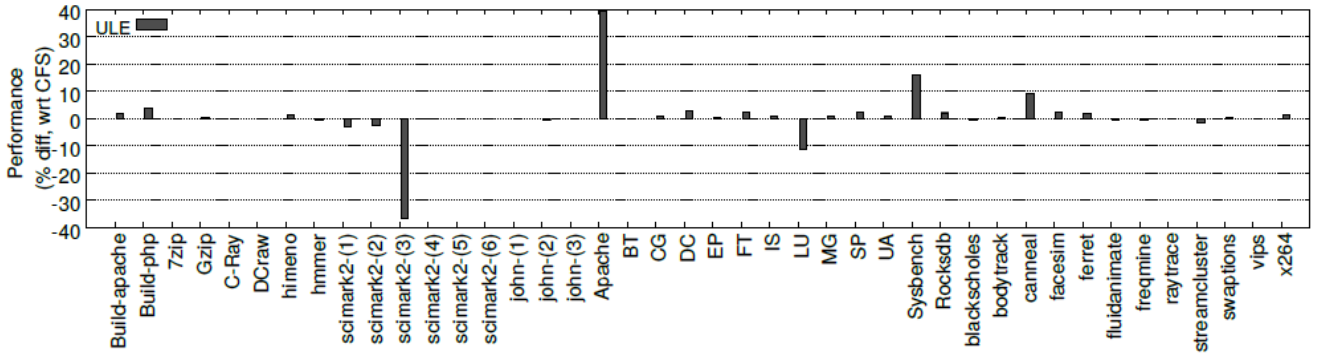
Based on the discoveries from Bouron, a significant difference between CFS and ULE in pre-core scheduling is in managing of batch threads. on the one hand, the CFS is attempting to be fair to the whole threads. on the other hand, ULE will award priority to interactive threads. To be able to evaluate the influence of design decisions that happened by co-scheduling a batch and an interactive application which is on the same core, it can be illustrated under ULE batch applications can starve for an unbound amount of time. However, starvation under ULE still can happen even if a single application is executing on the system.

Regarding the per-core scheduling, the impact on the performance of the 37 applications will be examined thoroughly.

When it comes to the definition of the performance, in terms of database jobs and NSA applications, the number of operations will be compared per second. On the hand, the rest of the application only 1/ex-execution time will be compared. Consequently, the higher the performance is, better performance a scheduler will have.

Moving on, the scheduler has shown a little impact on the majority of workloads. Also, most of the applications will use threads that all carry out the same job. Therefore, CFS and ULE will finish-up be scheduling the whole threads in a round-robin fashion.

The below picture will illustrate that the application will execute faster with ULE rather than CFS with the rates over 0.



8. What is the difference between the multi-core load balancing strategies used by CFS and ULE? Is any of them faster? Does any of them typically reach perfect load balancing?

When it comes to the differences between the CFS and ULE, the average performance between CFS and ULE is small. That is, 2.75% in favor of ULE.
MG, which is coming from the NSA benchmark suite, will take advantage mostly of ULE's load balancing strategy the reason is that it is 73% quicker functionality on ULE rather than CFS. MG can spawn as many threads as there are cores in the machine. Also, the entire threads to act as the same calculation. Once a thread has stopped its calculations, it will stand by on a spin-barrier for 100ms then it will sleep if it figures out there are other threads computing. Moving on, the arrangement of the ULE is like precisely put one thread for each core, and then under no circumstances, it will migrate them one more time. One interesting point about the threads is that they will be given a short amount of time standing by each other in the barriers and you can not see them sleeping. On the other side, CFS will show a response to tiny transformations in a load of cores. Therefore, occasionally by the accident, it will put two MG threads on the same core. Due to the reason that MG will take advantage of barriers, the two threads organized on the same core will result in long waiting for the entire application. The postponement is considered 50% due to the reason that threads are organized on their own on their cores when it goes to sleep but they should wake up so that the latency can be added to the obstacles.
Regarding the speed of ULE and CFS, because the load is always balanced in ULE, it requires more than 11 seconds for ULE to prepare all the threads ready to be runnable. However, the same operation will take two seconds for CFS. This lateness can be further discussed by starvation.
Lastly, ULE is suitable and will reach a perfect balance. Although CFS is perfectly suited for solving large inequity loads in the system, it will not act perfectly when a perfect balance is critical for performance. Also, ULE will always divide the threads

on the core with the least number of threads. Therefore, the load is supposed to be well balanced from the scratch.

Task 2: Developing a Scheduling Algorithm in Java

Based on the following screenshots, when it comes to the preemptive, the average waiting time is considerably higher than the non-Preemptive average waiting time. What I mean by this is that by taking average of all the simulation results of preemptive, the outcome would be 30.2. However, the result for the non-preemptive simulation is 22.8.

Simulation identifier	Average waiting time	
	Non-preemptive	Preemptive
0	28	41
1	30	36
2	19	24
3	20	26
4	17	24
Average	22.8	30.2

Simulation identifier	Simulation execution time	
	Non-preemptive	Preemptive
0	71	71
1	75	67
2	57	49
3	62	52
4	49	56
Average	62.8	59

```
Running non-preemptive simulation #0
Simulation results:
----------------------
process Id => 1 ,BurstTime => 8 ,arrivalMoment => 0 ,totalWaitingTime => 35
process Id => 2 ,BurstTime => 3 ,arrivalMoment => 0 ,totalWaitingTime => 0
process Id => 3 ,BurstTime => 10 ,arrivalMoment => 1 ,totalWaitingTime => 19
process Id => 4 ,BurstTime => 9 ,arrivalMoment => 1 ,totalWaitingTime => 10
process Id => 5 ,BurstTime => 8 ,arrivalMoment => 2 ,totalWaitingTime => 1
process Id => 6 ,BurstTime => 9 ,arrivalMoment => 2 ,totalWaitingTime => 60
process Id => 7 ,BurstTime => 8 ,arrivalMoment => 3 ,totalWaitingTime => 40
process Id => 8 ,BurstTime => 4 ,arrivalMoment => 3 ,totalWaitingTime => 48
process Id => 9 ,BurstTime => 5 ,arrivalMoment => 4 ,totalWaitingTime => 26
process Id => 10 ,BurstTime => 7 ,arrivalMoment => 5 ,totalWaitingTime => 50
**********
Complete execution time => 71
Average process Waiting time => 28


Running non-preemptive simulation #1
Simulation results:
----------------------
process Id => 1 ,BurstTime => 3 ,arrivalMoment => 0 ,totalWaitingTime => 0
process Id => 2 ,BurstTime => 6 ,arrivalMoment => 0 ,totalWaitingTime => 38
process Id => 3 ,BurstTime => 6 ,arrivalMoment => 1 ,totalWaitingTime => 43
process Id => 4 ,BurstTime => 8 ,arrivalMoment => 1 ,totalWaitingTime => 2
process Id => 5 ,BurstTime => 7 ,arrivalMoment => 2 ,totalWaitingTime => 66
process Id => 6 ,BurstTime => 10 ,arrivalMoment => 2 ,totalWaitingTime => 9
process Id => 7 ,BurstTime => 7 ,arrivalMoment => 3 ,totalWaitingTime => 18
process Id => 8 ,BurstTime => 9 ,arrivalMoment => 3 ,totalWaitingTime => 47
process Id => 9 ,BurstTime => 10 ,arrivalMoment => 4 ,totalWaitingTime => 24
process Id => 10 ,BurstTime => 9 ,arrivalMoment => 5 ,totalWaitingTime => 54
**********
Complete execution time => 75
Average process Waiting time => 30
```

```
Running non-preemptive simulation #2
Simulation results:
----------------------
process Id => 1 ,BurstTime => 3 ,arrivalMoment => 0 ,totalWaitingTime => 29
process Id => 2 ,BurstTime => 4 ,arrivalMoment => 0 ,totalWaitingTime => 0
process Id => 3 ,BurstTime => 6 ,arrivalMoment => 1 ,totalWaitingTime => 3
process Id => 4 ,BurstTime => 2 ,arrivalMoment => 1 ,totalWaitingTime => 9
process Id => 5 ,BurstTime => 9 ,arrivalMoment => 2 ,totalWaitingTime => 30
process Id => 6 ,BurstTime => 7 ,arrivalMoment => 2 ,totalWaitingTime => 10
process Id => 7 ,BurstTime => 2 ,arrivalMoment => 3 ,totalWaitingTime => 16
process Id => 8 ,BurstTime => 8 ,arrivalMoment => 4 ,totalWaitingTime => 17
process Id => 9 ,BurstTime => 6 ,arrivalMoment => 4 ,totalWaitingTime => 37
process Id => 10 ,BurstTime => 10 ,arrivalMoment => 5 ,totalWaitingTime => 42
**********
Complete execution time => 57
Average process Waiting time => 19


Running non-preemptive simulation #3
Simulation results:
----------------------
process Id => 1 ,BurstTime => 10 ,arrivalMoment => 0 ,totalWaitingTime => 52
process Id => 2 ,BurstTime => 5 ,arrivalMoment => 0 ,totalWaitingTime => 0
process Id => 3 ,BurstTime => 4 ,arrivalMoment => 1 ,totalWaitingTime => 4
process Id => 4 ,BurstTime => 7 ,arrivalMoment => 1 ,totalWaitingTime => 11
process Id => 5 ,BurstTime => 4 ,arrivalMoment => 2 ,totalWaitingTime => 17
process Id => 6 ,BurstTime => 7 ,arrivalMoment => 2 ,totalWaitingTime => 24
process Id => 7 ,BurstTime => 9 ,arrivalMoment => 3 ,totalWaitingTime => 40
process Id => 8 ,BurstTime => 10 ,arrivalMoment => 3 ,totalWaitingTime => 30
process Id => 9 ,BurstTime => 3 ,arrivalMoment => 4 ,totalWaitingTime => 19
process Id => 10 ,BurstTime => 3 ,arrivalMoment => 5 ,totalWaitingTime => 4
**********
Complete execution time => 62
Average process Waiting time => 20
```

```
Running non-preemptive simulation #4
Simulation results:
---------------------
process Id => 1 ,BurstTime => 3 ,arrivalMoment => 0 ,totalWaitingTime => 31
process Id => 2 ,BurstTime => 2 ,arrivalMoment => 0 ,totalWaitingTime => 0
process Id => 3 ,BurstTime => 2 ,arrivalMoment => 1 ,totalWaitingTime => 46
process Id => 4 ,BurstTime => 2 ,arrivalMoment => 1 ,totalWaitingTime => 1
process Id => 5 ,BurstTime => 4 ,arrivalMoment => 2 ,totalWaitingTime => 2
process Id => 6 ,BurstTime => 9 ,arrivalMoment => 2 ,totalWaitingTime => 36
process Id => 7 ,BurstTime => 9 ,arrivalMoment => 3 ,totalWaitingTime => 9
process Id => 8 ,BurstTime => 4 ,arrivalMoment => 3 ,totalWaitingTime => 5
process Id => 9 ,BurstTime => 10 ,arrivalMoment => 4 ,totalWaitingTime => 17
process Id => 10 ,BurstTime => 4 ,arrivalMoment => 4 ,totalWaitingTime => 30
**********
Complete execution time => 49
Average process Waiting time => 17
```

```
Running preemptive simulation #0
Simulation results:
---------------------
process Id => 1 ,BurstTime => 7 ,arrivalMoment => 0 ,totalWaitingTime => 63
process Id => 2 ,BurstTime => 8 ,arrivalMoment => 0 ,totalWaitingTime => 34
process Id => 3 ,BurstTime => 6 ,arrivalMoment => 1 ,totalWaitingTime => 29
process Id => 4 ,BurstTime => 8 ,arrivalMoment => 1 ,totalWaitingTime => 25
process Id => 5 ,BurstTime => 7 ,arrivalMoment => 2 ,totalWaitingTime => 31
process Id => 6 ,BurstTime => 3 ,arrivalMoment => 2 ,totalWaitingTime => 24
process Id => 7 ,BurstTime => 6 ,arrivalMoment => 3 ,totalWaitingTime => 59
process Id => 8 ,BurstTime => 9 ,arrivalMoment => 3 ,totalWaitingTime => 59
process Id => 9 ,BurstTime => 8 ,arrivalMoment => 4 ,totalWaitingTime => 51
process Id => 10 ,BurstTime => 9 ,arrivalMoment => 5 ,totalWaitingTime => 43
**********
Complete execution time => 71
Average process Waiting time => 41


Running preemptive simulation #1
Simulation results:
---------------------
process Id => 1 ,BurstTime => 8 ,arrivalMoment => 0 ,totalWaitingTime => 47
process Id => 2 ,BurstTime => 8 ,arrivalMoment => 0 ,totalWaitingTime => 46
process Id => 3 ,BurstTime => 4 ,arrivalMoment => 1 ,totalWaitingTime => 4
process Id => 4 ,BurstTime => 4 ,arrivalMoment => 1 ,totalWaitingTime => 22
process Id => 5 ,BurstTime => 7 ,arrivalMoment => 2 ,totalWaitingTime => 55
process Id => 6 ,BurstTime => 10 ,arrivalMoment => 2 ,totalWaitingTime => 54
process Id => 7 ,BurstTime => 9 ,arrivalMoment => 3 ,totalWaitingTime => 55
process Id => 8 ,BurstTime => 4 ,arrivalMoment => 3 ,totalWaitingTime => 14
process Id => 9 ,BurstTime => 5 ,arrivalMoment => 4 ,totalWaitingTime => 29
process Id => 10 ,BurstTime => 8 ,arrivalMoment => 6 ,totalWaitingTime => 35
**********
Complete execution time => 67
Average process Waiting time => 36
```

```
Running preemptive simulation #2
Simulation results:
---------------------
process Id => 1 ,BurstTime => 3 ,arrivalMoment => 0 ,totalWaitingTime => 28
process Id => 2 ,BurstTime => 2 ,arrivalMoment => 0 ,totalWaitingTime => 0
process Id => 3 ,BurstTime => 7 ,arrivalMoment => 1 ,totalWaitingTime => 25
process Id => 4 ,BurstTime => 4 ,arrivalMoment => 1 ,totalWaitingTime => 13
process Id => 5 ,BurstTime => 6 ,arrivalMoment => 2 ,totalWaitingTime => 34
process Id => 6 ,BurstTime => 4 ,arrivalMoment => 2 ,totalWaitingTime => 32
process Id => 7 ,BurstTime => 9 ,arrivalMoment => 3 ,totalWaitingTime => 36
process Id => 8 ,BurstTime => 2 ,arrivalMoment => 3 ,totalWaitingTime => 17
process Id => 9 ,BurstTime => 5 ,arrivalMoment => 4 ,totalWaitingTime => 40
process Id => 10 ,BurstTime => 7 ,arrivalMoment => 5 ,totalWaitingTime => 23
**********
Complete execution time => 49
Average process Waiting time => 24


Running preemptive simulation #3
Simulation results:
---------------------
process Id => 1 ,BurstTime => 8 ,arrivalMoment => 0 ,totalWaitingTime => 40
process Id => 2 ,BurstTime => 4 ,arrivalMoment => 0 ,totalWaitingTime => 39
process Id => 3 ,BurstTime => 6 ,arrivalMoment => 1 ,totalWaitingTime => 2
process Id => 4 ,BurstTime => 6 ,arrivalMoment => 1 ,totalWaitingTime => 45
process Id => 5 ,BurstTime => 2 ,arrivalMoment => 2 ,totalWaitingTime => 21
process Id => 6 ,BurstTime => 9 ,arrivalMoment => 2 ,totalWaitingTime => 27
process Id => 7 ,BurstTime => 6 ,arrivalMoment => 3 ,totalWaitingTime => 32
process Id => 8 ,BurstTime => 4 ,arrivalMoment => 3 ,totalWaitingTime => 10
process Id => 9 ,BurstTime => 2 ,arrivalMoment => 4 ,totalWaitingTime => 15
process Id => 10 ,BurstTime => 5 ,arrivalMoment => 4 ,totalWaitingTime => 36
**********
Complete execution time => 52
Average process Waiting time => 26
```

```
Running preemptive simulation #4
Simulation results:
---------------------
process Id => 1 ,BurstTime => 3 ,arrivalMoment => 0 ,totalWaitingTime => 8
process Id => 2 ,BurstTime => 4 ,arrivalMoment => 0 ,totalWaitingTime => 21
process Id => 3 ,BurstTime => 3 ,arrivalMoment => 1 ,totalWaitingTime => 23
process Id => 4 ,BurstTime => 4 ,arrivalMoment => 1 ,totalWaitingTime => 2
process Id => 5 ,BurstTime => 6 ,arrivalMoment => 2 ,totalWaitingTime => 23
process Id => 6 ,BurstTime => 8 ,arrivalMoment => 2 ,totalWaitingTime => 32
process Id => 7 ,BurstTime => 6 ,arrivalMoment => 3 ,totalWaitingTime => 45
process Id => 8 ,BurstTime => 10 ,arrivalMoment => 3 ,totalWaitingTime => 43
process Id => 9 ,BurstTime => 3 ,arrivalMoment => 4 ,totalWaitingTime => 19
process Id => 10 ,BurstTime => 9 ,arrivalMoment => 4 ,totalWaitingTime => 25
**********
Complete execution time => 56
Average process Waiting time => 24
```

Running non-preemptive simulation #0
Simulation results:
---
process Id => 1 ,BurstTime => 8 ,arrivalMoment => 0 ,totalWaitingTime => 35
process Id => 2 ,BurstTime => 3 ,arrivalMoment => 0 ,totalWaitingTime => 0
process Id => 3 ,BurstTime => 10 ,arrivalMoment => 1 ,totalWaitingTime => 19
process Id => 4 ,BurstTime => 9 ,arrivalMoment => 1 ,totalWaitingTime => 10
process Id => 5 ,BurstTime => 8 ,arrivalMoment => 2 ,totalWaitingTime => 1
process Id => 6 ,BurstTime => 9 ,arrivalMoment => 2 ,totalWaitingTime => 60
process Id => 7 ,BurstTime => 8 ,arrivalMoment => 3 ,totalWaitingTime => 40
process Id => 8 ,BurstTime => 4 ,arrivalMoment => 3 ,totalWaitingTime => 48
process Id => 9 ,BurstTime => 5 ,arrivalMoment => 4 ,totalWaitingTime => 26
process Id => 10 ,BurstTime => 7 ,arrivalMoment => 5 ,totalWaitingTime => 50
****
Complete execution time => 71
Average process Waiting time => 28

Running non-preemptive simulation #1
Simulation results:
---
process Id => 1 ,BurstTime => 3 ,arrivalMoment => 0 ,totalWaitingTime => 0
process Id => 2 ,BurstTime => 6 ,arrivalMoment => 0 ,totalWaitingTime => 38
process Id => 3 ,BurstTime => 6 ,arrivalMoment => 1 ,totalWaitingTime => 43
process Id => 4 ,BurstTime => 8 ,arrivalMoment => 1 ,totalWaitingTime => 2
process Id => 5 ,BurstTime => 7 ,arrivalMoment => 2 ,totalWaitingTime => 66
process Id => 6 ,BurstTime => 10 ,arrivalMoment => 2 ,totalWaitingTime => 9
process Id => 7 ,BurstTime => 7 ,arrivalMoment => 3 ,totalWaitingTime => 18
process Id => 8 ,BurstTime => 9 ,arrivalMoment => 3 ,totalWaitingTime => 47
process Id => 9 ,BurstTime => 10 ,arrivalMoment => 4 ,totalWaitingTime => 24
process Id => 10 ,BurstTime => 9 ,arrivalMoment => 5 ,totalWaitingTime => 54
****
Complete execution time => 75
Average process Waiting time => 30

Running non-preemptive simulation #2
Simulation results:
---
process Id => 1 ,BurstTime => 3 ,arrivalMoment => 0 ,totalWaitingTime => 29
process Id => 2 ,BurstTime => 4 ,arrivalMoment => 0 ,totalWaitingTime => 0
process Id => 3 ,BurstTime => 6 ,arrivalMoment => 1 ,totalWaitingTime => 3
process Id => 4 ,BurstTime => 2 ,arrivalMoment => 1 ,totalWaitingTime => 9
process Id => 5 ,BurstTime => 9 ,arrivalMoment => 2 ,totalWaitingTime => 30
process Id => 6 ,BurstTime => 7 ,arrivalMoment => 2 ,totalWaitingTime => 10
process Id => 7 ,BurstTime => 2 ,arrivalMoment => 3 ,totalWaitingTime => 16
process Id => 8 ,BurstTime => 8 ,arrivalMoment => 4 ,totalWaitingTime => 17
process Id => 9 ,BurstTime => 6 ,arrivalMoment => 4 ,totalWaitingTime => 37
process Id => 10 ,BurstTime => 10 ,arrivalMoment => 5 ,totalWaitingTime => 42
****
Complete execution time => 57
Average process Waiting time => 19

Running non-preemptive simulation #3
Simulation results:
---
process Id => 1 ,BurstTime => 10 ,arrivalMoment => 0 ,totalWaitingTime => 52
process Id => 2 ,BurstTime => 5 ,arrivalMoment => 0 ,totalWaitingTime => 0
process Id => 3 ,BurstTime => 4 ,arrivalMoment => 1 ,totalWaitingTime => 4
process Id => 4 ,BurstTime => 7 ,arrivalMoment => 1 ,totalWaitingTime => 11
process Id => 5 ,BurstTime => 4 ,arrivalMoment => 2 ,totalWaitingTime => 17
process Id => 6 ,BurstTime => 7 ,arrivalMoment => 2 ,totalWaitingTime => 24
process Id => 7 ,BurstTime => 9 ,arrivalMoment => 3 ,totalWaitingTime => 40
process Id => 8 ,BurstTime => 10 ,arrivalMoment => 3 ,totalWaitingTime => 30
process Id => 9 ,BurstTime => 3 ,arrivalMoment => 4 ,totalWaitingTime => 19
process Id => 10 ,BurstTime => 3 ,arrivalMoment => 5 ,totalWaitingTime => 4
****
Complete execution time => 62
Average process Waiting time => 20

Running non-preemptive simulation #4
Simulation results:
---
process Id => 1 ,BurstTime => 3 ,arrivalMoment => 0 ,totalWaitingTime => 31
process Id => 2 ,BurstTime => 2 ,arrivalMoment => 0 ,totalWaitingTime => 0
process Id => 3 ,BurstTime => 2 ,arrivalMoment => 1 ,totalWaitingTime => 46
process Id => 4 ,BurstTime => 2 ,arrivalMoment => 1 ,totalWaitingTime => 1
process Id => 5 ,BurstTime => 4 ,arrivalMoment => 2 ,totalWaitingTime => 2
process Id => 6 ,BurstTime => 9 ,arrivalMoment => 2 ,totalWaitingTime => 36
process Id => 7 ,BurstTime => 9 ,arrivalMoment => 3 ,totalWaitingTime => 9
process Id => 8 ,BurstTime => 4 ,arrivalMoment => 3 ,totalWaitingTime => 5
process Id => 9 ,BurstTime => 10 ,arrivalMoment => 4 ,totalWaitingTime => 17
process Id => 10 ,BurstTime => 4 ,arrivalMoment => 4 ,totalWaitingTime => 30
****
Complete execution time => 49
Average process Waiting time => 17

Running preemptive simulation #0
Simulation results:
---
process Id => 1 ,BurstTime => 7 ,arrivalMoment => 0 ,totalWaitingTime => 63
process Id => 2 ,BurstTime => 8 ,arrivalMoment => 0 ,totalWaitingTime => 34
process Id => 3 ,BurstTime => 6 ,arrivalMoment => 1 ,totalWaitingTime => 29
process Id => 4 ,BurstTime => 8 ,arrivalMoment => 1 ,totalWaitingTime => 25
process Id => 5 ,BurstTime => 7 ,arrivalMoment => 2 ,totalWaitingTime => 31
process Id => 6 ,BurstTime => 3 ,arrivalMoment => 2 ,totalWaitingTime => 24
process Id => 7 ,BurstTime => 6 ,arrivalMoment => 3 ,totalWaitingTime => 59
process Id => 8 ,BurstTime => 9 ,arrivalMoment => 3 ,totalWaitingTime => 59
process Id => 9 ,BurstTime => 8 ,arrivalMoment => 4 ,totalWaitingTime => 51
process Id => 10 ,BurstTime => 9 ,arrivalMoment => 5 ,totalWaitingTime => 43
****
Complete execution time => 71
Average process Waiting time => 41

Running preemptive simulation #1
Simulation results:
---
process Id => 1 ,BurstTime => 8 ,arrivalMoment => 0 ,totalWaitingTime => 47
process Id => 2 ,BurstTime => 8 ,arrivalMoment => 0 ,totalWaitingTime => 46
process Id => 3 ,BurstTime => 4 ,arrivalMoment => 1 ,totalWaitingTime => 4
process Id => 4 ,BurstTime => 4 ,arrivalMoment => 1 ,totalWaitingTime => 22
process Id => 5 ,BurstTime => 7 ,arrivalMoment => 2 ,totalWaitingTime => 55
process Id => 6 ,BurstTime => 10 ,arrivalMoment => 2 ,totalWaitingTime => 54
process Id => 7 ,BurstTime => 9 ,arrivalMoment => 3 ,totalWaitingTime => 55
process Id => 8 ,BurstTime => 4 ,arrivalMoment => 3 ,totalWaitingTime => 14
process Id => 9 ,BurstTime => 5 ,arrivalMoment => 4 ,totalWaitingTime => 29
process Id => 10 ,BurstTime => 8 ,arrivalMoment => 6 ,totalWaitingTime => 35
****
Complete execution time => 67
Average process Waiting time => 36

Running preemptive simulation #2
Simulation results:
---
process Id => 1 ,BurstTime => 3 ,arrivalMoment => 0 ,totalWaitingTime => 28
process Id => 2 ,BurstTime => 2 ,arrivalMoment => 0 ,totalWaitingTime => 0
process Id => 3 ,BurstTime => 7 ,arrivalMoment => 1 ,totalWaitingTime => 25
process Id => 4 ,BurstTime => 4 ,arrivalMoment => 1 ,totalWaitingTime => 13
process Id => 5 ,BurstTime => 6 ,arrivalMoment => 2 ,totalWaitingTime => 34
process Id => 6 ,BurstTime => 4 ,arrivalMoment => 2 ,totalWaitingTime => 32
process Id => 7 ,BurstTime => 9 ,arrivalMoment => 3 ,totalWaitingTime => 36
process Id => 8 ,BurstTime => 2 ,arrivalMoment => 3 ,totalWaitingTime => 17
process Id => 9 ,BurstTime => 5 ,arrivalMoment => 4 ,totalWaitingTime => 40
process Id => 10 ,BurstTime => 7 ,arrivalMoment => 5 ,totalWaitingTime => 23
****
Complete execution time => 49
Average process Waiting time => 24

Running preemptive simulation #3
Simulation results:
---
process Id => 1 ,BurstTime => 8 ,arrivalMoment => 0 ,totalWaitingTime => 40
process Id => 2 ,BurstTime => 4 ,arrivalMoment => 0 ,totalWaitingTime => 39
process Id => 3 ,BurstTime => 6 ,arrivalMoment => 1 ,totalWaitingTime => 2
process Id => 4 ,BurstTime => 6 ,arrivalMoment => 1 ,totalWaitingTime => 45
process Id => 5 ,BurstTime => 2 ,arrivalMoment => 2 ,totalWaitingTime => 21
process Id => 6 ,BurstTime => 9 ,arrivalMoment => 2 ,totalWaitingTime => 27
process Id => 7 ,BurstTime => 6 ,arrivalMoment => 3 ,totalWaitingTime => 32
process Id => 8 ,BurstTime => 4 ,arrivalMoment => 3 ,totalWaitingTime => 10
process Id => 9 ,BurstTime => 2 ,arrivalMoment => 4 ,totalWaitingTime => 15
process Id => 10 ,BurstTime => 5 ,arrivalMoment => 4 ,totalWaitingTime => 36
****
Complete execution time => 52
Average process Waiting time => 26

Running preemptive simulation #4
Simulation results:
---
process Id => 1 ,BurstTime => 3 ,arrivalMoment => 0 ,totalWaitingTime => 8
process Id => 2 ,BurstTime => 4 ,arrivalMoment => 0 ,totalWaitingTime => 21
process Id => 3 ,BurstTime => 3 ,arrivalMoment => 1 ,totalWaitingTime => 23
process Id => 4 ,BurstTime => 4 ,arrivalMoment => 1 ,totalWaitingTime => 2
process Id => 5 ,BurstTime => 6 ,arrivalMoment => 2 ,totalWaitingTime => 23
process Id => 6 ,BurstTime => 8 ,arrivalMoment => 2 ,totalWaitingTime => 32
process Id => 7 ,BurstTime => 6 ,arrivalMoment => 3 ,totalWaitingTime => 45
process Id => 8 ,BurstTime => 10 ,arrivalMoment => 3 ,totalWaitingTime => 43
process Id => 9 ,BurstTime => 3 ,arrivalMoment => 4 ,totalWaitingTime => 19
process Id => 10 ,BurstTime => 9 ,arrivalMoment => 4 ,totalWaitingTime => 25
****
Complete execution time => 56
Average process Waiting time => 24

2.3.1 Do the simulation results for the non-preemptive and preemptive versions of the scheduling algorithm differ with any observable patterns?

Based on the above tables, the average results of both non-preemptive and preemptive versions differ from each other. However, a noticeable fact about the preemptive algorithm is that it is more well-organized compared to the non-preemptive due to the reason that the process of choosing process happens randomly by the random scheduler. Also, there is no special pattern about the preemptive waiting time algorithm due to the reason that the processes are invoked not in a long time.

2.3.2 Would the observable behavior for non-preemptive vs preemptive versions be different, if the RNG seed was different? What if the number of simulations was increased to, e.g., 10000?

When it comes to seed alternation, the result would be noticeable. What I mean by this is that at the beginning I have set the RNG value to my birthday. The results are visible in the above tables. However, after changing the seed value different numbers (random) have been observed. Additionally, the usage of the seed is for obtaining the same number for each run.

2.3.3 What are the advantages and disadvantages of such a random scheduling algorithm compared to the First Come First Served (FCFS) algorithm?

Pros:

- The first come first served, it is simple and easy to understand. Therefore, it will run by turn in the run queue.
- It can execute multiple processes at a time.
- Since the process is organized randomly, the CPU will gain the priority of assigning itself to the shortest burst time process. Regarding the FCFS, in case of appearance of large burst time process, the rest of the briefer jobs will have to stand by.

Cons:

- The process with less execution time suffers i.e., waiting time is often quite long.
- It is particularly troublesome for time-sharing systems, where it is important that each user get a share of the CPU at regular intervals.
- Processes will be chosen randomly in the random scheduling algorithm.

Reference

- https://www.usenix.org/sites/default/files/conference/protected-files/atc18_slides_bouron.pdf
- https://www.usenix.org/legacy/event/bsdcon03/tech/full_papers/roberson/roberson.pdf
- https://queue.acm.org/detail.cfm?id=1035622#:~:text=The%20ULE%20scheduler%20creates%20a,affinity%20in%20an%20SMP%20system
- https://man7.org/linux/man-pages/man7/cgroups.7.html
- https://www.usenix.org/system/files/conference/atc18/atc18-bouron.pdf (The provided PDF).