# CS342 Operating Systems - Spring 2021
# Project #2 – Scheduling, Threads, Synchronization

Assigned: March 12, 2021, Friday.
Due date: March 28, 2021, Sunday, 23.59. Document version: 1.2

*This project will be done individually. You will use the C programming language in Linux OS.*

In this project you will implement a multi-threaded scheduling simulator.

There will be N threads running concurrently and generating cpu bursts (workload). Let us call them as workload threads (**W threads**). There will be one server thread that will be responsible from scheduling and executing the bursts (execution in a single cpu will be simulated). Let us call that thread as scheduler thread (**S thread**). Hence, there will be N+1 threads running concurrently, when we run your program. The main thread will create all these N+1 threads in your program. There will be a **runqueue** (**rq**) structure (a linked list or a tree structure, like a red-black tree), i.e., a ready-queue, that will keep the cpu bursts to be executed by the scheduler thread. W threads will add their cpu bursts to rq. S thread will take and execute cpu bursts from rq.

Each W thread will generate a sequence of cpu bursts one-by-one and add them to rq. The length of each burst is random and exponentially distributed with mean **avgB**. The inter-arrival time between two consecutive busts is also random and exponential distributed with mean **avgA**. A W thread will sleep between two burst generations (for that you can use the usleep function). When a burst is generated, its information (description) is added to rq structure as a new node. Information about a burst may include (at least): its thread index, its burst index, its length (in ms), the wall-clock time the burst is generated (you can get this by using the gettimeofday function; ms granularity should be enough).

Execution of a burst by the S thread will be simulated. When the S thread selected a burst to execute, it will simulate the execution of the burst by sleeping for a time duration that is equal to the length of the burst (again you can use usleep). After that much of sleeping, the burst is considered finished (executed), and the S thread will select another burst from the rq structure to run next. The selection is done according to the specified algorithm. If there is no burst to schedule in rq, the S thread will sleep on a **condition variable** until one of the W threads signal on the same condition variable (in this way you will practice **POSIX** condition variables).

Access to the rq structure requires synchronization of the threads (critical section). No two threads should access it simultaneously and cause race condition (in this way you will practice **POSIX mutex** variables).

The program will be called as **schedule** and will have the following parameters.

schedule <N> <Bcount> <minB> <avgB> <minA> <avgA> <ALG>

<N> is the number of W threads. It can be a value between 1 and 10. <Bcount> is the number of bursts that each W thread will generate. A burst time is to be generated as an exponentially distributed random value, with parameter <avgB>. If the generated random value is less than <minB>, it has to be generated again. Similarly, an interarrival time between two consecutive bursts (i.e., time to sleep between bursts in a W thread) is also to be generated as an exponentially distributed random value, with parameter <avgA>. If the generated random value is less than <minA>, it has to be generated again. The <ALG> parameter specifies the scheduling algorithm to use. It can be one of: "FCFS", "SJF", "PRIO", "VRUNTIME". All simulated scheduling algorithms are non-preemptive.

W threads are named, i.e., indexed, starting from 1, up to N. For example, W thread 1, or W thread 2. The first burst of a W thread has index 1, the next burst has index 2, etc.

Your simulator should simulate the following scheduling algorithms.

- FCFS. Bursts served in the order they are added to the runqueue.
- SJF: When scheduling decision is to be made, the burst that has the smallest burst length is selected to run next. But the bursts of a particular thread must be scheduled in the same order they are generated. Hence you look to the earliest burst of each thread and select the shortest one.
- PRIO. A W thread i has priority i. The smaller the number, the higher the priority. The burst in the runqueue belonging to the highest priority thread is selected. Again, we can not reorder the bursts of a particular thread. We have look to the earliest bursts of each thread in the runqueue and select the one that has highest priority.
- VRUNTIME. Each thread is associated with a virtual runtime (vruntime). When a burst of a thread is executed, the vruntime of the thread is advanced by some amount. The amount depends on the priority of the thread. When thread i runs t ms, its virtual runtime is advanced by t(0.7 + 0.3i). For example, virtual runtime of thread 1 is advanced by t ms when it runs t ms in cpu; the virtual runtime of thread 11 is advanced by 4t when it runs t ms in cpu. When scheduling decision is to be made, the burst of the thread that has the smallest virtual runtime is selected for execution. Again we can not reorder the bursts of a particular thread. They need to be served in the order they are added to the runqueue.

It should also be possible to read burst information from files, instead of generating randomly. For that your program should provide such an optionL -f <inprefix>. Then, a W thread i will read its burst information from a file called <inprefix>-i.txt. For example, if <inprefix> is "afile", then the name of the input file for W thread 3 will be "afile-3.txt".

Example invocations of the program can be as follows.

```
schedule 3 75 100 200 1000 1500 FCFS

schedule 5 FCFS -f infile
```

While setting parameter values, we should be careful not to exceed the capacity of the server (cpu). That means, the total burst arrival rate (from N W threads) should not exceed the serving rate (we should have large enough spacing between bursts, on the average).

You can implement the rq structure as a linked list. There is not bound on how large the list can grow.

**Experiments and Report (35 pts).**

Do a lot of experiments, i.e., run your program many times with various parameters values and measure some metrics. For example, you can measure the waiting time of bursts. Then you can find out the average waiting time for a thread. Then you can find the average waiting time caused by a scheduling algorithm for some number of threads. You can compare the scheduling algorithms in terms of waiting time. You can try to draw conclusions. You should repeat the experiments for various values of input parameters (avgB, avgA, N, etc,). It is up to you to design the experiments. For example, you can try to answer the following question: what is the average waiting time for each thread when we use the VRUNTIME algorithm?

**Submission**

Submit a pdf file as your report presenting and discussing your experiments. Your report will include the results, your interpretations and conclusions. Put your report.pdf file and all other files (schedule.c, Makefile, README.txt file) into a directory named with your ID. Then tar and gzip the directory. For example a student with ID 21404312 will create a directory named "21404312" and will put the files there. Then he/she will tar the directory (package the directory) as follows:

```
tar cvf 21404312.tar 21404312
```

Then he/she will gzip the tar file as follows:

```
gzip 21404312.tar
```

In this way he/she will obtain a file called 21404312.tar.gz. Then he/she will upload this file into Moodle.

**Tips and Clarification**

- Starting early is highly recommended.
- Work incrementally.
- The minB value can not be smaller than 100 ms.
- The minA value can not be smaller than 100 ms.
- Normally a thread will not generate a new burst before the current burst is executed. But here, in this project, a thread can. That means, it is possible that a thread i might have generated several bursts (with some time spacing between them), before the first of these bursts had a chance to execute (because there may be other bursts from other threads waiting in the runqueue to execute as well). But this is ok for this project. The goal is to practice and learn by doing.
- POSIX Pthreads API will be used for thread creation, mutex and condition variables.

- Each W thread will start with a random waiting (interarrival-time) initially. Then first burst is generated. That means at time 0, each thread will not generate a burst immediately. First, each thread will sleep for a while. The duration is determined with the same method that you determine the interarrival time between two consecutive bursts.
- Use a separate seed value for each thread to generate random numbers.