EOPSY Lab3 raport – Filip Wasilewski

Situation:

We have created a configuration file in which we set the parameters for the program to run. Configuration file structure looks like this:

```
---------------------------------------------------------------------
// # of Process
numprocess 3

// mean deivation
meandev 2000

// standard deviation
standdev 0

// process    # I/O blocking
process 500
process 500
process 500

// duration of the simulation in milliseconds
runtime 10000
---------------------------------------------------------------------
```

Where:

- Numprocess – is the number of processes to be run
- Meandev – average duration of process
- Standdev - The number of standard deviations from the average length a process should execute before terminating.
- process - specifies interval after which the process blocks I/O for some time. There should be a separate process directive for each one specified by the numprocess.
- runtime – overall runtime of simulation

The values to change are:

- # of proces – we test program for values 3, 5 and 10
- process # I/O blocking has to have the same numer of „proces 500” as the numer in first point

After every run of program, we receive two files with results:

- Summary-Processes.txt
- Summary-Results.txt

Result Analysis

By getting ther results, we can observe the way scheduler works.

As we can see in the Summary-Results.txt file, above the results, we have such info:

_____
Scheduling Type: Batch (Nonpreemptive)
Scheduling Name: First-Come First-Served
Simulation Run Time: 10000
Mean: 2000
Standard Deviation: 0
_____

This data allows us to guess how the algorithm will behave with the processes.

1. **Scheduling type: Batch (Nonpreemptive)** – means that once the scheduler assigns CPU to a proces, it will occupy CPU time until it gets terminated or reaches a waiting state. In our case, the I/O Blocking is the action that breaks CPU assignment for this process, that lasts for 500ms.

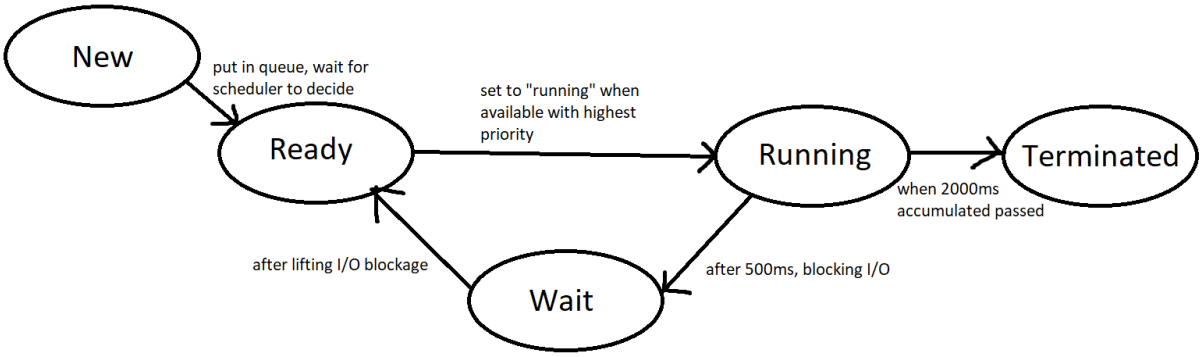
2. **Scheduling Name: First-Come First-Served** – this means that for the processes the program will start with the very first process, and until it is complete it will worry about the first available proces that was scheduled.

3. **Other data** - is the one specified in our config file. To briefly explain, the whole simulation will occupy about 10000ms, with every process averaging the time of 2000ms. Standard deviation is 0, that means that every process will not have randomized length of operation.

So, this means, that we can predict how the scheduler and CPU will handle multiple processes:

1. The scheduler waits for a first process to be ordered, immediately runs it
2. If there are more processes ordered, they wait in queue to be handled
3. When the process 0 gives I/O block for 500ms, the processor moves to first next process,
4. When that gives I/O block, after another 500ms, the scheduler sees, that the first process, which has the biggest priority, is again available, so it goes back to it
5. That means that processor handles processes in pairs, in which the earlier one has more priority, but exchanges CPU time with the second one due to I/O block taking 500ms every time
6. When the first pair is finished, processor moves to the next processes, the pair handling continues for next two processes
7. If at the end we have one process left, it happens on its own as last

8. All this happens until the runtime is finished, and then all processes are terminated, the statistics of their work are passed to Summary-Results.txt



This graph shows the process' life, from creation, until termination.

We will show the example of this process handling with scheduler operation for 3 processes:

Scheduler operations for 3 processes

We have three processes ordered, config file looks like this:

```
---------------------------------------------------------------------
// # of Process
numprocess 3

// mean deivation
meandev 2000

// standard deviation
standdev 0

// process    # I/O blocking
process 500
process 500
process 500

// duration of the simulation in milliseconds
runtime 10000
---------------------------------------------------------------------
```

And we know the processes will have id = 0, 1, 2

And so, let's check the processes log (a snippet of it) for first iteration:

```
Process: 0 registered... (2000 500 0 0)
Process: 0 I/O blocked... (2000 500 500 500)
Process: 1 registered... (2000 500 0 0)
Process: 1 I/O blocked... (2000 500 500 500)
```

Process: 0 registered... (2000 500 500 500)
(…)

Here we can see, the CPU takes the first process (First-Come First-Served) which has id = 0, and focuses only on that. After the process has blocked for input or output, the CPU takes another process with highest priority, id = 1.

As soon as this one blocks, the process 0 is ready to continue, and it has higher priority in the queue than process 2 (because it was ordered earlier) so the CPU returns to process 0 (registered). This cycle repeats until accumulated time (third and fourth value in parenteses) reaches 2000 for every process.

And so, this will repeat until the processes complete. We can see when the process 2 comes into play, when 0 and 1 finishit at the bottom:

(...)
Process: 0 registered... (2000 500 1500 1500)
Process: 0 completed... (2000 500 2000 2000)
Process: 1 registered... (2000 500 1500 1500)
Process: 1 completed... (2000 500 2000 2000)
Process: 2 registered... (2000 500 0 0)

After that moment, process 2 is the only one in the queue, so it will work alone. We can see when the process blocks I/O and later is immediately registered, since there are no free processes to be run:

(...)
Process: 2 registered... (2000 500 0 0)
Process: 2 I/O blocked... (2000 500 500 500)
Process: 2 registered... (2000 500 500 500)
Process: 2 I/O blocked... (2000 500 1000 1000)
Process: 2 registered... (2000 500 1000 1000)
Process: 2 I/O blocked... (2000 500 1500 1500)
Process: 2 registered... (2000 500 1500 1500)
Process: 2 completed... (2000 500 2000 2000)

When the processes accumulates 2000ms of operation, it completes. If we look into Summary-Results.txt file, we can see the statistics:

Process #	CPU Time	IO Blocking	CPU Completed	CPU Blocked
0	2000 (ms)	500 (ms)	2000 (ms)	3 times
1	2000 (ms)	500 (ms)	2000 (ms)	3 times
2	2000 (ms)	500 (ms)	2000 (ms)	3 times

It makes sense that we have 3 CPU blocks, since for every process the block happens every 500ms, so at time 500ms, 1000ms, 1500ms, because at 2000 the process ends.

Scheduling for 5 processes

What is interesting here, is the order of processes in Summary-Processes.txt. At first, the program returns the same values as for 2 processes, until two first processes complete. There we can see two new processes take the same cycle – one blocked, other working:

(...)
Process: 0 registered... (2000 500 1500 1500)
Process: 0 completed... (2000 500 2000 2000)
Process: 1 registered... (2000 500 1500 1500)
Process: 1 completed... (2000 500 2000 2000)
Process: 2 registered... (2000 500 0 0)
Process: 2 I/O blocked... (2000 500 500 500)
Process: 3 registered... (2000 500 0 0)
Process: 3 I/O blocked... (2000 500 500 500)
(...)

Because process 0 is not complete at first, the scheduler switches between available processes 1 and 0, since, as we said earlier, scheduling is „First-Come First-Served". So no process 2 and later will come into play until process 0 is finished.

The pair of processes 2 and 3 loops the same as 1 and 0. When they complete, process 4 comes into play, and since it is only one left, there are only process 4 logs:

(...)
Process: 2 registered... (2000 500 1500 1500)
Process: 2 completed... (2000 500 2000 2000)
Process: 3 registered... (2000 500 1500 1500)
Process: 3 completed... (2000 500 2000 2000)
Process: 4 registered... (2000 500 0 0)
Process: 4 I/O blocked... (2000 500 500 500)
Process: 4 registered... (2000 500 500 500)
Process: 4 I/O blocked... (2000 500 1000 1000)
Process: 4 registered... (2000 500 1000 1000)
Process: 4 I/O blocked... (2000 500 1500 1500)
Process: 4 registered... (2000 500 1500 1500)

Looking at statistics from Summary Results as expected:

Process #	CPU Time	IO Blocking	CPU Completed	CPU Blocked
0	2000 (ms)	500 (ms)	2000 (ms)	3 times
1	2000 (ms)	500 (ms)	2000 (ms)	3 times
2	2000 (ms)	500 (ms)	2000 (ms)	3 times
3	2000 (ms)	500 (ms)	2000 (ms)	3 times
4	2000 (ms)	500 (ms)	2000 (ms)	3 times

Every process finished properly. 3 CPU Blocks show that.

...but what if the processes go on and on until the 10000ms of full simulation run out?

Scheduling for 10 processes

Here is the factor, we have not taken into account yet: runtime of all simulation, which is 10000ms.

For 10 processes, the beginning is as usual, process 0 and 1 looping until they reach completion. The same for 2 and 3. Because we have 10, processes 4 and 5 come in pair into play:

```
(...)
Process: 3 completed... (2000 500 2000 2000)
Process: 4 registered... (2000 500 0 0)
Process: 4 I/O blocked... (2000 500 500 500)
Process: 5 registered... (2000 500 0 0)
Process: 5 I/O blocked... (2000 500 500 500)
(...)
```

But then those are next 3 LAST lines of log file:

```
(...)
Process: 4 registered... (2000 500 500 500)
Process: 4 I/O blocked... (2000 500 1000 1000)
Process: 5 registered... (2000 500 500 500)
```

Why has this stopped? As we look iinto Summary-Results.txt we get even stranger results:

Process #	CPU Time	IO Blocking	CPU Completed	CPU Blocked
0	2000 (ms)	500 (ms)	2000 (ms)	3 times
1	2000 (ms)	500 (ms)	2000 (ms)	3 times
2	2000 (ms)	500 (ms)	2000 (ms)	3 times
3	2000 (ms)	500 (ms)	2000 (ms)	3 times
4	2000 (ms)	500 (ms)	1000 (ms)	2 times
5	2000 (ms)	500 (ms)	1000 (ms)	1 times
6	2000 (ms)	500 (ms)	0 (ms)	0 times
7	2000 (ms)	500 (ms)	0 (ms)	0 times
8	2000 (ms)	500 (ms)	0 (ms)	0 times
9	2000 (ms)	500 (ms)	0 (ms)	0 times

That is because we have reached the end of simulation time. If we count all „CPU Completed”, we will get 10000ms. So unfortunely, the simulation has finished before we could finish all processes.