Simulation Techniques – CPU SCHEDULER

Maciej Anioł 116320

1. **Task number, simulation method.**

   Task number 4, simulation method M4 = Process interaction
   CPU scheduler simulation done in Process Interaction method with First Come First Serve CPU scheduling algorithm and Shortest Job First I/O scheduling algorithm. Number of processors is equal to 4 and number of I/O devices is equal to 5.

2. **Description of the main task**

   The process scheduling (also called as **CPU scheduler**) is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy. A scheduling allows one process to use the CPU while another is waiting for I/O, thereby making the system more efficient, fast and fair. In a multitasking computer system, processes may occupy a variety of states (Figure 1). When a **new** process is created it is automatically admitted the **ready** state, waiting for the execution on a CPU. Processes that are ready for the CPU are kept in a **ready queue**. A process moves into the **running** state when it is chosen for execution. The process's instructions are executed by one of the CPUs of the system. A process transitions to a **waiting** state when a call to an I/O device occurs. The processes which are blocked due to unavailability of an I/O device are kept in a **device queue**. When a required I/O device becomes idle one of the processes from its **device queue** is selected and assigned to it. After completion of I/O, a process switches from the **waiting** state to the **ready** state and is moved to the **ready queue**. A process may be **terminated** only from the **running** state after completing its execution. Terminated processes are removed from the OS.

   Develop a C++ project that simulates the process scheduling described above, in accordance with the following parameters:

   - **Process Generation Time (PGT)** [ms] – time before generation of a new processes (random variable with exponential distribution and intensity **L**) (round to natural number)
   - **CPU Execution Time (CET)** [ms] – process execution time in CPU. Random variable with uniform distribution between <1, 50> [ms] (natural number)
   - **I/O Call Time (IOT)** [ms] – time between getting an access to the CPU and an I/O call. Random variable with uniform distribution between <0, CET-1> [ms] (natural number). In case of 0, there is no I/O call.

- **I/O Device (IOD)** – indicates which I/O device is requested by the running process. Random variable with uniform distribution between <0, NIO-1>, where NIO is the number of I/O devices in the OS.
- **I/O Time (IOT)** [ms] – I/O occupation time. Random variable with uniform distribution between <1, 10> [ms] (natural number).

Determine the value of the parameter **L** that ensures the average waiting time in the ready queue not higher than 50 ms. Then, run at least ten simulations and determine:

- CPU utilization (for every CPU) [%]
- Throughput – number of processes completed (terminated) per unit time
- Turnaround time – time required for a particular process to complete, from its generation until termination [ms]

3. Simulation model scheme

Figure 1 – Simulation model scheme.

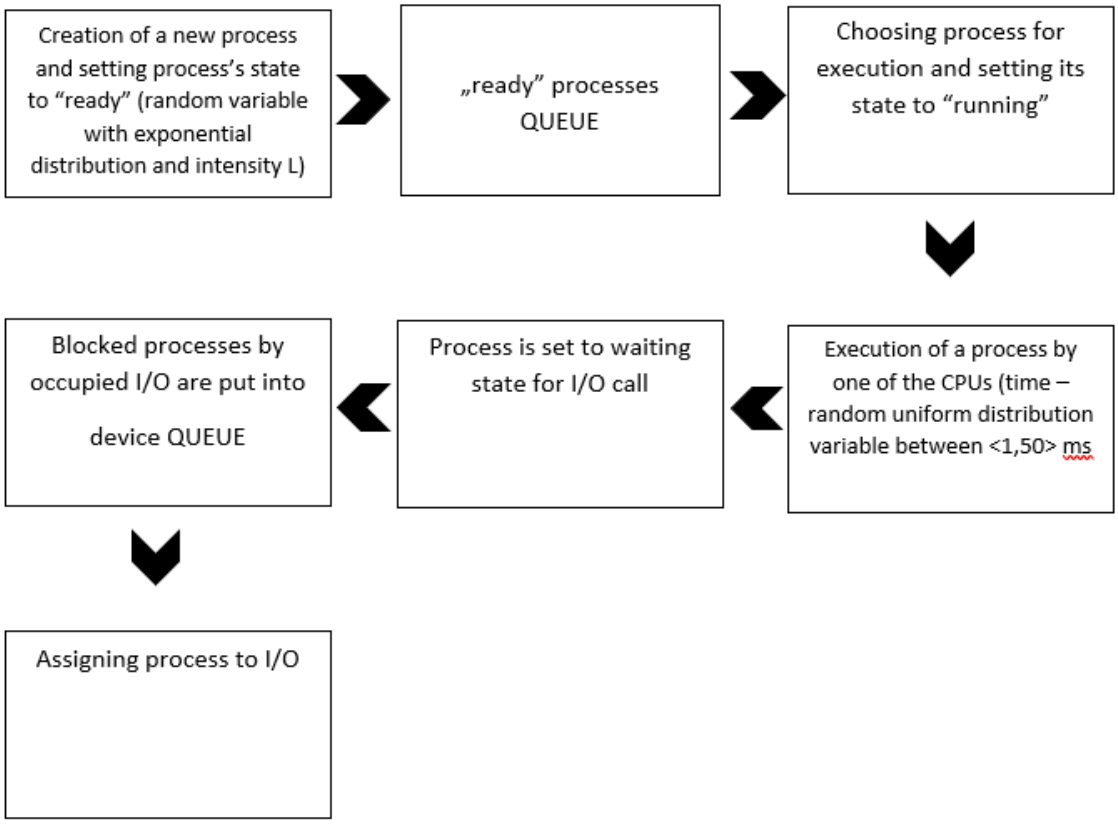
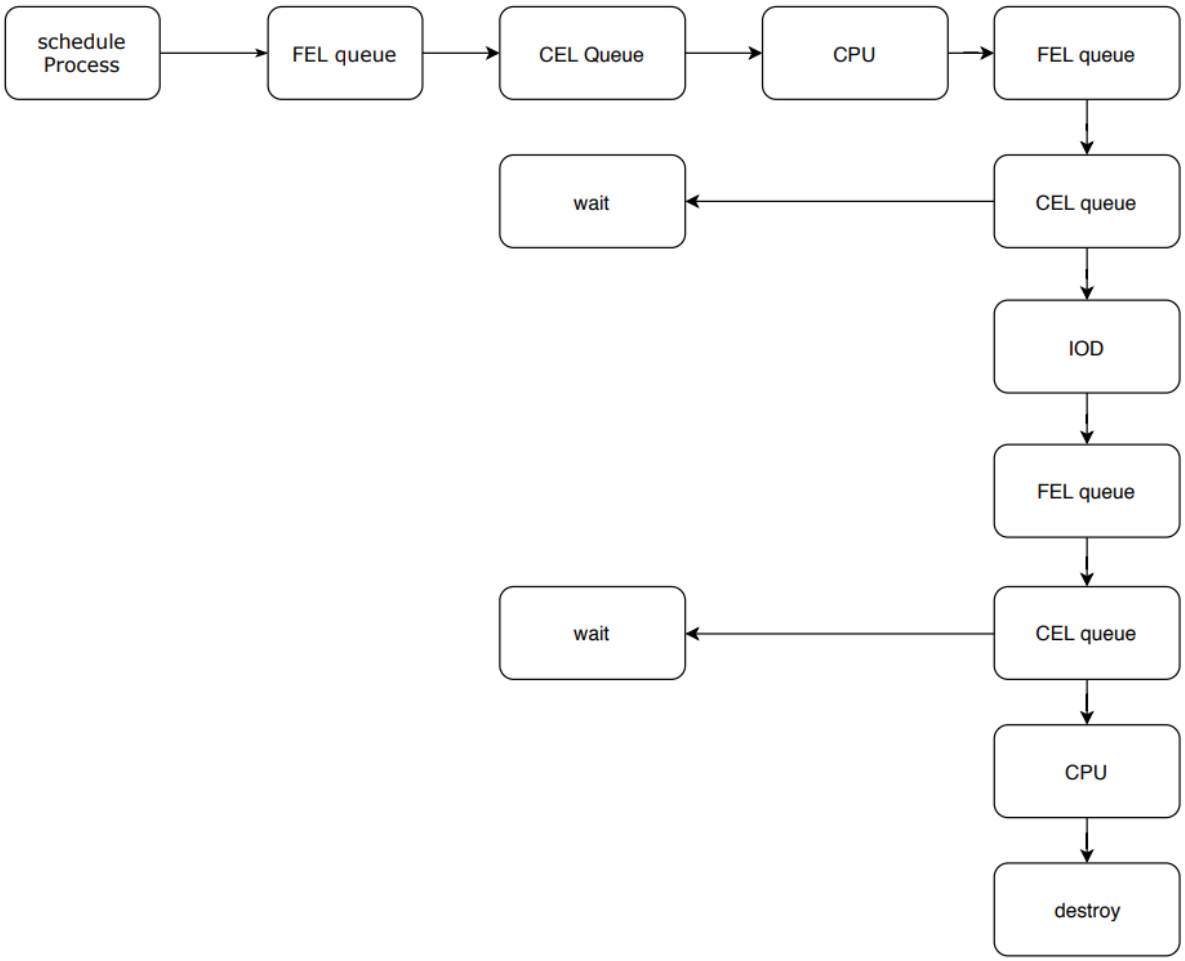


Figure 2 – Life cycle of one process.



Firstly, process needs to be scheduled, when its scheduled it is passed to **Future Event List (FEL)** queue. From there if there is place in **Conditional Event List (CEL)** queue it is sent there as a process waiting for **CPU**. Before passing process to **I/O Device (IOD)** it is again going through **FEL** and **CEL** queue. In **CEL** queue process is waiting for free **I/O Device**. After managing process by **I/O Device** it is passed to **FEL** -> **CEL** queue where it waits for free **CPU** and lastly it is destroyed/killed.

4. Description of objects and their attributes

Object	Class name	Description	Attributes
cpuCS	CPUCoreSupervisor	Class used for managing CPU cores (CPUs cointainer)	- CPUs ID
No object name because it is stored in a vector: `vector<CPUCore> cpus;`	CPUCore	Class that represents a single CPU core	- current Proccess pointer (*cPtr) - loadTime (int) - busySince (int) - CPUbusy state (bool)
No object name because it is stored in vector: `vector<IODevice> iod`	IODevice	Class represents single I/O device	- pointer to process that is currently served (cPtr) - I/O Device busy state (bool)
ioDS	IODeviceSupervisor	Class for managing I/O devices	- ID of I/O Device
No object name; only pointers to objects (newPtr, pPtr)	simprocess	Class represents single process	- creation time (int) - CPU execution time (int) - state of a process + next state (STATE) - I/O Device delay (int) - I/O Device occupation state (int) - I/O Device ID (int)

5. List of events/processes

Time processes:

- Process generation – generate new process and put it in FEL queue
- Process execution – execute process by putting it into CPU or I/O device

Conditional events:

- Check if CPU is free; if yes move process from CELcpu queue to CPU
- Check if I/O Device is free; if yes move process from CELio queue to I/O
- Termination of a process – remove process after it is executed
- If current process was just created, schedule a new one

6. Generators

Pseudo-random number generator is used to generate numbers in given probability distribution. In simulation multiplicative generator was used to obtain Uniform and Exponential random number generators.

- Description of implemented generators:

  Uniform Distribution Random Generator

  Values: M = 2147483647.0; A = 16807; Q = 127773; R = 2836;
  kernel_ = current seed

  Function:

```
double UniformGenerator::Rand(){
      int h = kernel_/Q;
      kernel_ = A*(kernel_-Q*h)-R*h;
      if (kernel_ < 0)
            kernel_ = kernel_ +
      static_cast<int>(M);
      return kernel_/M;
      }
```

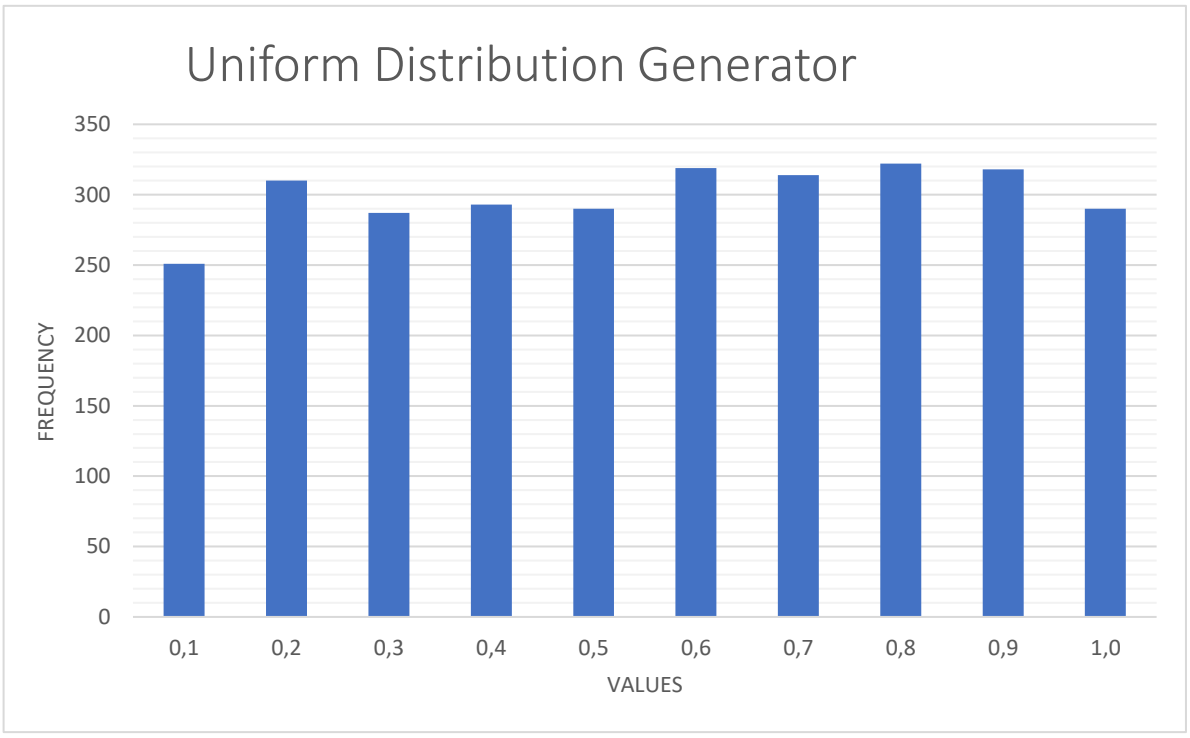
  Exponential Distribution Random Generator

  Exponential distribution was achieved by implementing formula: $F^{-1} = -\lambda^{-1}\ln(k)$.
  Where: k – random number with uniform distribution in range (0,1)
  λ – exponential distribution intensity

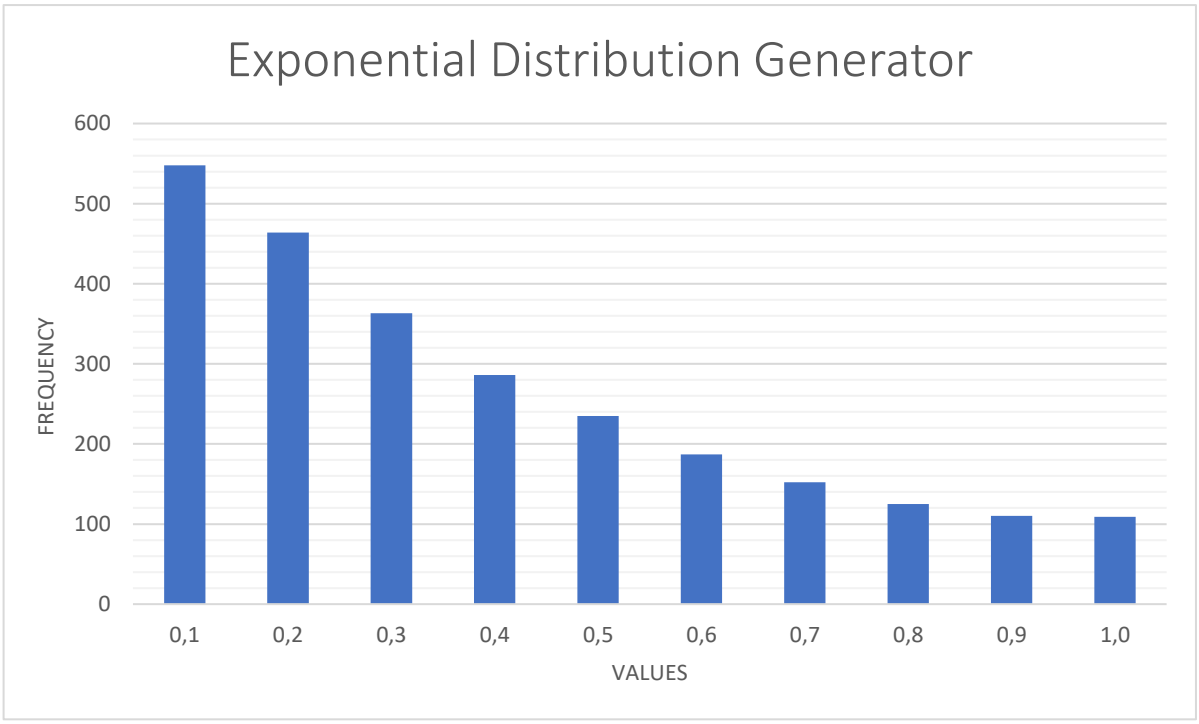
  Function:

```
int ExpGenerator::Rand(){
      double k = uniform_->Rand();
return static_cast<int>( ceil( -
(1.0/lambda_)*log(k) ));
}
```

- Histograms
    - Uniform Distribution RNG histogram for 3000 samples:



    - Exponential Distribution RNG histogram for 3000 samples:



7. Program input parameters

a. *Global:*

Number of CPUs CPUCount = 4
Number of I/O devices IODCount = 5
Lambda = 0.65

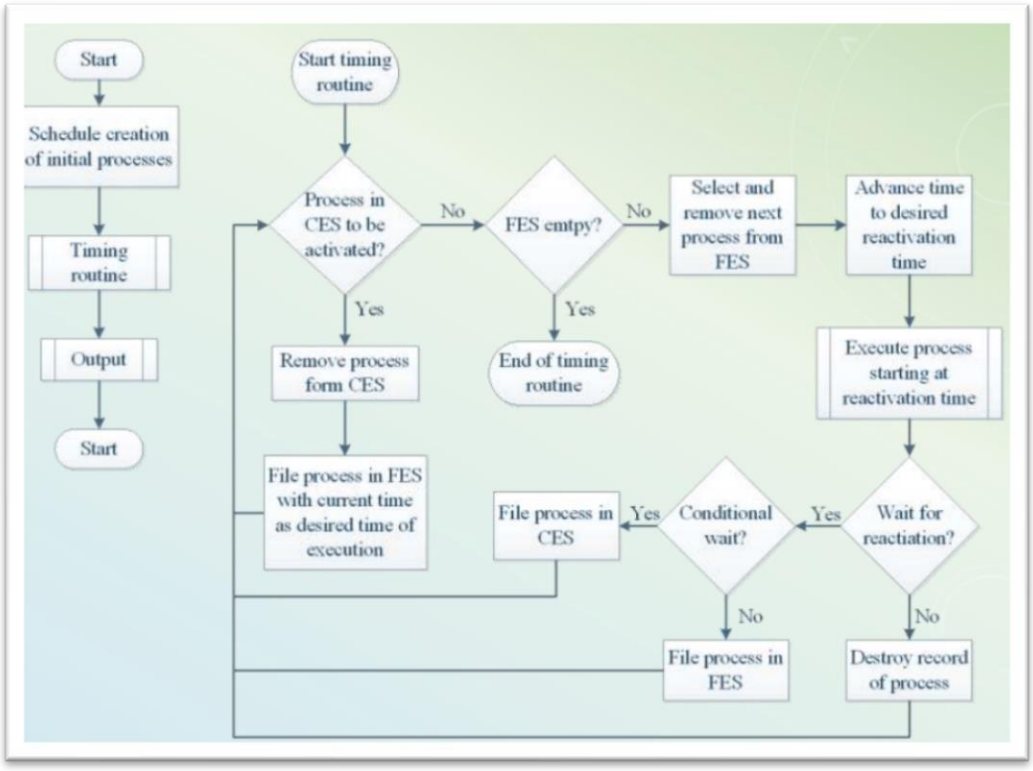
b. main.cpp main function:

seed = 1546833492
simulationTime = 0
processCount = 0
processCountTotal = 1

8. Code testing methods.

I did a step by step mode for few iterations and checked if every function works properly by investigating every variable change. I checked if every queue works properly, if generators have proper distribution histogram.

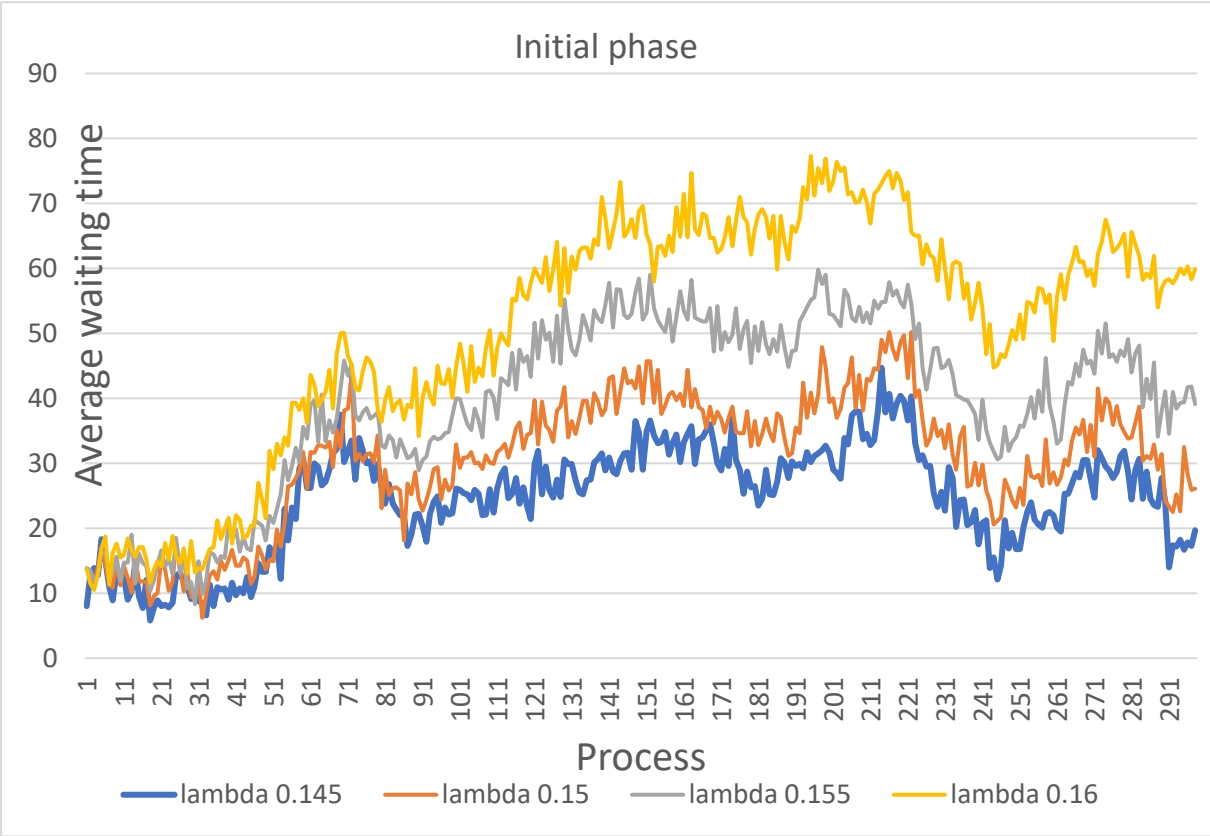
I tried to predict output of any action in each step by following process interaction algorithm block diagram from lecture materials:



9. Simulation results:

a. Initial phase

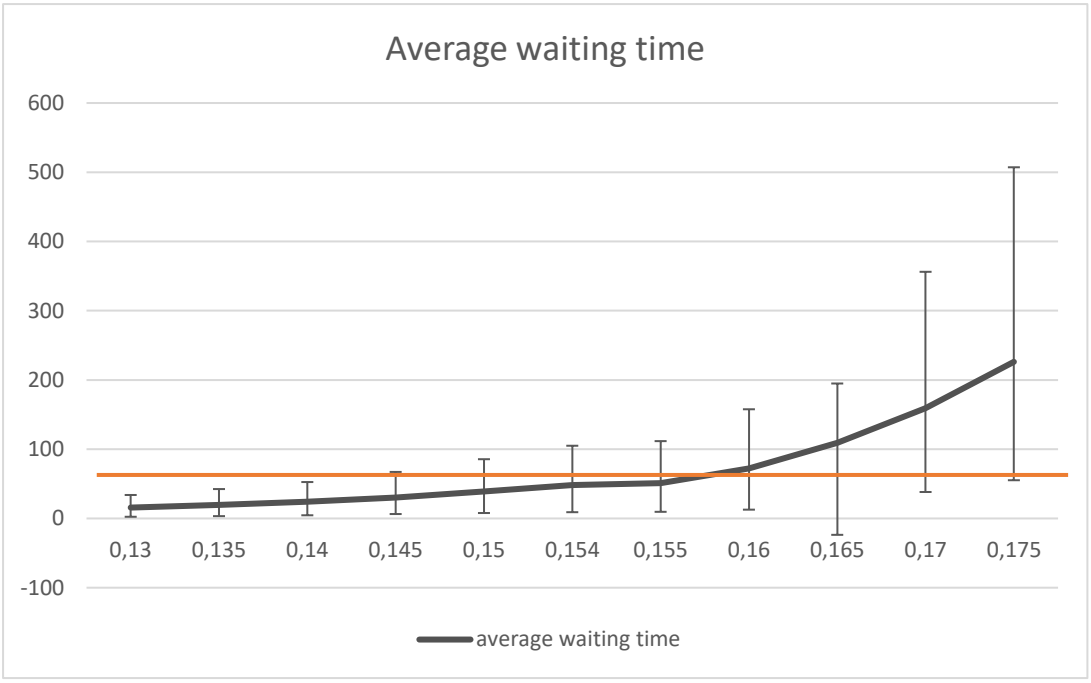
Initial phase determination was done for simulation time equals 5000 for four different lambdas: 0.145, 0.15, 0.155, 0.16. At process number 51 with simulation time at this point 467 graph quickly ascends so I set 500ms initial phase delay. Chart can be seen below:



b. Determination of lambda parameter was done for simulation time 5000ms and ten lambdas:

lambda	0.13	0.135	0.14	0.145	0.15	0.154	0.155	0.16	0.165	0.17	0.175
average w	15.71271	19.56389	23.98303	30.41033	38.86927	48.04118	51.08635	72.50221	109.3276	159.2259	226.1967

Lambda 0.154 is chosen because it's the closes to 50ms mark.



Simulation number	Average waiting time in the ready queue	CPU #1 utilization	CPU #2 utilization	CPU #3 utilization	CPU #4 utilization	Through-put	Average turnaround time
1	34.9051	92.755556,	90.911111,	87.400000,	83.377778,	0.135778	67.2848,
2	44.6059	96.688889,	95.311111,	92.511111,	89.933333,	0.143778	76.9938,
3	43.5158	96.000000,	95.888889,	92.800000,	90.533333,	0.140444	76.1915,
4	38.2837	95.933333,	94.200000,	90.800000,	88.688889,	0.141778	70.279,
5	78.4293	95.288889,	93.044444,	90.133333,	88.511111,	0.133556	112.073,
6	39.5827	95.733333,	92.133333,	91.200000,	89.466667,	0.135778	72.6579,
7	74.375	97.200000,	96.755556,	96.288889,	94.688889,	0.149333	106.054,
8	35.9462	92.177778,	89.288889,	89.133333,	84.488889,	0.136222	68.3899,
9	45.3981	93.444444,	92.511111,	90.866667,	89.533333,	0.141778	77.4295,
10	45.37	94.20000	93.71111	89.91111	87.4000	0.139333	77.9649,
Average	48.04118	94.94222,	93.37556	91.10444	88.66222	0.139778	80.53183

Confidence interval	39 to 57.1	93.9 to 95.9	92 to 94.7	89.7 to 92.5	86.8 to 90.5	0.137 to 0.143	71.4 to 89.7

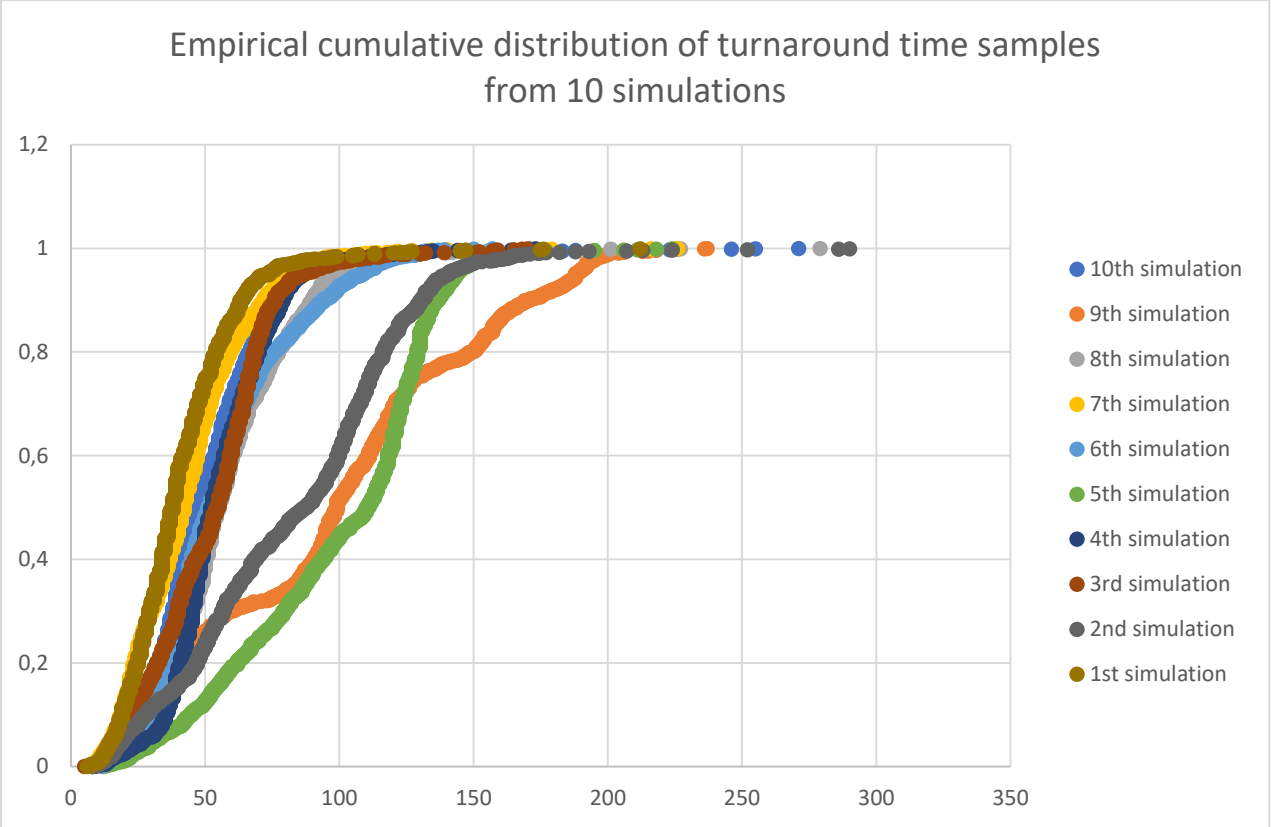
To calculate confidence interval following steps were made:

- Subtraction of value in each simulation with average value from 10 simulations = a_1-$a_{10.}$
- a_1-a_{10} to the power of 2 = b_1-b_{10}.
- Sum of all b_1-b_{10} from previous point divided by 9 = *c*
- Square root of *c* = *d.*
- Finally, calculation confidence interval by subtraction (2,26*(*d*/3)) from average value from 10 simulations (a_n).

Confidence interval calculated for 10 different lambdas according to average waiting time in ready queue are presented below:

lambda	0.13	0.135	0.14	0.145	0.15	0.154	0.155	0.16	0.165	0.17	0.175
average w	15.71271	19.56389	23.98303	30.41033	38.86927	48.04118	51.08635	72.50221	109.3276	159.2259	226.1967
	13.8835	17.2661	20.1313	23.1828	27.9697	34.9051	36.9283	46.8203	60.4845	76.4524	97.2412
	16.0157	18.8291	22.3488	28.0725	34.8404	44.6059	49.6185	78.6182	135.154	229.464	329.465
	14.1852	16.4324	19.5321	24.6341	31.8964	43.5158	47.3501	71.6573	103.134	144.811	226.515
	11.2044	14.4741	18.738	23.6908	30.7107	38.2837	40.259	68.8807	132.659	215.587	300.235
	24.5679	32.9565	41.8655	54.0516	66.9357	78.4293	81.1047	101.921	126.196	151.943	191.724
	13.2756	16.9619	19.7075	24.6271	30.9715	39.5827	42.6499	65.7449	93.3932	127.266	192.131
	21.2285	25.6065	34.0098	46.682	59.5445	74.375	80.5319	115.616	196.163	279.657	395.245
	11.72	14.5323	17.5661	21.5889	30.6801	35.9462	38.0553	49.4205	60.3379	83.1559	118.857
	14.7468	18.4735	21.9261	29.0231	38.8481	45.3981	47.2833	65.0341	95.3739	130.052	168.388
	16.2995	20.1065	24.0051	28.5504	36.2956	45.37	47.0825	61.3091	90.3805	153.871	242.166
upper banc	18.2	22.9	28.6	36.8	46.7	57	60.6	85.2	85.6	197	281
lower banc	13.3	16.2	19.4	24	31	39	41.5	59.8	133	121	171

c. Empirical cumulative distribution function chart of turnaround time samples from ten simulations:



10. Conclusions

- For more accurate results number of tested simulations should be increased. It helps with decreasing confidence intervals.
- First CPU utilization is always bigger than second because system of scheduling processes is set up to occupy first CPU if its free but if it is not it goes to next CPU.
- Presence time in the system is connected with waiting time in queue to CPU, to I/O device, occupation time in CPU and in I/O device.
- Simulation omits first 500ms due to initial phase calculation
- Lambda was set to 0.154 because it is the closest value to 50ms average waiting time in queue.
- When lambda lowers, simulation produces less processes.