**Simulation methods - project**

Tomasz Krawczyk

Task number : 3

Simulation method : M3 – ABC approach

CPU scheduling algorithm : SJF

I/O scheduling algorithm : Random

Number of processors : 4

Number of I/O devices : 5

1. 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.

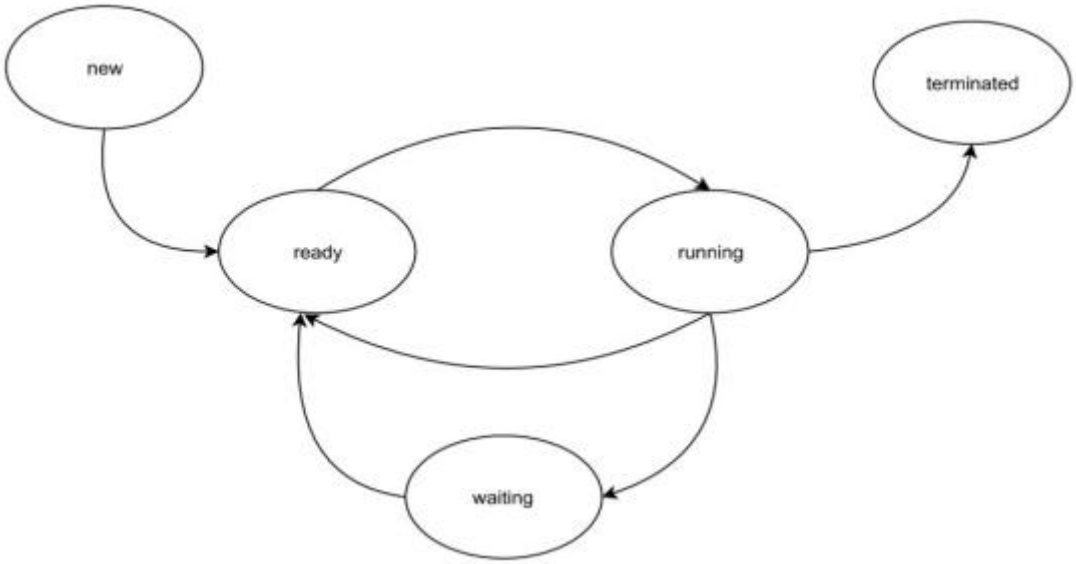


*Figure 1. Process state diagram*

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 [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 [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 , 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

[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]

Draw a figure of an average waiting time in the ready queue in the function of parameter

2. Simulation model scheme

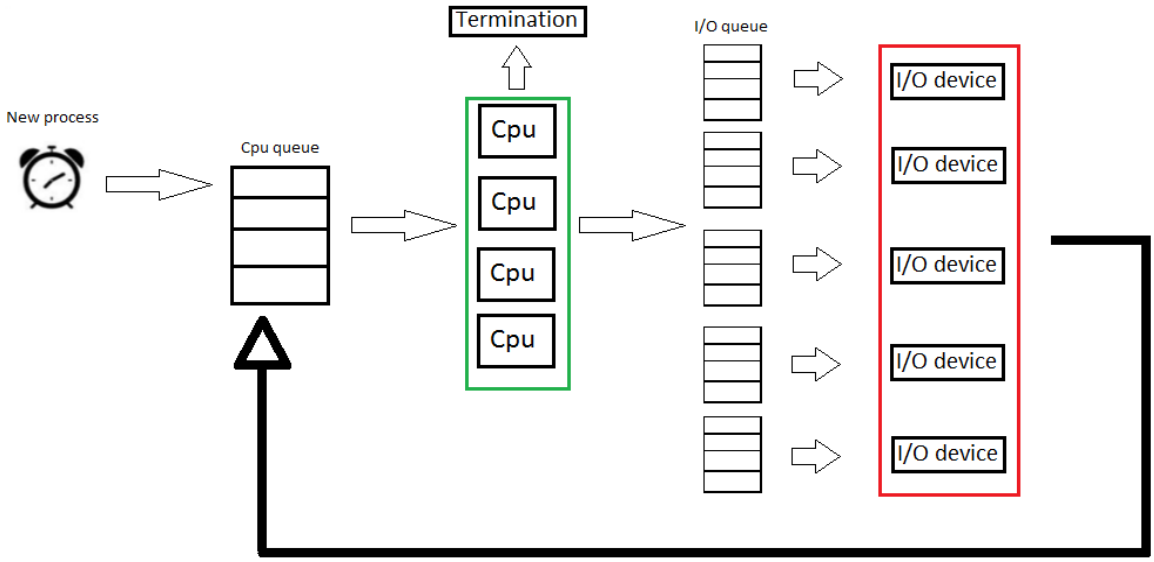


*Figure 2. Simulation model scheme*

3. Description of objects and their attributes

| Object | Class name | Description | Attributes |
|---|---|---|---|
| CPU scheduler | CpuScheduler | Class used to contain queues, CPUs, I/O devices, generators (uniform/exponential),variables used to implement statistic | -Number of processors and number of I/O devices – const int<br>-Processors vector of type vector <CPU*><br>-I/O devices vector of type vector <I/O device*> |

| | | | |
|---|---|---|---|
| | | | -Processes queue of type CPU queue<br>-Processes queue of type I/O queue<br>-Generators of type UniformGenerator/ExpGenerator<br>-Variables used to implement statistic of type int/long int/double |
| Process | Process | Class that represent single process with all its core attributes – also contain attributes used to implement statistic | -ID – unique identifier of a process<br>-CET – random variable of type int<br>-IOCT – random variable of type int<br>-IOD – random variable of type int<br>-IOT – random variable of type int<br>-Waiting time – variable of type int<br>-Turnaround time – variable of type int<br>-Final time – variable of type int |
| CPU | Cpu | Class represent single processor | - pointer to process that is currently executed – type Process*<br>-Busy time – variable of type int<br>-Busy – variable of type bool<br>-End of work time – variable of type int |
| CPU queue (Ready queue) | CpuQueue | Queue to processor - SJF | -processes list – vector (at this moment) <Processes*><br>-Generator – random generator of type UniformGenerator |
| I/O queue (Devices queue) | IoQueue | Queue to I/O device - Random | -processes list - vector (at this moment)<br><Processes*><br>-Generator – random generator of type UniformGenerator |
| I/O device | IoDevice | Class represent single I/O device | -pointer to process that is currently executed – type Process*<br>-Busy time – variable of type int<br>-End of work time – variable of type int |

*Table 1. Description of objects and their attributes*

4. List of event

| Event | Algorithm |
|---|---|
| New process | 1) Creates a new process<br>2) Set CET,IOCT and IOT to the process<br>3) Get new PGT value<br>4) Add process to the queue<br>5) Schedule next process |
| End of work of processor | 1) Check which processor should end work<br>2) Set IOD to the process<br>3) Update CET of the process<br>4) Delete process if it's IOCT = 0<br>5) Otherwise add process to the I/O queue<br>6) Make processor free |
| End of work of I/O device | 1) Check which I/O device should end work<br>2) Update IOCT of process<br>3) Update IOT of process<br>4) Add process to the ready queue<br>5) Make I/O device free |

*Table 2. Time events table*

| Event | Algorithm |
|---|---|
| Start of work of processor | 1) Check if processor is free<br>2) Check if queue is not empty<br>3) Make processor busy<br>4) Assign process to the processor<br>5) Remove process from ready queue<br>6) Schedule end of work event and add it to calendar |
| Start of work of I/O device | 1) Check if I/O device is free<br>2) Check if queue is not empty<br>3) Make I/O device busy<br>4) Assign process to the I/O device<br>5) Remove process from I/O queue<br>6) Schedule end of work event and add it to calendar |

*Table 3. Conditional events table*

5. Generators

Way of generating seeds :

My initial seed is 10, I need 70 seeds for my simulation(70 in total, 7 for each simulation – 10 simulations). I saved every seed after 100000 draws (span between seeds is 100000).

Generators:

I'm using 2 types of generators – exponential generator and uniform generator.

Uniform generator:
M = 2147483647.0;
A = 16807;
Q = 127773;
R = 2836;

```
int h = kernel_/Q;
kernel_ = A*(kernel_-Q*h)-R*h;
if (kernel_ < 0)
        kernel_ = kernel_ +static_cast<int>(M);
```

Variable „kernel_" is the current seed. Program calculated new value for kernel and this value will be returned as the new random number and also will be saved as the new seed.

Histogram of uniform generator ( 1.000.000 results were drawn, range (0 – 10))

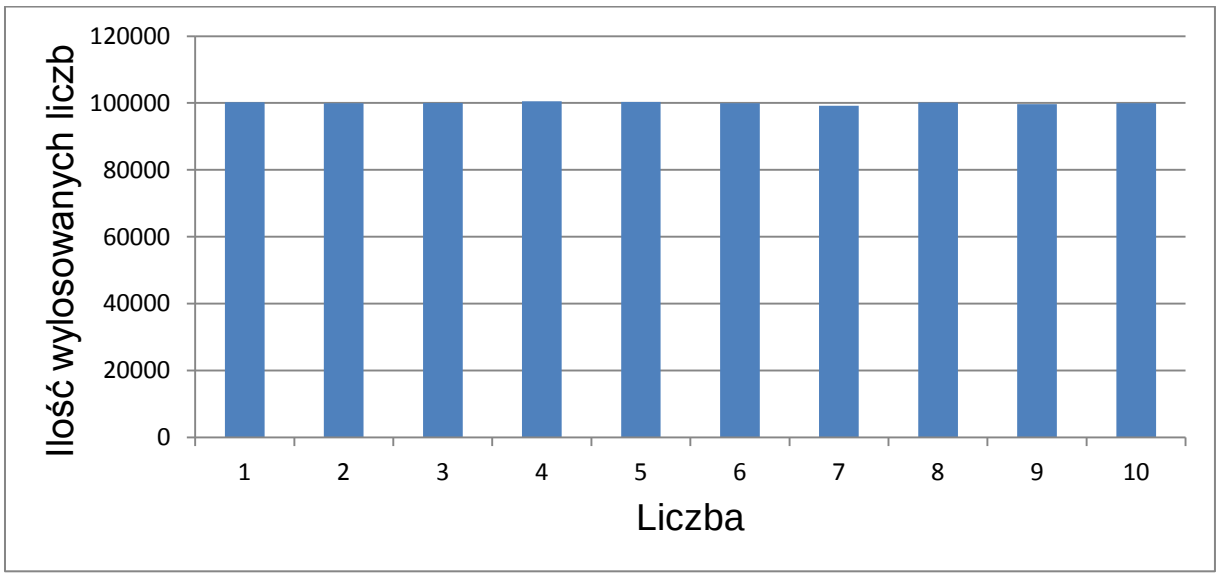


*Figure 3. Uniform generator histogram*

Exponential generator:

```
double k = uniform_->Rand01();
 return -(1.0/lambda_)*log(k);
```

uniform_ – uniform generator
Rand01 – function returns random number in the range 0-1

Histogram of exponential generator(100.000 results were drawn, lambda = 0.132)

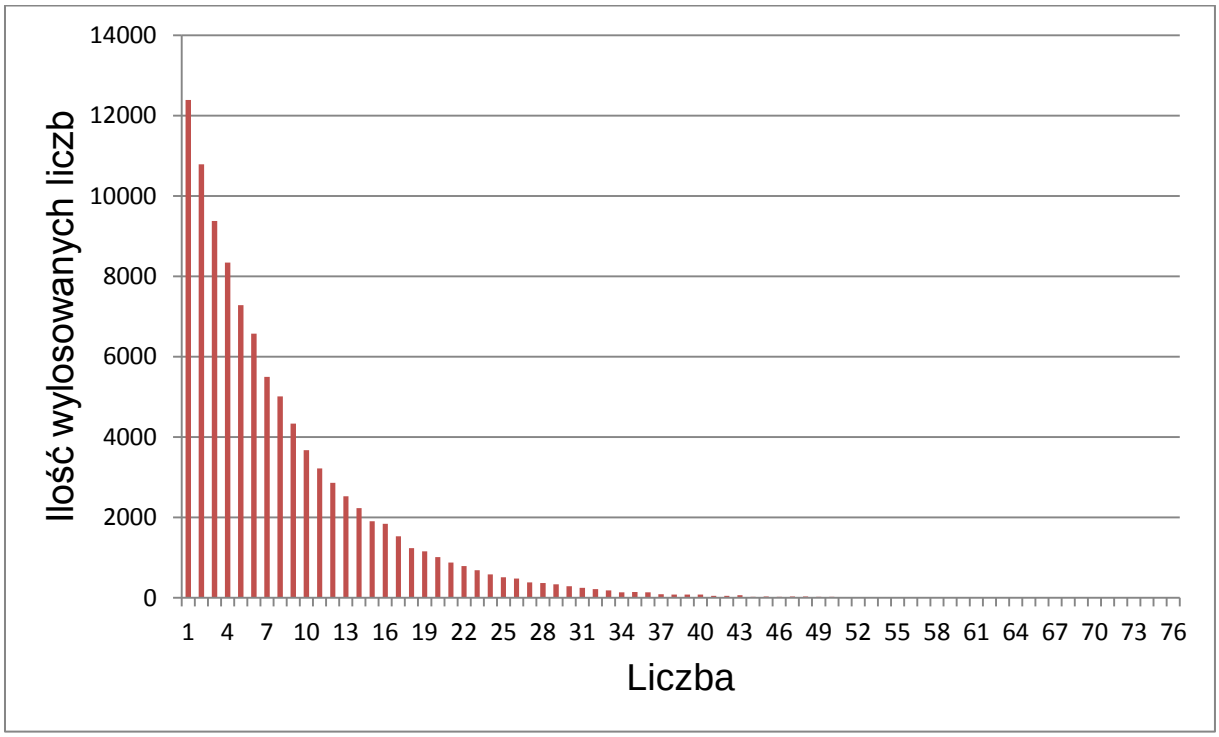


*Figure 4. Exponential generator histogram*

6. Program input parameters

Simulation time – user can specify duration of simulation
Lambda – user can specify lambda parameter which is needed for exponential generator
Mode of work – user can specify in which mode program should work : 0 – continuous mode (only results are shown), 1 – step by step mode (logs analysis)
Set initial phase – user can specify if the initial phase should be omitted in final results

7. Code testing methods

Log analysis in step-mode – logs are showing current state of the process i.e. where the process is, current simulation time, in which processor and I/O device process is currently executed. In other words, we can track the whole "life cycle" of a process – from its birth to termination. This helps in finding unexpected behaviors of a program and leads to eliminate bugs and errors.

8. Simulation results
   a) Initial phase determination

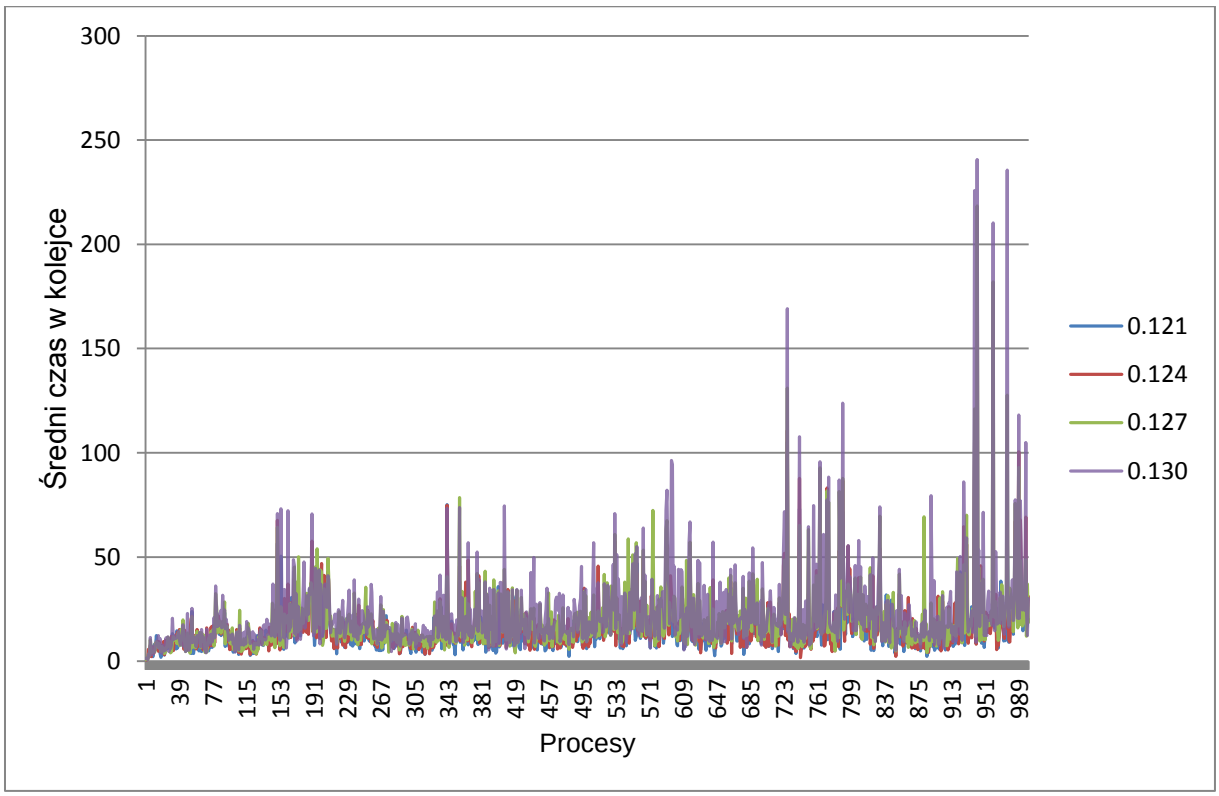


*Figure 5. Initial phase determination*

Initial phase was determined according to following rules :
-run at least ten simulations
-for at least one parameter – waiting time in the ready queue
-for at least four different lambdas

According to the rules presented on project lessons my initial time is 1150 [ms] – that is termination of process number 120. Time parameter was determined on the basis of ten simulations. In the worst case process #120 was terminated in time 1120 [ms], so time 1150 [ms] assures that initial phase of simulation is over and results can be gathered.

b) Determination of exponential generator intensity L (lambda)

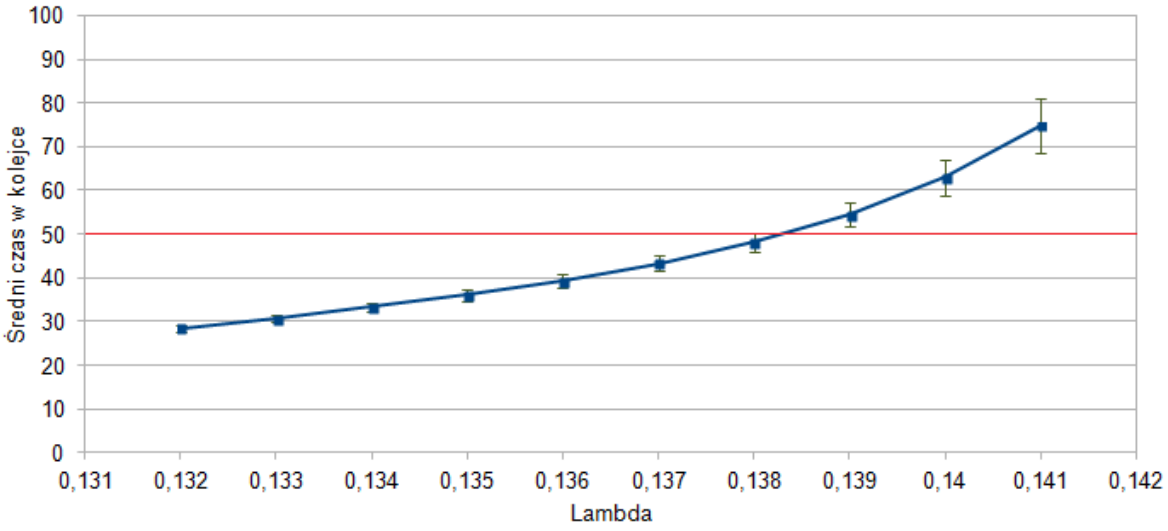


*Figure 6. Lambda determination*

Lambda was determined according to following rules :
-initial phase is omitted
-taking the highest lambda that gives the average waiting time in the ready queue less than 50 ms
-at least five lambdas simulated

I chose lambda equal to 0.137

c) Results for at least ten simulations gathered in the table

| Simulation number | Average waiting time in the ready queue [ms] | CPU #0 utilization [%] | CPU #1 utilization [%] | CPU #2 utilization [%] | CPU #3 utilization [%] | Throughput [process/ms] | Average turnaround time [ms] |
|---|---|---|---|---|---|---|---|
| 0 | 40.0 | 95.8 | 94.4 | 92.4 | 90.1 | 0.146427 | 85.7 |
| 1 | 42.6 | 95.9 | 94.5 | 92.5 | 90.2 | 0.146468 | 88.4 |
| 2 | 42.0 | 96.2 | 94.8 | 93.0 | 90.7 | 0.147217 | 88.0 |
| 3 | 41.1 | 95.9 | 94.5 | 92.6 | 90.1 | 0.146680 | 86.8 |
| 4 | 44.8 | 96.4 | 95.0 | 93.3 | 91.1 | 0.147520 | 90.7 |
| 5 | 43.6 | 96.2 | 94.8 | 93.2 | 90.9 | 0.147224 | 89.4 |
| 6 | 47.4 | 96.1 | 94.8 | 93.0 | 90.6 | 0.146998 | 93.3 |
| 7 | 42.6 | 95.9 | 94.5 | 92.5 | 90.1 | 0.146543 | 88.3 |
| 8 | 44.1 | 96.3 | 95.0 | 93.3 | 91.1 | 0.147066 | 90.0 |
| 9 | 45.5 | 96.4 | 95.2 | 93.5 | 91.4 | 0.147172 | 91.4 |
| Average | 43.4 | 96.1 | 94.7 | 92.9 | 90.6 | 0.146932 | 89.2 |
| Confidence interval +/- | 1.65 | 0.15 | 0.20 | 0.29 | 0.36 | 0.000284 | 1.70 |

*Table 4. Simulation results*

d) Empirical cumulative distribution function chart of turnaround time samples from ten simulations

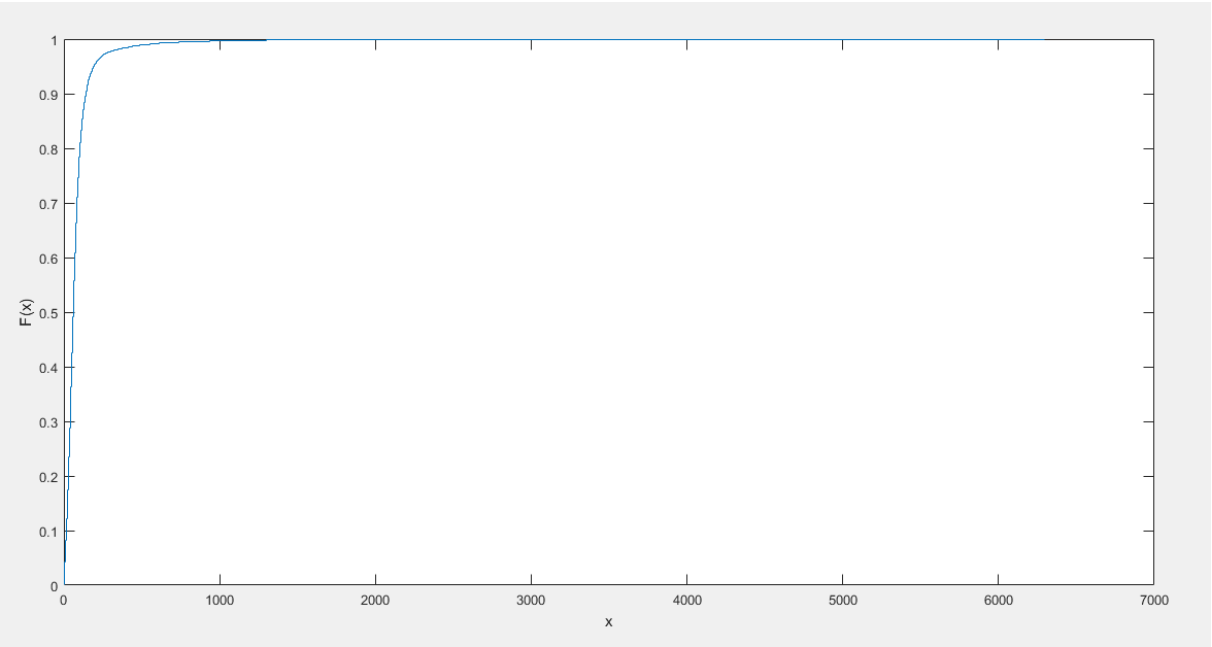


*Figure 7. Empirical cumulative distribution function*

e) Conclusions

Presented results coincided with expectations of how process scheduling should look like. All of variables given in the task description are random thanks to generators provided by lecturer (uniform and exponential generators). While CPU scheduling algorithm (Shortest Job First algorithm) is working perfectly (when processes have the same time there is draw which one should go first, implemented with use of uniform generator), I/O device scheduling algorithm (Random algorithm) reduces efficiency of the system(picking a random index of queue is implemented with use of uniform generator). It can be seen in turnarounds times and – more importantly in empirical cumulative distribution function chart. Turnaround times can reach up to 1000 ms because of random algorithm implemented in I/O queue – some processes have to wait enormous amount of time to get to the I/O device. That have visible impact on system.

My simulation time is 1000000 – that ensures that mean waiting times will be more balanced, that means, they are more aligned.

From perspective of lambda value (rather high) I can tell that this system (with 4 processors and 5 I/O devices) is working well enough – high utilization of CPUs and relatively high throughput are satisfying. There could be made prediction that system would work better with different I/O device scheduling algorithm.