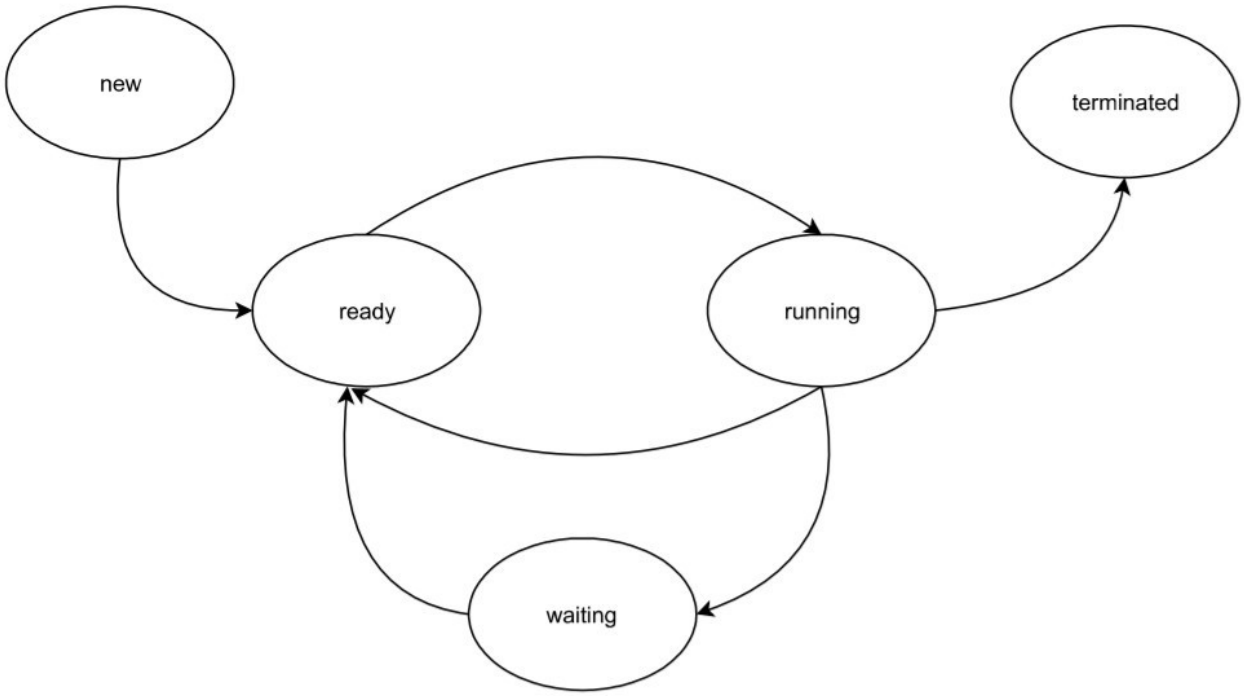
THE PROCESS SCHEDULING

BARTŁOMIEJ PRASELSKI

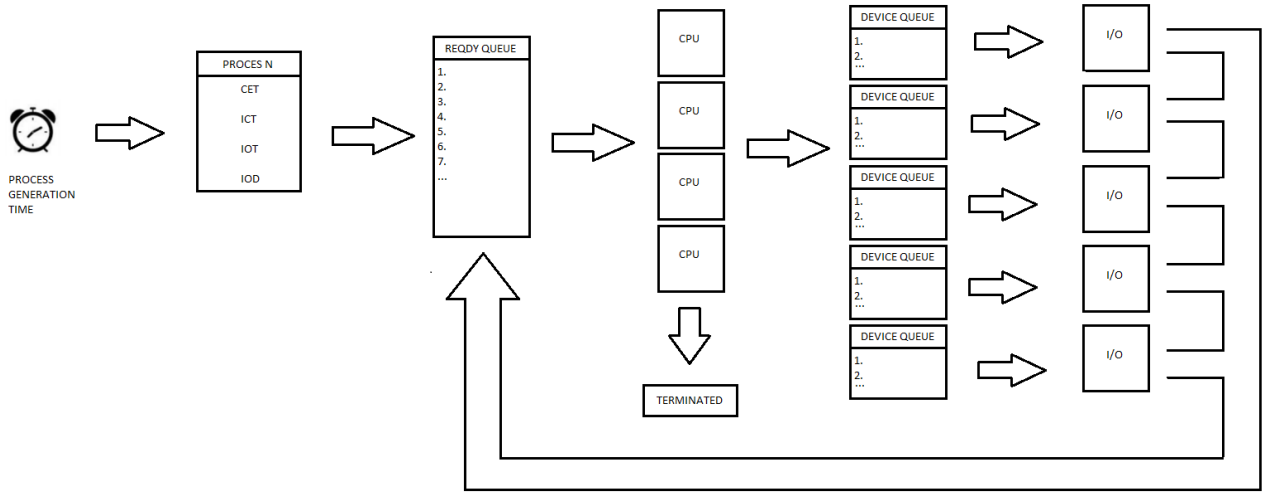
Method : Activity Scanning
CPU scheduling algorithm : FCFS
I/O scheduling algorithm : SJF
Number of processors : 4
Number of I/O devices : 5

1. Description of the projekt

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 automaticaly 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.



2. Simulation model scheme



3. Description of objects and their attributes

OBJECTS	CLASS NAME	DESCRIPTION	ATTRIBUTES
CPU	CPU	Class that represents single CPU.	• *process_cpu* type *Process** - the pointer to process that is currently being served by CPU, • *busy* type *bool* - the variable which checks that CPU is busy or not, • *c_busy* type *int* - the variable which contains time when CPU was busy, • *c_idle* type *int* – the variable which contains time when CPU ends being idle.
CPU Scheduler	CPUscheduler	Class used as a container of all other objects	• *ReadyQueue* type *queue<Process*>* - vector of pointers to processes in queue to CPUs, • *CPUs* type *vector<CPU*>* - vector of pointers to all 4 CPUs, • *IOs* type *vector<IO*>* - vector of pointers to all 5 IOs, • *device_queues* type

			vector<Deviceq>* - vector of pointers to 5 queues to IOs, • *cpuNumb* type *const int* - variable that contains numer of CPUs, • *ioNumb* type *const int* – variable that contains numer of IOs.
Device queue	Deviceq	Class that represents single queue to IO device	• *DeviceQueue* type *vector<Process*>* - vector of pointers to processes in queue to IO device,
IO	IO	Class that represents single IO.	• *process_io* type *Process** - the pointer to process that is currently being served by IO.
Process	Process	Class that represents single Process.	• *Id* type *int* – represents unique number • *CET* type *int* – CPU execution time • *ICT* type *int* – I/O call time • *IOT* type *int* – I/O occupationt time • *IOD* type *int* – I/O device • *start_time* type *int* – indicates the time when process enter the queue • *queue_time* type *int* – indicates the time of being in queue of single process. • *birth* type *int*- contains time at which process was create.
Simulation	Simulation	Class that performs time and conditional events	• *next_arrival* type *double* – time when new process arrives • *sim_time* type *int* – simulation time • *initial_phase* type *int*, • *event_trig* type *bool* • *cpu_sim_ICT[]* type *int[]* - used to sort ICT times

			• *io_sim_ICT[]* type *int[]* - used to sort ICT times • *all_processes* type *int* • *ended_processes* type *int* • *turnaround* type *int* • *lambda* type *double* • *time* type *int*

4. List of time and conditional events

TIME EVENTS	
Event	**Algorithm**
New process	1. Generate a new process 2. Add process to Ready Queue 3. Generate the time of next arrival.

End of work in CPU	1. Take the process from the CPU
	2. If CET time of process is greater than 0
	  1. Update CET and ICT times
	  2. Read the IOD of process
	  3. Delete the process from CPU and put into right device queue
	  4. Set CPU not busy
	3. Else
	  1. Delete the process from CPU
	  2. Set CPU not busy

End of work in IO	1. Take the process from the IO 2. Rand new IOT and ICT times 3. Delete the process from IO and put into ready queue 4. Set IO not busy

CONDITIONAL EVENTS	
Event	**Algorithm**
Start of work in CPU	1. Take the first process form the ready queue 2. Set CPU busy 3. Start of working
Start of work in IO	4. Take the first process form the device queue 5. Set IO busy 6. Start of working

5. Generators

- Description of implemented generators

I used 7 generators in my simulations. Six of them are uniform random generator and last is the exponential random generator. Each of generator has own seeds.

Seeds :

- Initial seed : 10
- Span between seeds : 100 000 draws
- Save every seed after : 100 000 draws

Uniform random 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(M);
```

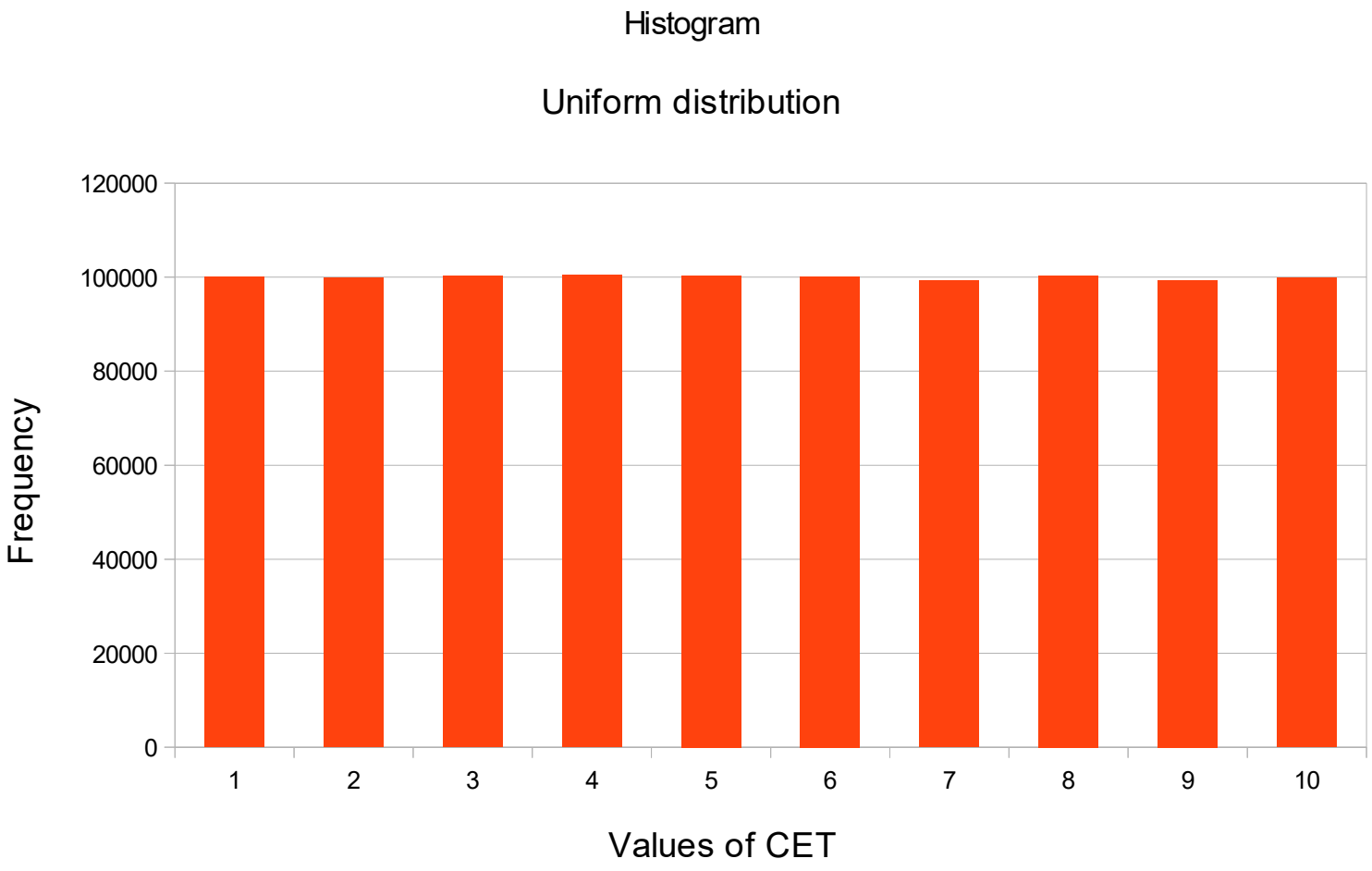
Variable „kernel_" is the current seed. Program calculate new value for kernel and this value will be returned as the new random number and also will be safe as the new seed.
This generators are used to generate times when new processes are created, to choose number of I/O device, to help with sorting process in device queues and to exponential random generator.

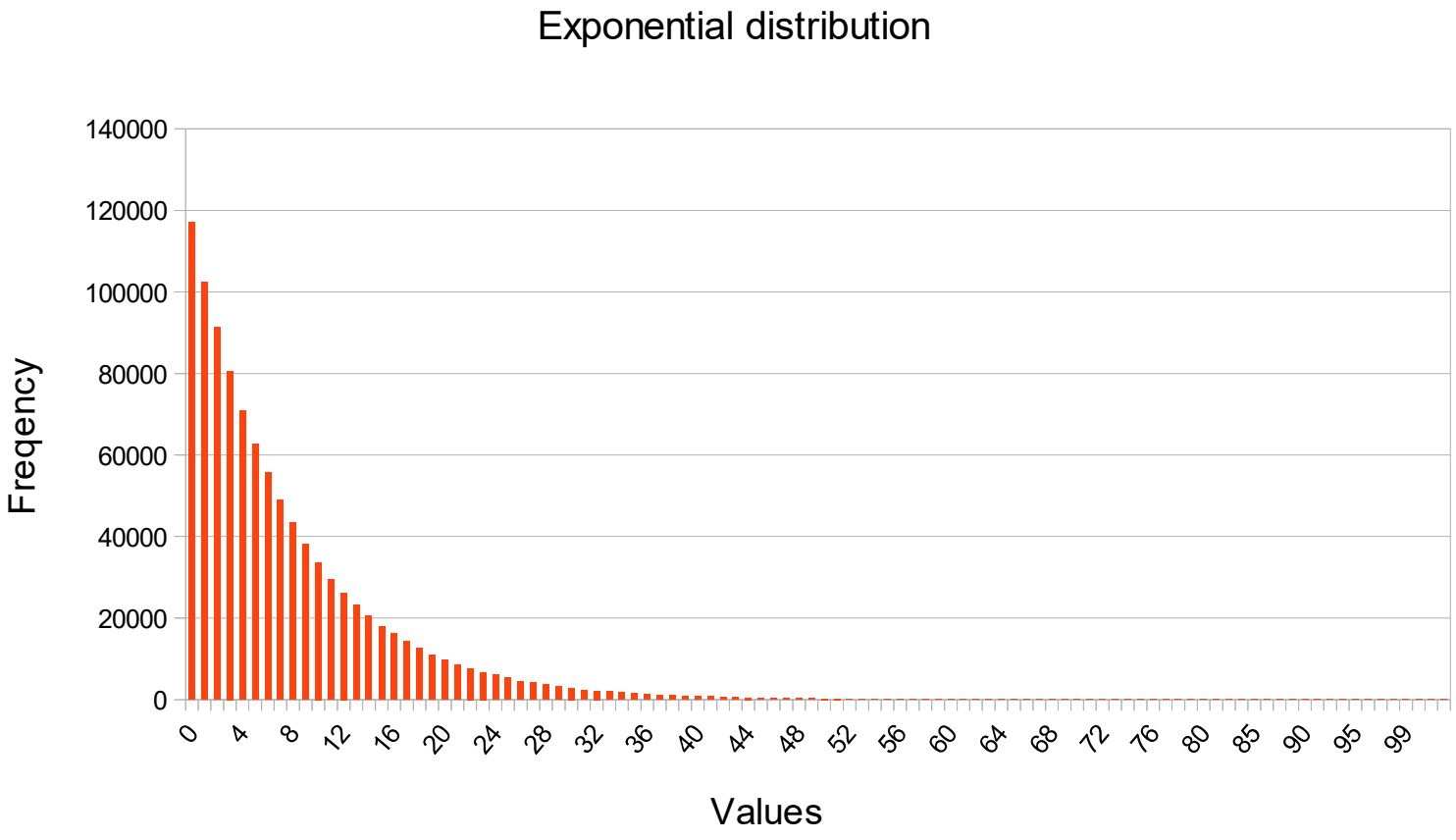
Exponential random generator :

```
double k = uniform_->Rand_1();
 return -(1.0/lambda_)*log(k);
```

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

This generator uses uniform random generator and generate the time at which new process arrives.





This histograms describes the results of uniform random generator and exponential random generator. The histograms confirm that generators work properly.

6. **Program input parameters**
    - Mode – to choose are two modes:
        - Continous : only the output info will be shown
        - Step By Step : at each time when something is executed, informations are displayed
    - Lambda – it is used in exponential random generator (prefer value is 0,127)
    - Initial phase – boolean value, set 1 to ommit initial phase, 0 if not.
    - Sim Time – time of simulation.

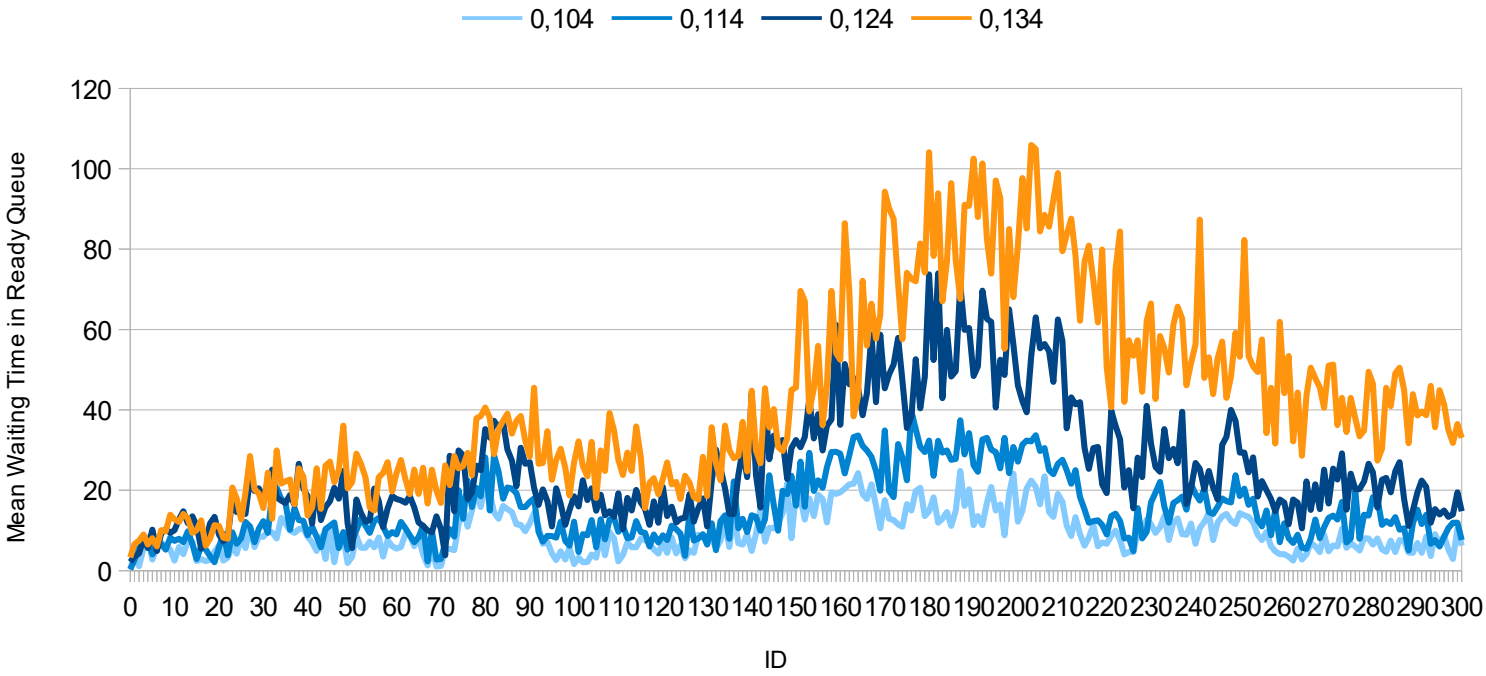
7. **Code testing methods**

To test our simulation we need to set mode to step by step. Displayed informations confirmed that simulation works correctly :

```
--------------------------------------------------------------------------------
SIMULATION TIME : 1699 [ ms ]
--------------------------------------------------------------------------------
Proces nr 193 konczy prace i usuwa sie z CPU nr 3
Proces nr 201 zaczyna prace na CPU nr 3, planowany czas zakonczenia : 1704
Ready Queue size : 7

--------------------------------------------------------------------------------
SIMULATION TIME : 1700 [ ms ]
--------------------------------------------------------------------------------
Proces nr 199 konczy prace na CPU nr 0
Proces nr 192 zaczyna prace na CPU nr 0, planowany czas zakonczenia : 1701
Ready Queue size : 6
Device Queue nr 0 has size : 1

--------------------------------------------------------------------------------
SIMULATION TIME : 1701 [ ms ]
--------------------------------------------------------------------------------
Utworzenie procesu nr 209
Proces nr 192 konczy prace i usuwa sie z CPU nr 0
Proces nr 202 zaczyna prace na CPU nr 0, planowany czas zakonczenia : 1721
Ready Queue size : 6
Device Queue nr 0 has size : 1

--------------------------------------------------------------------------------
SIMULATION TIME : 1704 [ ms ]
--------------------------------------------------------------------------------
Proces nr 201 konczy prace na CPU nr 3
Proces nr 205 zaczyna prace na CPU nr 3, planowany czas zakonczenia : 1712
Ready Queue size : 5
Device Queue nr 0 has size : 2
```

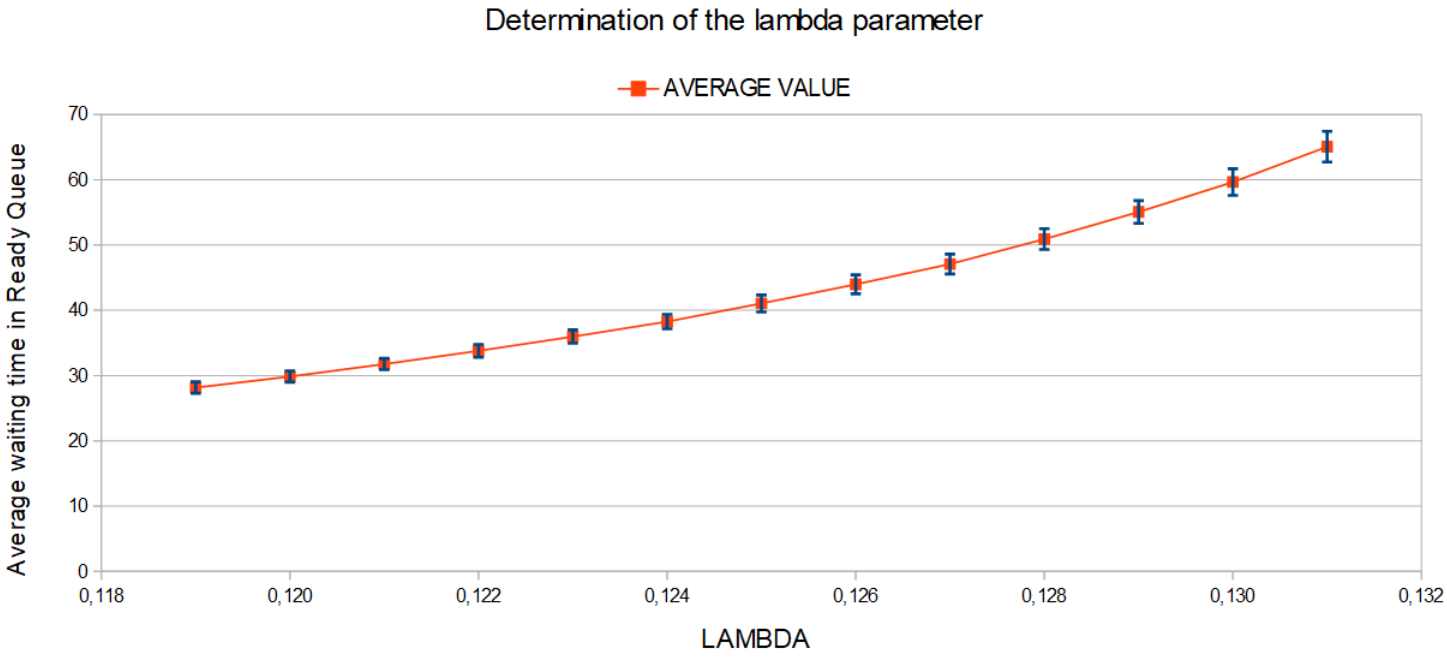
Example:
At time 1699 process nr 201 starts working at CPU nr 3 and it ends at time 1704. Next at time 1704 process 201 end works at CPU nr 3. IO nr 0 is busy so it goes to the device queue nr 0 becaouse this process has IOD = 0.

8. **Simulation results:**
    - **Initial phase determination**



The graph shows the mean waiting time over 10 simulation of each process. To set the initial phase I choose process nr 150. This process ends after 1500 sim time so I decided that initial phase will be 1600.

- **Determination of exponential generator intensity L**

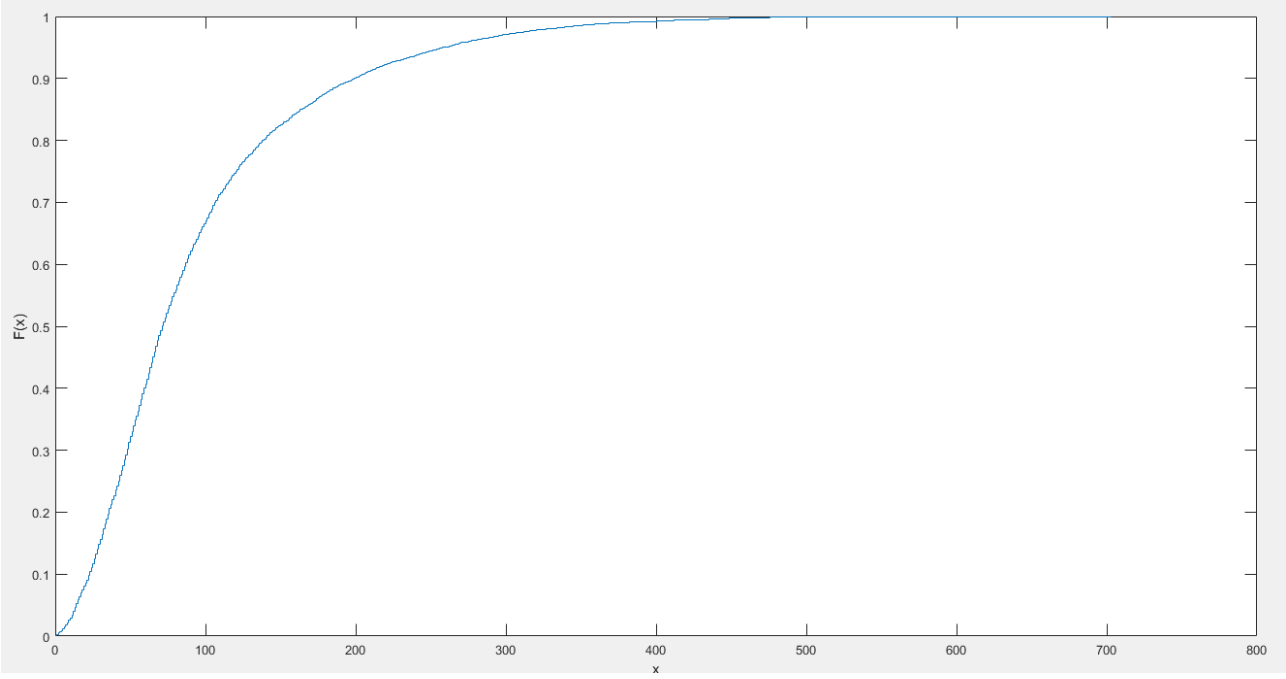


For each lambda I calculated average waiting time in ready queue for 10 simulation and calulate the average. Initial phase was ommited. Blue posts are the convidence intervals. **L is equal to 0,127**

- **Results for at least ten simulations with average values and confidence intervals**

Simulation number	Average waiting time in the ready queue	CPU #1 utilization %	CPU #2 utilization %	CPU #3 utilization %	CPU #4 utilization %	Throughput	Average turnaround time
1	46,2825	92,2213	88,6888	84,3009	79,0279	0,13521700	89,4005
2	45,4162	92,2177	88,8359	84,2072	78,8968	0,13512400	88,6154
3	46,6138	92,3838	89,1520	84,6101	79,5943	0,13586400	89,6844
4	44,2650	92,1612	88,6450	84,2399	79,0635	0,13529700	87,4099
5	49,0969	92,4460	89,2427	84,9301	79,8301	0,13607500	92,2895
6	45,2848	92,3938	89,1708	84,7292	79,4970	0,13579500	88,5061
7	50,1090	92,3996	89,0867	84,6422	79,4070	0,13550500	93,3877
8	47,1039	92,1634	88,7250	84,1970	78,9531	0,13521600	90,2773
9	46,6602	92,4724	89,0551	84,7202	79,6063	0,13541500	90,0083
10	49,9710	92,5220	89,2123	84,9672	80,0007	0,13550400	93,3116
Average	47,0803	92,3381	88,9814	84,5544	79,3877	0,13550120	90,2891
Confidence interval	1,5195	0,1013	0,1757	0,2237	0,2908	0,00023855	1,5544

- **Empirical cumulative distribution function chart of turnaround time samples from ten simulations**



- **Conclusions**

Simulation works very good. Algorithms that I used (FCFS and SJF) were fine. I think to get better resoults in CPUs utilizations we should use SJF algorithm instead of FCFS. It should decrease the size of the ready queue and increase number of executed processes.