Università degli Studi di Torino
(a.a. 2017 - 2018)
Laurea Magistrale
INFORMATICA
Corso di
Valutazione delle Prestazioni: Simulazione e Modelli (Simulation and Modelling

Master of Science
STOCHASTIC AND DATA SCIENCE
Course in
Simulation
Docente/Instructor
Gianfranco Balbo

November 24, 2017 **Final Project** Due ?

The following are the proposals of two projects that can be used to complete the implementation work required by this introductory course on Discrete Event Simulation.

The topics addressed by these two proposals are quite different, but the corresponding models are instead very similar.
Choose the project that you prefer to develop and start working at the design and the implementation of the simulator as soon as possible.

When you have a good idea on how to implement your simulator, come to my office (or contact me by e-mail) to tell me how you want to proceed so that I can make sure that you are not heading in a wrong direction with a consequent waste of time.

The project involves the implementation of algorithms and methods developed in the last home-works. I suggest to proceed with the completion of the homeworks first where these features are debugged and tuned in a simpler environment that should allow you to get the desired answers in a shorter time.

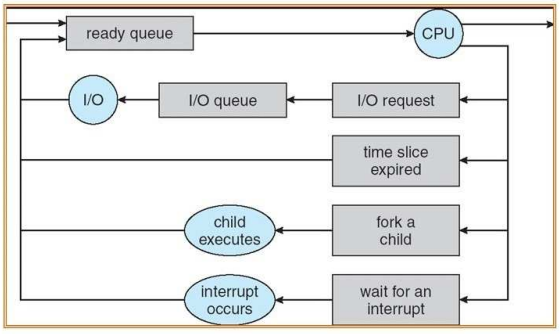
When you have terminated your work, prepare a final report that you will submit together with a complete version of the code you have developed so that I can run it in case I have doubts about the implementation and/or the results that you are obtaining.

Good luck with your work!

Project Nr. 1: Operating System Scheduling

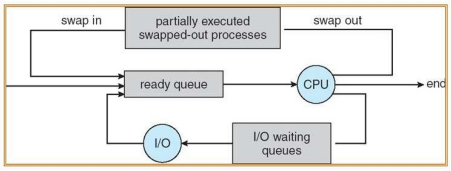
System Description

Processes[1] are programs in execution that are managed by the Operating System (of a computer) to perform associated tasks. As processes enter the system, they are put in a *job list* which consists of all processes in the system. Processes that are residing in main memory and are ready or waiting to execute are kept on a list called the *ready queue*. When a process is allocated the CPU, it executes for a while and eventually quits, is interrupted, or waits for the occurrence of a particular event, such as the completion of an I/O request. Suppose the process makes an I/O request to a shared device, such as a disk. Since there are many processes in the system, the disk may be busy with the I/O request of some other process. The process therefore may have to wait for the disk. The list of processes waiting for a particular I/O device is called a *device queue*. Each device has its own queue. A common representation of process scheduling is a *queueing diagram* such as that depicted here.



The execution of a process is characterized by its switching between the *ready*, *executing*, and *waiting* states. A process continues this cycle until it terminates, at which time releases all the resources that it was using, and is removed from all queues and from the job list as well.

Actually, a process to be executed needs to have some memory allocated and usually a limited number of processes are allowed to reside concurrently in main memory. The number of processes in memory is managed by a long-term scheduler that controls the *degree of multiprogramming* by swapping in and out of memory processes according to certain memory management rules. The overall behavior of a process can thus be also represented by this higher level diagram that combines the effects of the short-term and long-term schedulers of an operating system.



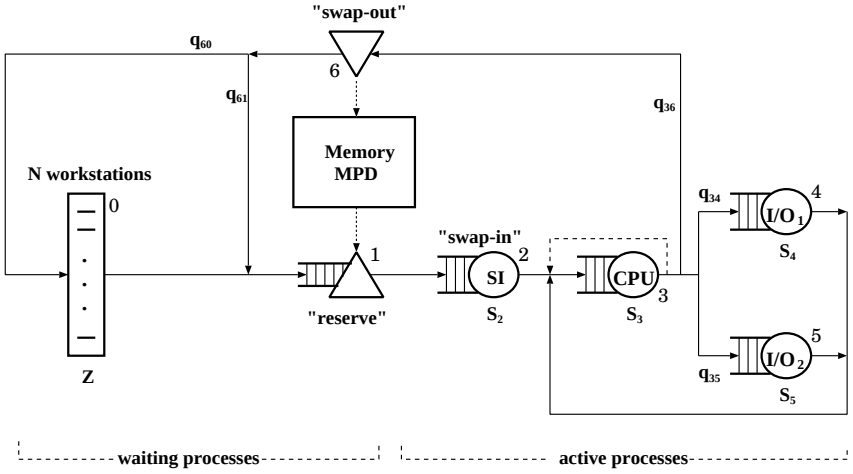
[1] Concepts and pictures contained in this "System description" are taken from one of the standard textbooks used for the introductory courses on Operating Systems: **A. Silberschatz, P.B. Galvin, G. Gagne**, *"Operating System Concepts"*, 8-th Edition, John Wiley & Sons, 2009

Besides the time required to carry on all the specific tasks, the performance of this system is very much affected by two elements that may be worth studying: (i) the length of the time-slice that defines an upper limit for the amount of CPU time that may be continuously devoted to a single process (introduced to avoid that a very long CPU-bound job prevents other jobs from using the CPU for very long times), and (ii) the multiprogramming degree that limits the number of processes concurrently active in the system. We can thus think of studying this system from the point of view of its response time to jobs submitted by a pool of work-stations (or terminals) connected to the system when these two parameters change together with the size of the pool of workstations.

Modelling assumptions

The behavior of this system is quite complex also because several other aspects should be taken into consideration to really provide complete guidelines to assist the system administrator in choosing the relevant parameters of a system configuration (notice that this basic mechanism is also present in the systems running modern data centers where the role of the administrator is often played by control algorithms that optimize the performance of the system with load balancing and reconfiguration actions).

For the purpose of this project, we can assume that the number N of work-stations connected to the system is sufficiently large and that these work-stations are always occupied by users.
Moreover, we assume (i) that all the queues in the system are managed with FCFS policies, but the workstation pool that behaves as an infinite Server station, (ii) that the intrinsic variability of all the operations represented in the system is captured by assuming that the service times of all the components of this system are represented by random variables, and (iii) that all the choices concerning the routing of the processes among the different nodes of the model are probabilistically decided.
A model of the operation of this system (often called a Central Server System) can be represented by the following queueing network.



A job cycle in this system can be described in the following manner. A work-station operates locally for a certain amount of time and then submits a request to the Central Server System and an associated process is created.
To become active the process needs the allocation of a portion of memory and waits at the "reserve" station to be admitted in the system. The admission is made on the basis of a Multi-Programming Degree (MPD) that establishes the maximum number of processes allowed

to be simultaneously active. Whenever there is room, the process is admitted by the Multi-Programming Degree Controller and its image is loaded into main memory by the node "swap-in". When the memory is properly loaded the process is finally ready to be executed and it is inserted in the CPU queue.
When the CPU is allocated to the process, a processing time is defined on the basis of possible interruptions due to the need for I/O operations, or to the completion of the computation. In any case, if this processing time is larger than the time-slice (also called CPU quantum), a premature interruption occurs and the process is returned at the end of the CPU queue to continue its processing for the remaining time until the natural conclusion of its "service" when the CPU will become available again (if no other processes are in the CPU queue at that time this continuation takes place immediately).
When the process leaves the CPU to perform some I/O operation it is moved to the input queue of the corresponding device and will return to the ("ready") queue of the CPU when the data transfer is completed. If instead the process leaves the CPU because of the completion of the computation, its memory is released in favour of some other process waiting to be admitted by the Multi-Programming Degree Controller and a decision must be taken to distinguish the case of a true completion (which corresponds to the termination of the process and to the corresponding completion of the job originally submitted by the work-station that can now use the job's result to start a new local computation), from the case of a "swap-out" action that brings the process to the reserve queue again. To keep the model simple, we assume that memory release (swap-out) takes a negligible amount of time and is not explicitly represented in the model. Notice that the "reserve" queue (managed by the MPD Controller) is formed in front of a node which operates differently from the others and whose service time depends on the behavior of the active processes that allow a new process to get in the system only when one of them is swapped out (unless there is room in the system because the number of active processes is smaller than MPD)
The presence of a time-slice to control the length of the CPU burst and of a Multi-Programming Degree controller make the analysis of this model complex, especially when the CPU service requests have high variability. We must thus use Discrete Event Simulation for assessing the behavior of this system.

Given all these assumptions, the parameters of the model summarized in the previous figure are the following:

- Average work-station think time $Z = 5\,sec$,
- Average swap-in time $S_1 = 210\,msec$,
- Average CPU time $S_2 = 27\,msec$
- Length of the time slice (CPU quantum) $\delta = 3\,msec$
- Average I/O_1 time $s_4 = 40\,msec$
- Average I/O_2 time $s_4 = 180\,msec$
- CPU completion choice $q_{3,4} = 0.65, \quad q_{3,5} = 0.25, \quad q_{3,6} = 0.1$
- Swap-out choice $q_{6,0} = 0.4, \quad q_{6,1} = 0.6.$
- Multi-Programming Degree $MPD = 10$

All the stations are characterized by negative exponential distributions, but the CPU in which the time slice is deterministic (constant) and the total length of the CPU burst has a two stage hyper-exponential distribution

$$f_X(x) = \alpha * 1/\mu_1 * exp(-x/\mu 1) + \beta * 1/\mu_2 * exp(-x/\mu 2)$$

with parameters $\alpha = 0.8$, $\beta = 0.2$, $\mu_1 = 15\,msec$, and $\mu_2 = 75\,msec$.

Analysis

- Write the simulator of this simple system assuming the presence of 20 work-stations, providing interval estimates for the
  - average response time
  - and for the average active time, where the active time corresponds to the time taken by a process to move from the "reserve" to the "swap-out" nodes (*).

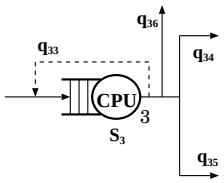
  Use different random number streams for the different random variables represented in the model.

  Use the regenerative method to compute the confidence intervals, starting the simulation from a suited regeneration point.

  Provide the results with a 90% confidence level and a precision of $\mp 5\%$.

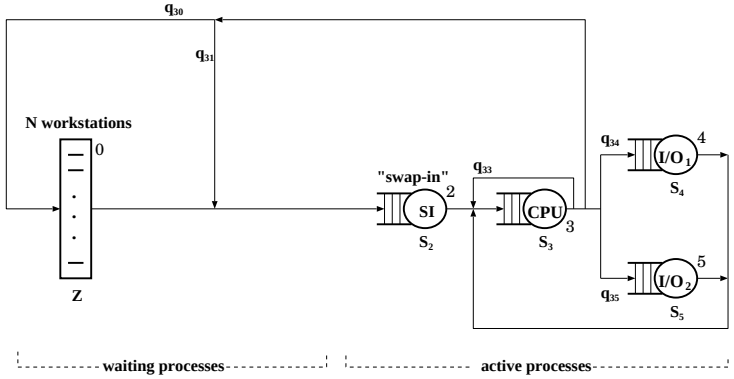
- In order to check the correctness of the simulator that you have just implemented, you need to compare the results that you are obtaining with those provided with an alternative method applied to a simplified version of the model.

  Indeed, the operation of the CPU can be captured in an approximate manner by assuming that its service time is negative exponential distributed with average $s_3 = 3\,msec$ and that, upon completion of this short service the process returns to the CPU input queue with probability $q_{3,3} = 0.9$, while the probability of the other routes are now $q_{3,4} = 0.065$, $q_{3,5} = 0.025$, $\quad q_{3,6} = 0.01$.



  With this simple modification, and assuming that the MPD is greater than the number of work-stations, the model becomes a Product Form Queueing Network for which a preliminary Bottleneck Analysis can be performed, before computing the average response and active times using the Mean Value Analysis (MVA) algorithm and assuming that the number of work-stations connected to the system varies from 1 up to 30.

  The resulting simplified model is represented in the following picture



- To improve your confidence on the correctness of your simulator, check first that the interval estimate(s) includes the theoretical value obtained with the MVA algorithm.
- To improve further your confidence on your simulator, perform an extensive validation using the technique discussed at the end of the course.

Report

Summarize the results of your work writing a brief document in which you

- describe the basic features of the simulator you have implemented;
  - define the regeneration point you have used in your simulation;
  - provide a print-out of the state of the system after the handling of each of the first 30 events processed by the "engine" of your simulator;
  - implement a feature that allows to turn on/off the possibility of printing the measures performed during each regeneration cycle and used to compute the confidence intervals requested by the text of the project;
- devote a particular attention to the description of the validation process you have used.
- comment on the performance of the system you have simulated providing the results for the following four cases
  1. time slice $\delta = 3\,msec$; CPU burst with hyper-exponential distribution with parameters $\alpha = 0.8$, $\beta = 0.2$, $\mu_1 = 15\,msec$, and $\mu_2 = 75\,msec$.
  2. time slice $\delta = 3\,msec$; CPU burst with negative exponential distribution with parameter $\mu = 27\,msec$
  3. time slice $\delta = 3000\,msec$; CPU burst with hyper-exponential distribution with parameters $\alpha = 0.8$, $\beta = 0.2$, $\mu_1 = 15\,msec$, and $\mu_2 = 75\,msec$.
  4. time slice $\delta = 3000\,msec$; CPU burst with negative exponential distribution with parameter $\mu = 27\,msec$

  In particular, comment on the combined impact of the variance of the CPU burst distribution and of the length of the time slice.

Hints:

- To implement the time-slice mechanism (Round-Robin Queueing Policy) in your simulator, consider that every-time the CPU is allocated to a process two possibilities exist: either the service time requested by the process is shorter than the time-slice and the process leaves the CPU at the end of this service period or the time-slice is shorter than the requested service time and the process returns to the CPU queue at the end of the time slice with a new service request that accounts (it is shorter!) for the received service.
- The validation of the simulator against the results obtained with the MVA algorithm, is more effective if smaller are the changes introduced in the simulator to adapt it to the simplified model.

Having implemented the detailed simulator you can use it for validation purposes by simply introducing the following modifications:

- Initialize the Multi-Programming Degree (MPD) to a value larger than the number of connected work-stations
- Set the time-slice value (CPU quantum) to a very high value to eliminate its influence in the execution of the simulation
- In the probabilistic choice of the routes taken by the processes, add the possibility for the processes to immediately return to the CPU's queue which can be turned on or off depending on the value that you assign to the corresponding probability.

Project Nr. 2: Flexible Manufacturing Cell

System Description

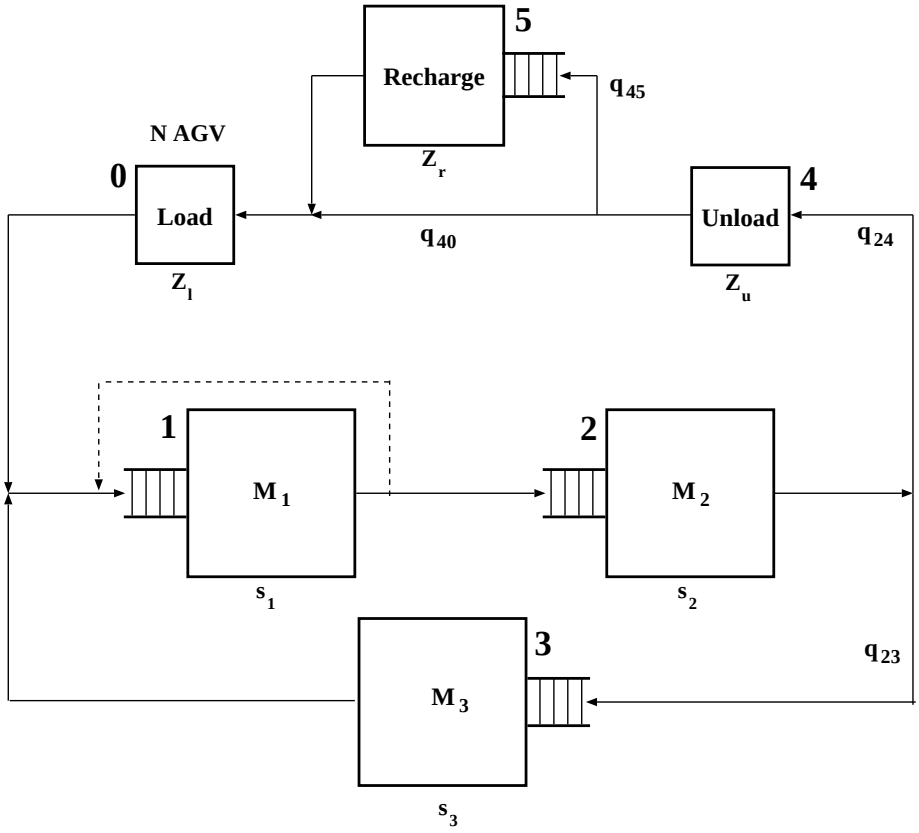
Consider a Flexible Cell of an automated factory (Flexible Manufacturing System - FMS) where a set of operations is performed on raw parts in order to obtain a final product.
In a FMS, raw parts are usually loaded on intelligent carts (Automated Guided Vehicles - AGV) to be processed by several machine-tools.
In this system, we consider a very simplified version where only three machine-tools are present. The complete manufacturing of these parts requires several interventions of the three machines. The whole manufacturing process thus consists of loading a part on an AGV, moving the part to the machine-tools several times as needed, removing the manufactured part from the AGV.
When a part is unloaded, the AGV becomes available for carrying a new part. Occasionally the AGVs (which are moved by electric motors, powered by electric batteries) need to stop for a while at a recharge/maintenance station to become ready again for working, later on. A fixed number of AGVs is present in this manufacturing plant, while raw parts are assumed to be stocked next to the loading station so that no AGV remains idle at any point in time waiting for a raw part to become available.
The first machine (M_1) is a sophisticated high precision cell which uses tools that must frequently be re-calibrated and sharpened so that its individual operations are quite short. To simplify the description of the system, assume that the machine is grinding surfaces with an extremely small tolerance.
The second machine is capable of performing complex operations, but on completion of its work a quality control is performed that may send back the part to machine M_1 to be worked again, but only after the intervention of machine M_3 that returns the rejected part to a status that allows for machine M_1 to operate as if the part was still raw. Referring again to machine M_1, we may observe that, when the tool wears out the operation has not been completed yet, the old tool is replaced by a new one and the machine is ready to continue its work. Since the total operation times of the raw parts may be quite different, to avoid long waiting, the machine takes advantage of the need for the tool replacement to give the possibility to another part to start receiving service. The part that was being manufactured is moved at the end of the input queue of the machine and a new part (if there exist one in that moment) is taken in charge by the machine.
A part that has been partially manufactured will be worked by machine M_1 later on (unless it is the only part asking for service from M_1 in that precise moment) to continue with the remaining grinding and will leave this machine-tool, only if the production is completed during the operation of the new tool, otherwise the process is repeated as many times as needed. Notice that with the working organization of machine M_1 implies that the moving of parts from machine M_1 to machine M_2 is determined by the completion or not of the grinding operation.

Modelling assumptions

Real systems of this type are very complex and even the small version considered before would require quite a lot of additional specifications to make the description complete.

For the purpose of this project, we can assume that (1) the number N of AGVs available in this system is sufficiently large, that (2) the input buffers of the machine-tools are large enough to accommodate all the AGVs that may require service, that (3) all the queues in the system are managed with FCFS policies, that (4) the intrinsic variability of all the operations represented

in the system are captured by assuming that the service times of all the components of this system are represented by random variables, and that (5) all the choices concerning the routing of the AGVs among the different nodes of the model are probabilistically decided. Moreover, we can also assume that when the grinding of a part is completed the partially consumed old tool is nevertheless replaced by a new one before the machine becomes ready to continue its work (notice however that the overhead of installing the new tool is not explicitly considered in the model and is assumed to be part of tool consumption).
A model o^ this manufacturing system can be represented by the following queueing network.



Notice that in this system the association of a part to an AGV wich occurs in the loading station allows to use AGVs and parts as synonyms in the sequel of this description.
The loading and unloading stations are assumed to be of type *Infinite Server*. The recharge station is a single server queueing station. Machines M_1, M_2, and M_3 are single server queueing stations as well.
To limit the impact of the recharge/maintenance station on the behavior of the model we assume that a preventive maintenance policy is followed that makes the stops of the AGVs relatively frequent, but for individual service times that are not too large.
The operation of machine M_1 is the element that makes the analysis of this model complex because of the necessity of considering parts with large variability of their grinding times and because of the mechanism which implements tool consumption and part turnover. We must thus use Discrete Event Simulation for assessing the behavior of this system. Given all these assumptions, the parameters of the model are the following:

- Average loading time $Zl = 60\,sec$,
- Average un-loading time $Zu = 50\,sec$,

- Average recharge/maintenance time $Sr = 90\,sec$
- Tool Consumption time $\delta = 10\,sec$
- Average grinding time of machine M_1 $s_1 = 90\,sec$
- Average operational time of machine M_2 $s_2 = 80\,sec$
- Average operational time of machine M_3 $s_3 = 100\,sec$
- Quality control choice $q_{2,3} = 0.2, \quad q_{2,4} = 0.8,$
- Recharge/maintenance choice $q_{4,0} = 0.6, \quad q_{4,5} = 0.4.$

All the stations are characterized by negative exponential distributions, but the machine M_1 in which the tool consumption time is deterministic (constant) and the total grinding time has a two stage hyper-exponential distribution

$$f_X(x) = \alpha * 1/\mu_1 * exp(-x/\mu 1) + \beta * 1/\mu_2 * exp(-x/\mu 2)$$

with parameters $\alpha = 0.8$, $\beta = 0.2$, $\mu_1 = 50\,sec$, and $\mu_2 = 250\,sec$.

Analysis

- Write the simulator of this simple system assuming the presence of 10 AGVs (in total) and providing interval estimates for
  - the average cycle time
  - and manufacturing times, where the manufacturing time corresponds to the time taken by a part to be completed (i.e., excluding loading and un-loading times)(*).

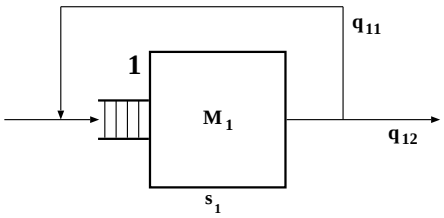
  Use different random number streams for the different random variables represented in the model.

  Use the regenerative method to compute the confidence intervals, starting the simulation from a suited regeneration point.

  Provide the results with a 90% confidence level and a precision of $\mp 5\%$.

- In order to check the correctness of the simulator that you have just implemented, you need to compare the results that you are obtaining with those provided with an alternative method applied to a simplified version of the model.

  The operation of machine M_1 can be captured in an approximate manner by assuming that its service time is negative exponential distributed with average $s_1 = 10\,sec$ and that, upon completion of this short service the AGVs move to machine M_2 with probability $q_{12} = 0.1$, while they return to the queue of the same machine M_1 with probability $q_{11} = 0.9$.



  With this simple modification, the model becomes a Product Form Queueing Network for which a Bottleneck Analysis can be preliminary performed, before computing the aver-

age cycle and manufacturing times using the Mean Value Analysis (MVA) algorithm and assuming that the number of AGVs present in the system varies from 1 up to 30.

- To improve your confidence on the correctness of your simulator, check first that the interval estimate(s) includes the theoretical value obtained with the MVA algorithm.
- To improve further your confidence on your simulator, perform an extensive validation using the technique discussed at the end of the course.

Report

Summarize the results of your work writing a brief document in which you

- describe the basic features of the simulator you have implemented;
  - define the regeneration point you have used in your simulation;
  - provide a print-out of the state of the system after the handling of each of the first 30 events processed by the "engine" of your simulator;
  - implement a feature that allows to turn on/off the possibility of printing the measures performed during each regeneration cycle and used to compute the confidence intervals requested by the text of the project;
- devote a particular attention to the description of the validation process you have used.
- comment on the performance of the system you have simulated providing the results for the following four cases

  1. Tool Consumption time $\delta = 10\,sec$; grinding time with hyper-exponential distribution with parameters $\alpha = 0.8$, $\beta = 0.2$, $\mu_1 = 50\,sec$, and $\mu_2 = 250\,sec$.
  2. Tool Consumption time $\delta = 10\,sec$; grinding time with negative exponential distribution with parameter $\mu = 90\,sec$
  3. Tool Consumption time $\delta = 10000\,sec$; grinding time with hyper-exponential distribution with parameters $\alpha = 0.8$, $\beta = 0.2$, $\mu_1 = 50\,sec$, and $\mu_2 = 250\,sec$.
  4. Tool Consumption time $\delta = 10000\,sec$; grinding time with negative exponential distribution with parameter $\mu = 90\,sec$

  In particular, comment on the combined impact of the variance of the grinding time distribution and of the Tool Consumption time.

Hints:

- To implement the Tool Consumption/Replacement mechanism (Round-Robin Queueing Policy) in your simulator, consider that every time machine M_1 starts operating on a part (raw or already partially grinded) two possibilities exist: either the grinding time requested

by the part is shorter than the tool consumption time so that the work of machine m_1 on this part is completed and the AGV moves to machine M_2, or the tool is worn out before the end of the grinding request and the AGV returns to the input queue of machine M_1 with a part that will require next time a shorter operation time due to the processing that it just received .

- The validation of the simulator against the results obtained with the MVA algorithm, is more effective if smaller are the changes introduced in the simulator to adapt it to the simplified model.

  Having implemented the detailed simulator you can use it for validation purposes by simply introducing the following modifications:

  - Set the Tool Consumption time to a very high value to eliminate its influence in the operation of machine M_1.
  - On exit from machine M_1, introduce the possibility for the AGV of choosing whether to continue directly to machine M_2 or to return to the input queue of machine M_1 again. Associate with this possibility a probability value that you can use to turn on and off this possible choice.