Design and Development of a CPU Scheduler Simulator for Educational Purposes Using SDL

Manuel Rodríguez-Cayetano

Department of Signal Theory and Telematics Engineering
University of Valladolid, ES-47011 Valladolid, Spain
manuel.rodriguez@tel.uva.es

6th Workshop on System Analysis and Modelling
Oslo, October 4th-5th 2010

Summary

Summary

Summary

Summary

Summary

Summary

Summary

Summary

Summary

1. CPU Scheduling Overview
   - CPU Scheduling Basics
   - Related work
2. The SDL CPU Scheduler Simulator
   - Requirements
   - Behavior Overview
   - Simulator structure
   - Detailed Behavior
3. The CPU Scheduler Simulator Graphical User Interface
4. Conclusions and Further Work

Summary

CPU Scheduling Basics (I)

- CPU scheduling: deciding which of the ready processes is to be allocated the CPU
- Different selection criteria $\Rightarrow$ different scheduling algorithms
- Two types of scheduling algorithms:
  - one queue algorithms: appropriate when all the processes belong to the same class (same scheduling requirements)
  - multilevel queue algorithms: appropriate for processes belonging to several classes (different scheduling requirements)
- Every algorithm may favor one class of process over another due its properties:
  - performance evaluation parameters (throughput, turnaround time, waiting time, ...) are used for comparing CPU scheduling algorithms

CPU Scheduling Basics (I)

- CPU scheduling: deciding which of the ready processes is to be allocated the CPU
- Different selection criteria $\Rightarrow$ different scheduling algorithms
- Two types of scheduling algorithms:
  - one queue algorithms: appropriate when all the processes belong to the same class (same scheduling requirements)
  - multilevel queue algorithms: appropriate for processes belonging to several classes (different scheduling requirements)
- Every algorithm may favor one class of process over another due its properties:
  - performance evaluation parameters (throughput, turnaround time, waiting time, ...) are used for comparing CPU scheduling algorithms

CPU Scheduling Basics (I)

- CPU scheduling: deciding which of the ready processes is to be allocated the CPU
- Different selection criteria $\Rightarrow$ different scheduling algorithms
- Two types of scheduling algorithms:
  - one queue algorithms: appropriate when all the processes belong to the same class (same scheduling requirements)
  - multilevel queue algorithms: appropriate for processes belonging to several classes (different scheduling requirements)
- Every algorithm may favor one class of process over another due its properties:
  - performance evaluation parameters (throughput, turnaround time, waiting time, ...) are used for comparing CPU scheduling algorithms

CPU Scheduling Basics (I)

- CPU scheduling: deciding which of the ready processes is to be allocated the CPU
- Different selection criteria $\Rightarrow$ different scheduling algorithms
- Two types of scheduling algorithms:
  - one queue algorithms: appropriate when all the processes belong to the same class (same scheduling requirements)
  - multilevel queue algorithms: appropriate for processes belonging to several classes (different scheduling requirements)
- Every algorithm may favor one class of process over another due its properties:
  - performance evaluation parameters (throughput, turnaround time, waiting time, ...) are used for comparing CPU scheduling algorithms

CPU Scheduling Basics (II)

- Selecting an algorithm appropriate for a process workload $\Rightarrow$ evaluation of the algorithms based on values of the performance evaluation parameters
  - for example, select the algorithm that produces the least mean waiting time for a process workload
- Several evaluation methods:
  - deterministic modeling: results only valid for the concrete (deterministic) workload used
  - queuing models based: classes of algorithms and statistical distributions of process parameters limited, results not accurate
  - simulation: can support deterministic and statistical based workloads, results can be more accurate than those obtained using queuing models

CPU Scheduling Basics (II)

- Selecting an algorithm appropriate for a process workload $\Rightarrow$ evaluation of the algorithms based on values of the performance evaluation parameters
  - for example, select the algorithm that produces the least mean waiting time for a process workload
- Several evaluation methods:
  - deterministic modeling: results only valid for the concrete (deterministic) workload used
  - queuing models based: classes of algorithms and statistical distributions of process parameters limited, results not accurate
  - simulation: can support deterministic and statistical based workloads, results can be more accurate than those obtained using queuing models

CPU Scheduling Basics (III)

- Simulation consist of programming a model of the computer system that behaves like the actual system (at least regarding CPU scheduling) $\Rightarrow$ modeling:
  - arrival, ready and finished queues
  - clock
  - ...
- Design, coding and debugging of a simulator are usually a major task
  - appropriate specification techniques (like SDL) should be used instead of just coding

CPU Scheduling Basics (III)

- Simulation consist of programming a model of the computer system that behaves like the actual system (at least regarding CPU scheduling) $\Rightarrow$ modeling:
  - arrival, ready and finished queues
  - clock
  - ...
- Design, coding and debugging of a simulator are usually a major task
  - appropriate specification techniques (like SDL) should be used instead of just coding

Related work and motivation

- Several CPU scheduling simulators have been previously developed: *CPU Scheduling Simulator*, *CPU Scheduler Application*, *Process Scheduling Simulator*, *MOSS Scheduling Simulator* ...
- Main shortcomings of these simulators:
  - algorithms specific for real-time processes are not supported
  - multilevel queue algorithms are not supported, only single-queue ones
  - deterministic workloads (useful for testing special scenarios) not supported in some of them
- These shortcomings have led to the development of a CPU scheduling simulator using SDL (*sdlCPUSched*)

Related work and motivation

- Several CPU scheduling simulators have been previously developed: *CPU Scheduling Simulator*, *CPU Scheduler Application*, *Process Scheduling Simulator*, *MOSS Scheduling Simulator* ...
- Main shortcomings of these simulators:
  - algorithms specific for real-time processes are not supported
  - multilevel queue algorithms are not supported, only single-queue ones
  - deterministic workloads (useful for testing special scenarios) not supported in some of them
- These shortcomings have led to the development of a CPU scheduling simulator using SDL (*sdlCPUSched*)

Related work and motivation

- Several CPU scheduling simulators have been previously developed: *CPU Scheduling Simulator*, *CPU Scheduler Application*, *Process Scheduling Simulator*, *MOSS Scheduling Simulator* ...
- Main shortcomings of these simulators:
  - algorithms specific for real-time processes are not supported
  - multilevel queue algorithms are not supported, only single-queue ones
  - deterministic workloads (useful for testing special scenarios) not supported in some of them
- These shortcomings have led to the development of a CPU scheduling simulator using SDL (*sdlCPUSched*)

sdlCPUSched: requirements and assumptions

- Main requirements:
  - simulator for educational purposes, mainly used for behavior and performance analysis of CPU scheduling algorithms
  - support for non real-time and real-time algorithms
  - support for multilevel queue algorithms (number of queues, queue algorithm and queue priority can be configured)
- Main assumptions (to simplify the development of the simulator):
  - all the processes characteristics are independent
  - only one CPU burst and zero I/O bursts per process

sdlCPUSched: requirements and assumptions

- Main requirements:
  - simulator for educational purposes, mainly used for behavior and performance analysis of CPU scheduling algorithms
  - support for non real-time and real-time algorithms
  - support for multilevel queue algorithms (number of queues, queue algorithm and queue priority can be configured)
- Main assumptions (to simplify the development of the simulator):
  - all the processes characteristics are independent
  - only one CPU burst and zero I/O bursts per process

sdlCPUSched: behavior overview

- Signal-based communication between the simulator and its environment
  - interactions are mainly asynchronous (only synchronous during the phase of simulation configuration)
- Input signals:
  - simulation configuration (algorithm simulated, process parameters...)
  - simulation control (start, stop, pause...)
- Output signals:
  - confirmation of simulation configuration
  - notification of simulation events (if verbose mode is selected)
  - values of per-process and global statistics
- Scenarios of interactions between the simulator and its environment modeled by MSCs
  - main scenarios: algorithm configuration, process workload configuration and simulation execution

sdlCPUSched: behavior overview

- Signal-based communication between the simulator and its environment
  - interactions are mainly asynchronous (only synchronous during the phase of simulation configuration)
- Input signals:
  - simulation configuration (algorithm simulated, process parameters...)
  - simulation control (start, stop, pause...)
- Output signals:
  - confirmation of simulation configuration
  - notification of simulation events (if verbose mode is selected)
  - values of per-process and global statistics
- Scenarios of interactions between the simulator and its environment modeled by MSCs
  - main scenarios: algorithm configuration, process workload configuration and simulation execution

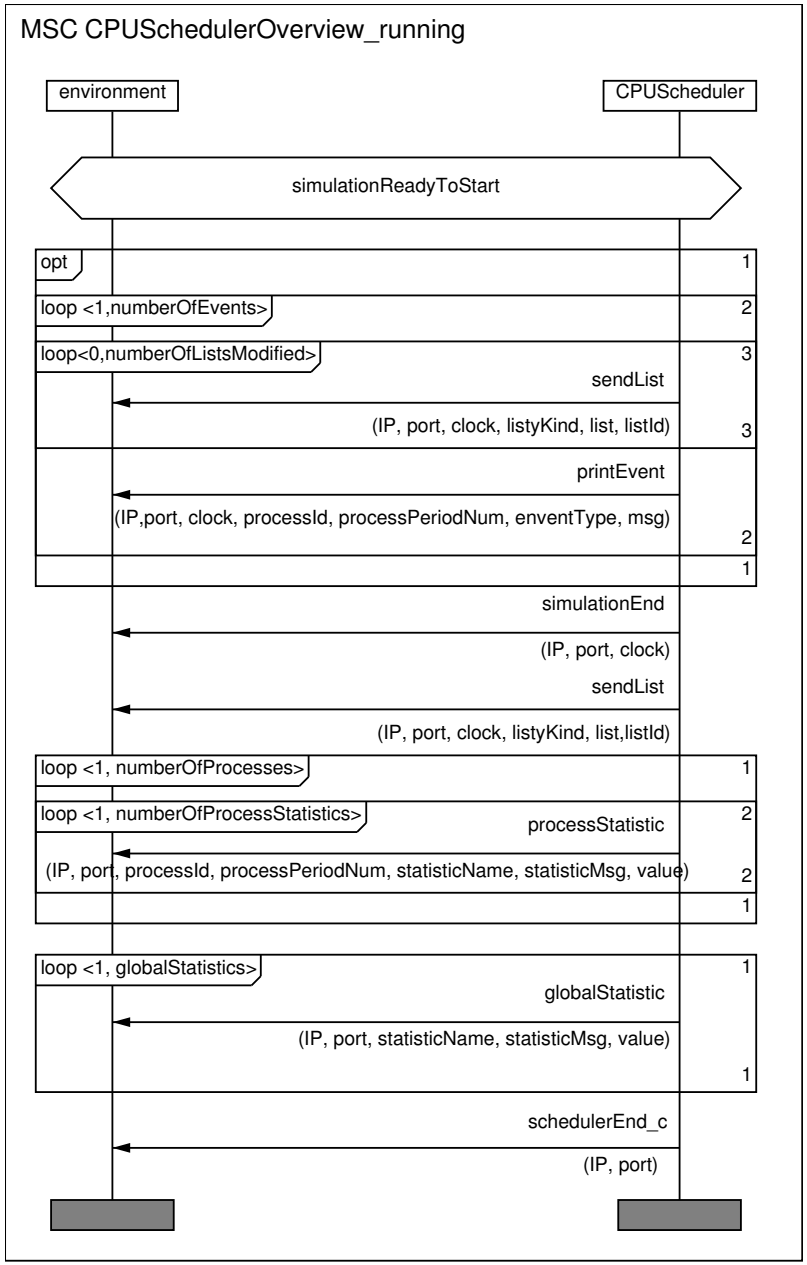
sdlCPUSched: behavior overview

- Signal-based communication between the simulator and its environment
  - interactions are mainly asynchronous (only synchronous during the phase of simulation configuration)
- Input signals:
  - simulation configuration (algorithm simulated, process parameters...)
  - simulation control (start, stop, pause...)
- Output signals:
  - confirmation of simulation configuration
  - notification of simulation events (if verbose mode is selected)
  - values of per-process and global statistics
- Scenarios of interactions between the simulator and its environment modeled by MSCs
  - main scenarios: algorithm configuration, process workload configuration and simulation execution

sdlCPUSched: behavior overview

- Signal-based communication between the simulator and its environment
  - interactions are mainly asynchronous (only synchronous during the phase of simulation configuration)
- Input signals:
  - simulation configuration (algorithm simulated, process parameters...)
  - simulation control (start, stop, pause...)
- Output signals:
  - confirmation of simulation configuration
  - notification of simulation events (if verbose mode is selected)
  - values of per-process and global statistics
- Scenarios of interactions between the simulator and its environment modeled by MSCs
  - main scenarios: algorithm configuration, process workload configuration and simulation execution

sdlCPUSched: example of an scenario



Simulator structure: motivation issues

- To keep independent intra-queue and inter-queue scheduling functions ⇒ two process types:
  - *MasterSchedulerType* for inter-queue scheduling, one instance
  - *SlaveSchedulerType* for intra-queue scheduling, as many instances as ready queues
- To minimize the amount of code not automatically generated from the SDL specification:
  - one simulator program for all the users using the TCP communication module included in the SDL tool
    - programming communication functions with the system environment is avoided
    - no need to use extra coding in C for sending the port number of every instance of the user's simulator program to its GUI (using files, running program parameters, . . .)
    - one control process required to route signals of one user to the corresponding scheduling processes

Simulator structure: motivation issues

- To keep independent intra-queue and inter-queue scheduling functions ⇒ two process types:
  - *MasterSchedulerType* for inter-queue scheduling, one instance
  - *SlaveSchedulerType* for intra-queue scheduling, as many instances as ready queues
- To minimize the amount of code not automatically generated from the SDL specification:
  - one simulator program for all the users using the TCP communication module included in the SDL tool
    - programming communication functions with the system environment is avoided
    - no need to use extra coding in C for sending the port number of every instance of the user's simulator program to its GUI (using files, running program parameters, ...)
    - one control process required to route signals of one user to the corresponding scheduling processes

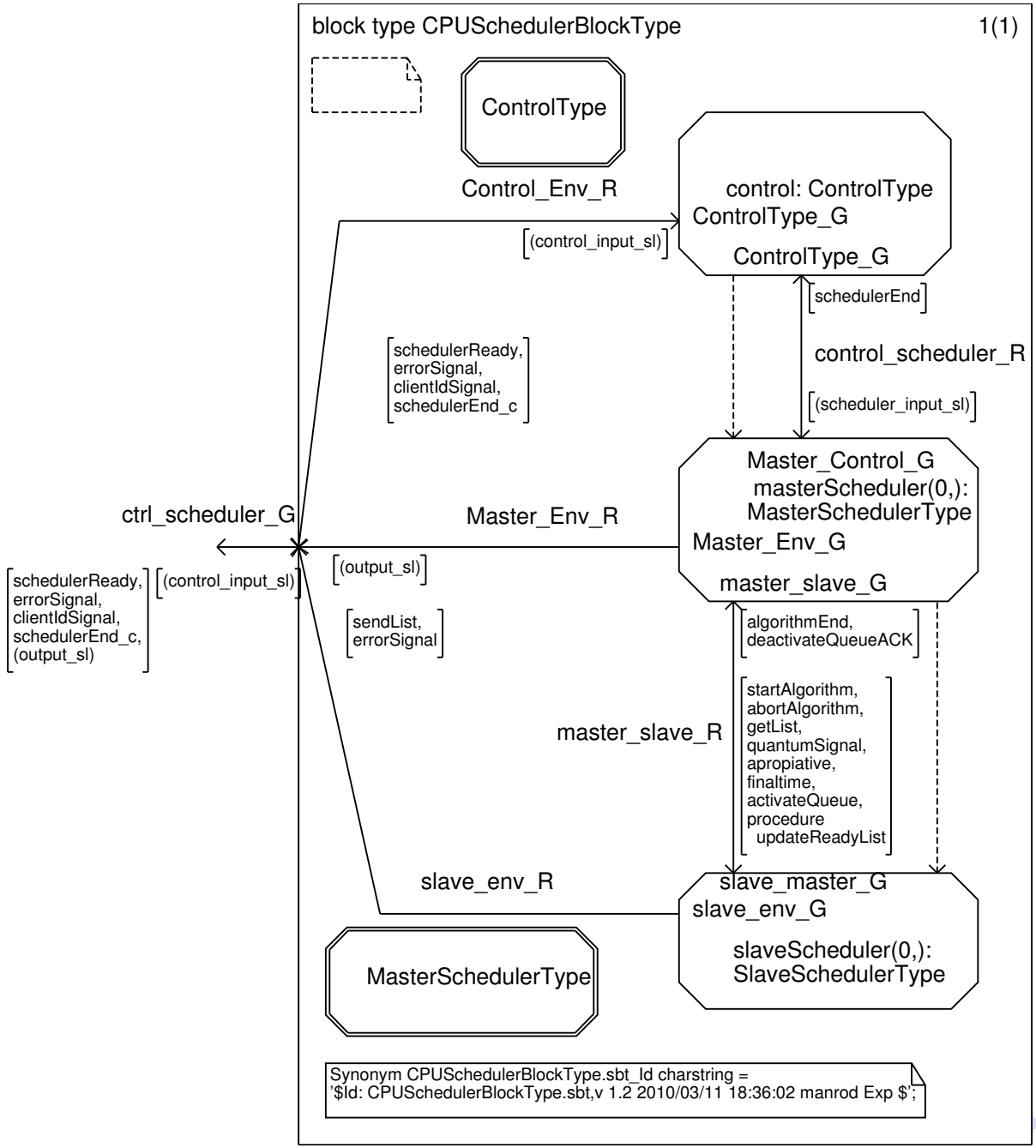
Simulator structure: motivation issues

- To keep independent intra-queue and inter-queue scheduling functions ⇒ two process types:
  - *MasterSchedulerType* for inter-queue scheduling, one instance
  - *SlaveSchedulerType* for intra-queue scheduling, as many instances as ready queues
- To minimize the amount of code not automatically generated from the SDL specification:
  - one simulator program for all the users using the TCP communication module included in the SDL tool
    - programming communication functions with the system environment is avoided
    - no need to use extra coding in C for sending the port number of every instance of the user's simulator program to its GUI (using files, running program parameters, . . .)
    - one control process required to route signals of one user to the corresponding scheduling processes

Simulator structure: motivation issues

- To keep independent intra-queue and inter-queue scheduling functions ⇒ two process types:
  - *MasterSchedulerType* for inter-queue scheduling, one instance
  - *SlaveSchedulerType* for intra-queue scheduling, as many instances as ready queues
- To minimize the amount of code not automatically generated from the SDL specification:
  - one simulator program for all the users using the TCP communication module included in the SDL tool
    - programming communication functions with the system environment is avoided
    - no need to use extra coding in C for sending the port number of every instance of the user's simulator program to its GUI (using files, running program parameters, . . .)
    - one control process required to route signals of one user to the corresponding scheduling processes

Simulator structure: CPUSchedulerBlockType



Process Type functions

- `controlType` process type:
  - creating a *masterScheduler* process for a simulation requested by one user
  - dispatching signals coming from different users to the corresponding *masterScheduler* process
- `masterSchedulerType`:
  - simulation configuration
  - inter-queues algorithm (the maximum priority non-empty ready queue is activated)
  - access control of the ready queues shared variables
- `slaveSchedulerType`:
  - simulation of a simple (one ready queue) scheduling algorithm
    - finding process arrivals (storing every arrived process in the ready queue)
    - selecting a process from the ready queue to be executed (leaves the ready queue)
    - interrupting a process and returning it to the ready queue
    - terminating a process when its CPU burst is completed (storing the process in the finished queue)

Process Type functions

- `controlType` process type:
  - creating a *masterScheduler* process for a simulation requested by one user
  - dispatching signals coming from different users to the corresponding *masterScheduler* process
- `masterSchedulerType`:
  - simulation configuration
  - inter-queues algorithm (the maximum priority non-empty ready queue is activated)
  - access control of the ready queues shared variables
- `slaveSchedulerType`:
  - simulation of a simple (one ready queue) scheduling algorithm
    - finding process arrivals (storing every arrived process in the ready queue)
    - selecting a process from the ready queue to be executed (leaves the ready queue)
    - interrupting a process and returning it to the ready queue
    - terminating a process when its CPU burst is completed (storing the process in the finished queue)

Process Type functions

- `controlType` process type:
  - creating a *masterScheduler* process for a simulation requested by one user
  - dispatching signals coming from different users to the corresponding *masterScheduler* process
- `masterSchedulerType`:
  - simulation configuration
  - inter-queues algorithm (the maximum priority non-empty ready queue is activated)
  - access control of the ready queues shared variables
- `slaveSchedulerType`:
  - simulation of a simple (one ready queue) scheduling algorithm
    - finding process arrivals (storing every arrived process in the ready queue)
    - selecting a process from the ready queue to be executed (leaves the ready queue)
    - interrupting a process and returning it to the ready queue
    - terminating a process when its CPU burst is completed (storing the process in the finished queue)

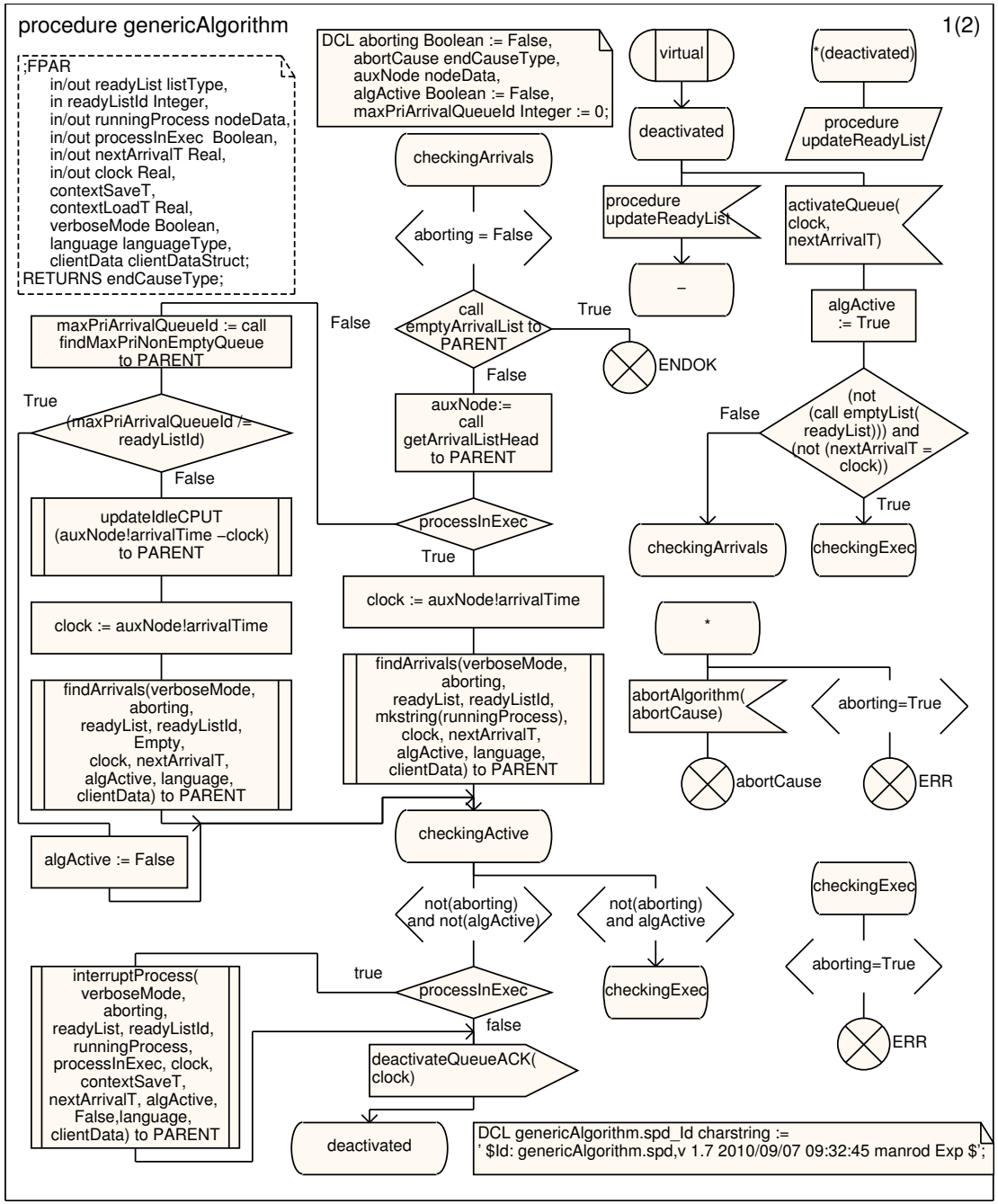
Procedures relating to single-queue algorithms

- Two types of behavior in single queue scheduling algorithms:
  - behavior common to all the algorithms: included in the *genericAlgorithm* procedure
  - algorithm-dependent behavior: included in algorithm specific procedures (one per single-queue algorithm)
- Low level tasks (for example, managing process queues and updating the clock) are specified in auxiliary procedures (*findArrivals*, *executeProcess*, *interruptProcess*, etc.)
  - main behavior of the algorithm is clearly specified without unnecessary low-level details

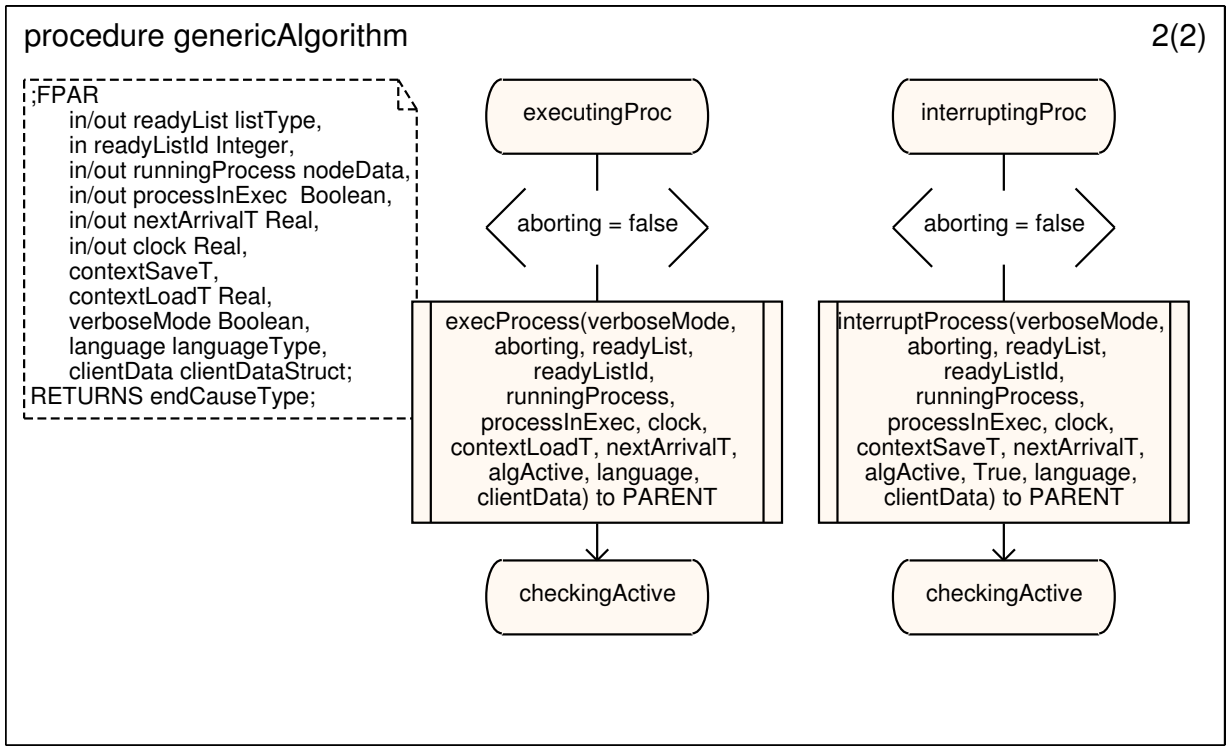
Procedures relating to single-queue algorithms

- Two types of behavior in single queue scheduling algorithms:
  - behavior common to all the algorithms: included in the *genericAlgorithm* procedure
  - algorithm-dependent behavior: included in algorithm specific procedures (one per single-queue algorithm)
- Low level tasks (for example, managing process queues and updating the clock) are specified in auxiliary procedures (*findArrivals*, *executeProcess*, *interruptProcess*, etc.)
  - main behavior of the algorithm is clearly specified without unnecessary low-level details

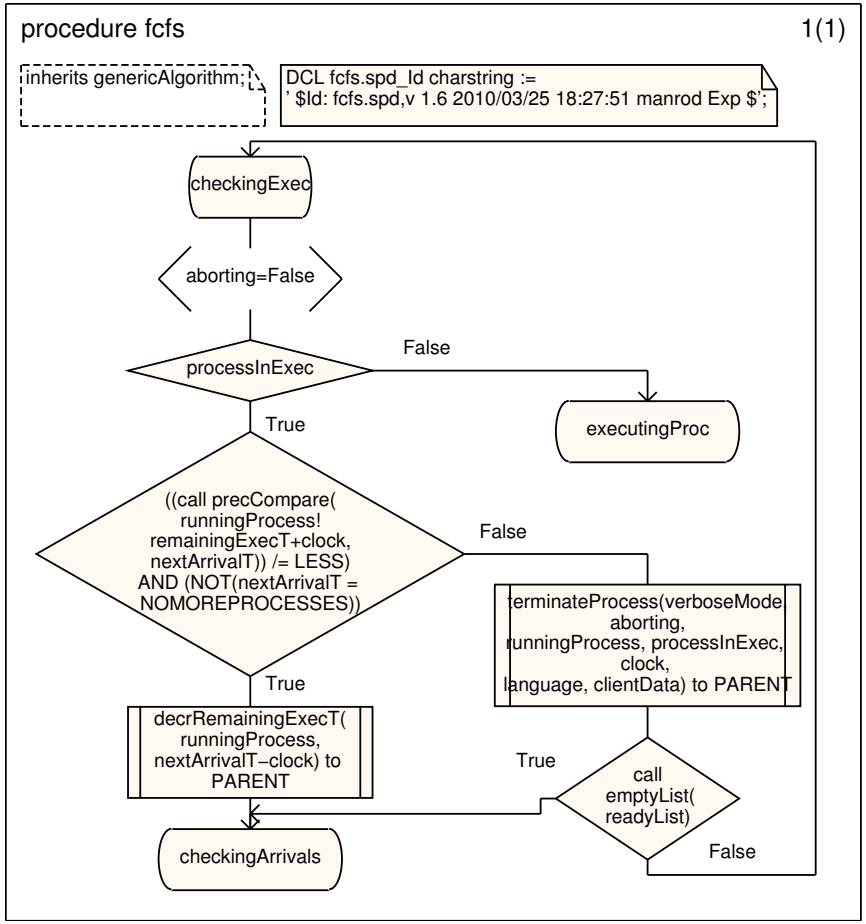
genericAlgorithm procedure specification (I)



genericAlgorithm procedure specification (II)



An example of an specific algorithm procedure



The CPU Scheduler Simulator Graphical User Interface: *tkCPUSched*

- A graphical user interface (GUI) has been developed to simplify the access to the simulator functionalities
  - simulator program generated from the SDL specification is only accessible through a TCP socket
- GUI developed using Tcl/Tk language
  - interpreted language that provides rapid development of cross-platform graphical user interfaces
- GUI module for parsing simulator outgoing signals:
  - format of the signals interchanged with the environment: Telelogic ASCII encoding (TCP communications module)
    - example: {1}{'schedulerEnd_c'}{{'127.0.0.1',50205}}
  - parsing module based on an automatically generated parser and lexical analyzer
    - parser and lexical analyzer based on a grammar specific to the ASCII encoding: changes in the signal encoding $\Rightarrow$ modifying the grammar and re-generating the parser and lexical analyzer
    - GUI can be adapted to support a simulator generated by different SDL case tools

The CPU Scheduler Simulator Graphical User Interface: *tkCPUSched*

- A graphical user interface (GUI) has been developed to simplify the access to the simulator functionalities
  - simulator program generated from the SDL specification is only accessible through a TCP socket
- GUI developed using Tcl/Tk language
  - interpreted language that provides rapid development of cross-platform graphical user interfaces
- GUI module for parsing simulator outgoing signals:
  - format of the signals interchanged with the environment: Telelogic ASCII encoding (TCP communications module)
    - example: {1}{'schedulerEnd_c'}{{'127.0.0.1',50205}}
  - parsing module based on an automatically generated parser and lexical analyzer
    - parser and lexical analyzer based on a grammar specific to the ASCII encoding: changes in the signal encoding ⇒ modifying the grammar and re-generating the parser and lexical analyzer
    - GUI can be adapted to support a simulator generated by different SDL case tools

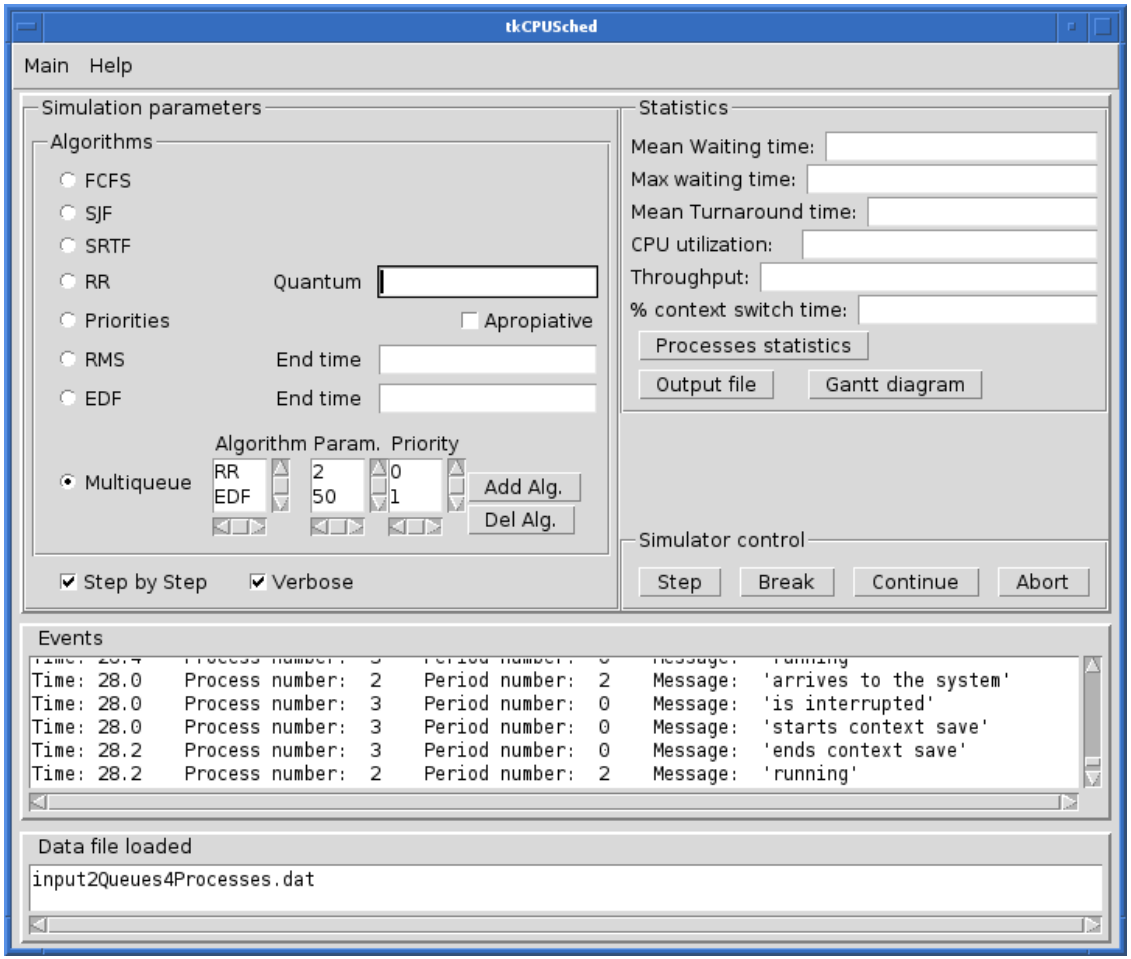
The CPU Scheduler Simulator Graphical User Interface: *tkCPUSched*

- A graphical user interface (GUI) has been developed to simplify the access to the simulator functionalities
  - simulator program generated from the SDL specification is only accessible through a TCP socket
- GUI developed using Tcl/Tk language
  - interpreted language that provides rapid development of cross-platform graphical user interfaces
- GUI module for parsing simulator outgoing signals:
  - format of the signals interchanged with the environment: Telelogic ASCII encoding (TCP communications module)
    - example: {1}{'schedulerEnd_c'}{{'127.0.0.1',50205}}
  - parsing module based on an automatically generated parser and lexical analyzer
    - parser and lexical analyzer based on a grammar specific to the ASCII encoding: changes in the signal encoding $\Rightarrow$ modifying the grammar and re-generating the parser and lexical analyzer
    - GUI can be adapted to support a simulator generated by different SDL case tools

The CPU Scheduler Simulator Graphical User Interface: *tkCPUSched*

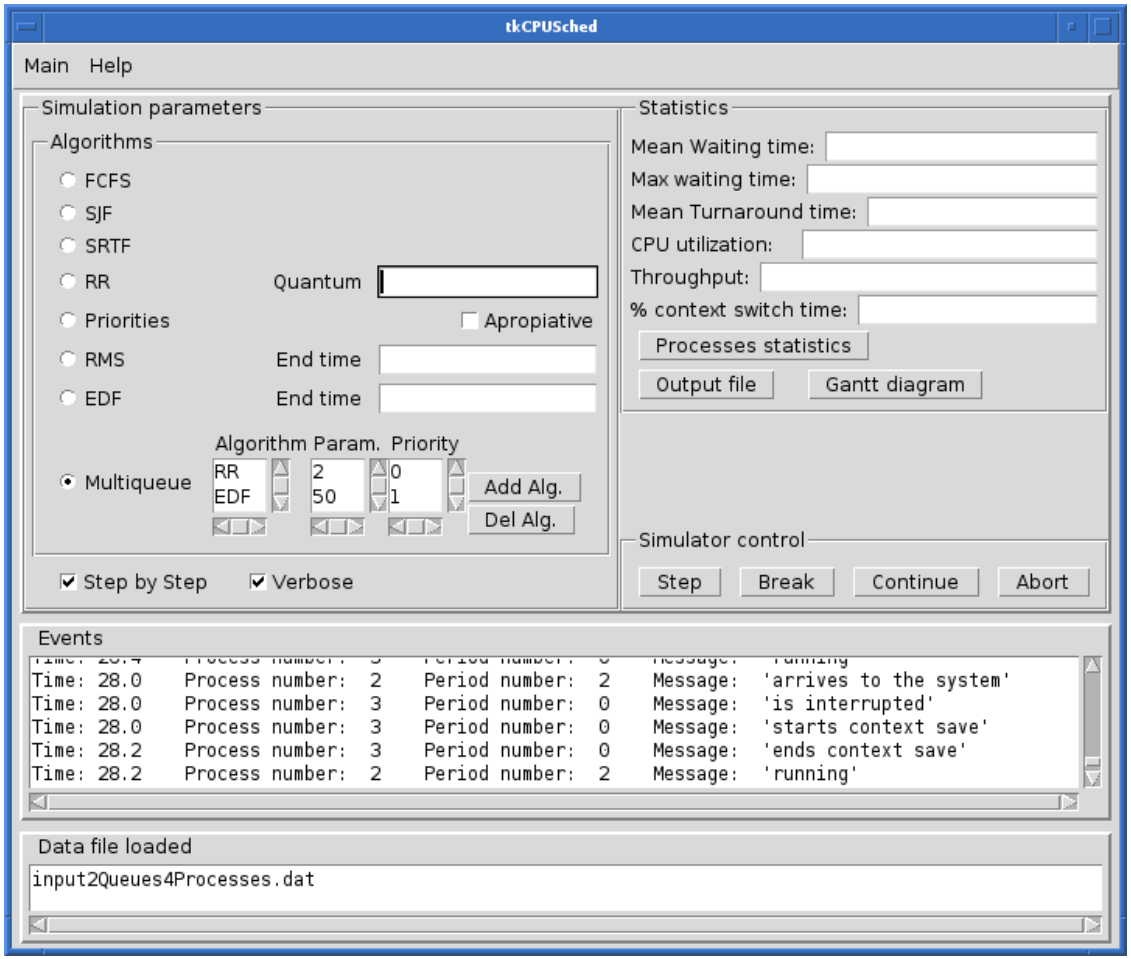
- A graphical user interface (GUI) has been developed to simplify the access to the simulator functionalities
  - simulator program generated from the SDL specification is only accessible through a TCP socket
- GUI developed using Tcl/Tk language
  - interpreted language that provides rapid development of cross-platform graphical user interfaces
- GUI module for parsing simulator outgoing signals:
  - format of the signals interchanged with the environment: Telelogic ASCII encoding (TCP communications module)
    - example: {1}{'schedulerEnd_c'}{{'127.0.0.1',50205}}
  - parsing module based on an automatically generated parser and lexical analyzer
    - parser and lexical analyzer based on a grammar specific to the ASCII encoding: changes in the signal encoding $\Rightarrow$ modifying the grammar and re-generating the parser and lexical analyzer
    - GUI can be adapted to support a simulator generated by different SDL case tools

tkCPUSched: main window



- Additional features:
  - queue status shown during simulation (and process statistic values)
  - plotting of Gantt charts during simulation

tkCPUSched: main window



- Additional features:
  - queue status shown during simulation (and process statistic values)
  - plotting of Gantt charts during simulation

Conclusions (I)

- Simulator features:
  - simulator capable of simulating the behavior of
    - one-queue scheduling algorithms (appropriate for one class of processes workloads)
    - multilevel queue algorithms (appropriate for workloads consisting of processes with different scheduling requirements)
  - simulation results (per-process and global statistics) and simulation events shown in a human-readable format using a graphical user interface
    - results and events can also be saved in files
  - graphical user interface developed with the Tcl/Tk language: rapid development and cross-platform support
  - parsing of signals outgoing simulator program based on a grammar definition of their encoding method
    - change of the encoding method only implies minimum changes in the graphical user interface (adapting the grammar and re-generating the parser and lexical analyzer)
    - GUI customizable to interact with simulators generated by different SDL case tools

Conclusions (I)

- Simulator features:
  - simulator capable of simulating the behavior of
    - one-queue scheduling algorithms (appropriate for one class of processes workloads)
    - multilevel queue algorithms (appropriate for workloads consisting of processes with different scheduling requirements)
  - simulation results (per-process and global statistics) and simulation events shown in a human-readable format using a graphical user interface
    - results and events can also be saved in files
  - graphical user interface developed with the Tcl/Tk language: rapid development and cross-platform support
  - parsing of signals outgoing simulator program based on a grammar definition of their encoding method
    - change of the encoding method only implies minimum changes in the graphical user interface (adapting the grammar and re-generating the parser and lexical analyzer)
    - GUI customizable to interact with simulators generated by different SDL case tools

Conclusions (I)

- Simulator features:
  - simulator capable of simulating the behavior of
    - one-queue scheduling algorithms (appropriate for one class of processes workloads)
    - multilevel queue algorithms (appropriate for workloads consisting of processes with different scheduling requirements)
  - simulation results (per-process and global statistics) and simulation events shown in a human-readable format using a graphical user interface
    - results and events can also be saved in files
  - graphical user interface developed with the Tcl/Tk language: rapid development and cross-platform support
  - parsing of signals outgoing simulator program based on a grammar definition of their encoding method
    - change of the encoding method only implies minimum changes in the graphical user interface (adapting the grammar and re-generating the parser and lexical analyzer)
    - GUI customizable to interact with simulators generated by different SDL case tools

Conclusions (I)

- Simulator features:
  - simulator capable of simulating the behavior of
    - one-queue scheduling algorithms (appropriate for one class of processes workloads)
    - multilevel queue algorithms (appropriate for workloads consisting of processes with different scheduling requirements)
  - simulation results (per-process and global statistics) and simulation events shown in a human-readable format using a graphical user interface
    - results and events can also be saved in files
  - graphical user interface developed with the Tcl/Tk language: rapid development and cross-platform support
  - parsing of signals outgoing simulator program based on a grammar definition of their encoding method
    - change of the encoding method only implies minimum changes in the graphical user interface (adapting the grammar and re-generating the parser and lexical analyzer)
    - GUI customizable to interact with simulators generated by different SDL case tools

Conclusions (II)

- Experiences in the simulator development with SDL:
  - behavior specification of a scheduling algorithm is easier in SDL than in a programming language
    - a flow chart describing high-level algorithm behavior can be used to obtain SDL specifications
    - SDL graphical syntax is more expressive than code for the description of an algorithm behavior
  - development and testing of concurrent applications easier in SDL
    - signal or procedure based communication in SDL versus external communications libraries, ad-hoc methods or shared variables in a programming language
    - communications between processes explicitly shown (messages that can be exchanged, the sources and destinations allowed . . .)
    - powerful testing tools, including tracing of messages exchanged among processes (for example, MSC trace)

Conclusions (II)

- Experiences in the simulator development with SDL:
  - behavior specification of a scheduling algorithm is easier in SDL than in a programming language
    - a flow chart describing high-level algorithm behavior can be used to obtain SDL specifications
    - SDL graphical syntax is more expressive than code for the description of an algorithm behavior
  - development and testing of concurrent applications easier in SDL
    - signal or procedure based communication in SDL versus external communications libraries, ad-hoc methods or shared variables in a programming language
    - communications between processes explicitly shown (messages that can be exchanged, the sources and destinations allowed . . .)
    - powerful testing tools, including tracing of messages exchanged among processes (for example, MSC trace)

Further work

- Improvements for the current version of the simulator:
  - adding more complex algorithms, like multilevel feedback queue scheduling
  - multi-processor / multi-core algorithms
  - support of processes with several CPU and I/O bursts

Further work

- Improvements for the current version of the simulator:
  - adding more complex algorithms, like multilevel feedback queue scheduling
  - multi-processor / multi-core algorithms
  - support of processes with several CPU and I/O bursts

Further work

- Improvements for the current version of the simulator:
  - adding more complex algorithms, like multilevel feedback queue scheduling
  - multi-processor / multi-core algorithms
  - support of processes with several CPU and I/O bursts

Questions?