МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

Методичні вказівки і завдання

до виконання лабораторних робіт
з курсу «Програмування розподілених систем обробки даних»
(англійською мовою)

2011

МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
ДВНЗ «ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК І ТЕХНОЛОГІЙ
КАФЕДРА ПРИКЛАДНОЇ МАТЕМАТИКИ ТА ІНФОРМАТИКИ

Методичні вказівки і завдання

до виконання лабораторних робіт
з курсу «Програмування розподілених систем
обробки даних» (англійською мовою)
(для студентів спеціальності 6.080403 „Програмне забезпечення автоматизованих систем")

Укладачі:
А. М. Гізатулін, к.е.н., доц.
Ю. В. Попов, к.т.н., доц.

Розглянуто на засіданні кафедри
прикладної математики і інформатики
Протокол № 2 від __.__.2011

Затверджено на засіданні
Навчально-видавничої ради ДонНТУ
Протокол № ___ від ___.___.2011

2011

УДК 004.052

Методичні вказівки і завдання до виконання лабораторних робіт з курсу «Програмування розподілених систем обробки даних» (англійською мовою) / для студентів спеціальності 6.080403 Програмне забезпечення автоматизованих систем / Укладачі Гізатулін А.М., Попов Ю.В.– Донецьк: ДонНТУ, 2011. – 26 с.

Методичні вказівки і завдання до виконання лабораторних робіт з курсу «Програмування розподілених систем обробки даних» підготовлені на основі типової програми курсу і направлені на вивчення методології, методики та інструментарію програмування розподілених систем обробки даних, їх аналізу та використання. Метою лабораторного практикуму є формування системи практичних знань у галузі дослідження та програмування розподілених систем обробки даних.

Укладачі: А.М. Гізатулін, к.е.н., доц.
Ю.В. Попов, к.т.н., доц.

TABLE OF CONTENTS

LAB 1
IMPLEMENTING A VIRTUAL CLOCK

Lab objective:
To learn ways to control the virtual time in logic simulation system.

The theoretical background
The simulation is processed in a discrete virtual time VT. One tick of VT corresponds to one cycle of a physical device. The simulation begins at moment of time VT=0 and finishes when it reaches an end moment of time (that is specified by a user), or when there are no more events in the event list.

The following should be considered when implementing virtual clocks:
- virtual clock shows virtual time;
- The virtual time is a positive integer;
- At the beginning moment VT=0;
- The time is advanced (moved forward) when there are no more events in the event list scheduled for a current moment of virtual time;
- In some simulation algorithms the time is allowed to be put back;
- The virtual clock should be thread safe to allow multithreaded environment.

Virtual clock fields:
- Current Virtual Time (an integer)
- Whether the time is allowed to put back (a boolean)

Virtual clock methods:
- Set a current time (the method should check if a time is a positive integer, and if it can be put back, in accordance to the flag)
- Get the current time
- Reset the current time to zero
- Set the allow-put-back flag (to true or to false)
- Get the allow-put-back

The task:
1. Create a virtual clock class.
2. Create Unit-tests for all method of the virtual clock.
3. Create a main class that will call all unit-tests.

The report contents
1. The front page.
2. The laboratory work theme.
3. The task.
4. The source code of the virtual clock class.
5. The source code of all unit-tests.
6. The source code of a main class.
7. Conclusions.

The quiz
1. What methods should the virtual clock contain?
2. What fields does the virtual clock have?
3. What is the unit of a virtual time?
4. Can the virtual time be put back?

LAB №2
IMPLEMENTING AN EVENT LIST

Lab objective:
To learn ways to control the event flow in a logic simulation system.

The theoretical background

Event list (EVL) is a container for events at a given simulation processor. At the beginning of simulation EVL is filled with stimulus EVL sorts events by their timestamp (within a given simulation processor). Events can be selected in timestamp order only. Events can be added to EVL in a random order. For each event, the EVL changes twice: 1) when an event is added, and 2) when an event is removed. At the beginning of a simulation cycle, all events scheduled for current moment of VT are selected, one by one. After all these events are selected and applied, new output signals are computed for all elements affected. These new signals are turned into events and added to the EVL. After that the simulation cycle repeats. In conservative and combined synchronization protocols, EVL is able to wait until an appropriate event occurs. Event list should use thread synchronization tools.

The event list fields:

- The list. This is an array of structures. Every structure contains three fields:
  - Virtual time (unsigned integer)
  - Node index (index in an array of nodes, unsigned integer)
  - New signal value
- Simulation endtime
- LVTH (contains +? in optimistic models)

The event list methods:

- A method to select a VT, a node index and a signal value for the next event. Returns data for an event with the lowest timestamp. This is allowed to be three independent methods.
- A method to add an event. New events are added in a way that allows you to keep the list sorted by timestamp.
- A method to clear the EVL. This method is called at the beginning of a simulation to clear the EVL. The method should reset all object fields, including simulation endtime and LVTH.
- A method to remove the next event. Removes the next event (an event with the lowest timestamp) from the event list.
- A method to get the total number of events. This method return the total number of events in the event list.
- A method to get/set simulation endtime. When you try to add an event beyond the simulation endtime, this event is ignored. No error exceptions are raised in this case.
- A method to load the stimulus. This method reads a file, parses it and fills the event list with the events read.
- A method to get/set the LVTH. LVTH equals +infinity by default. This corresponds the optimistic simulation model. This value should also be used in a sequential model. In a conservative model LVTH contains the current local virtual time horizon.
- A method to wait for an appropriate event to occur. If a VT of the next event is less then LVTH, this method does nothing. In other case, this methods waits until such an event occurs. There are two cases when such events can occur: 1) such an event can arrive from a remote SimProc; 2) LVTH can increase. Java contains special tools that allows you to lock the current thread until an event occurs (for example, a message is arrived from a remote SimProc).

All event list methods must use thread synchronization tools! A single (a main thread) thread received event messages from remote SimProcs, and an another thread (a computational thread) – processes the events. Computational thread is locked, if there are not events to process (but such events can occur).

The event list is required to be implemented universally. The EVL is not allowed to be dependent on a synchronization protocol or a simulation algorithm used (sequential, conservative, optimistic, etc.).

Special protected methods are allowed for testing purposes. These methods can, for example, return values of private fields. The test class should inherit the main class (to allow usage of that protected methods).

The Event List is changed very often in a course of a simulation process (twice for every event). The most resource-intensive operations are:

- to find a place in the list of events, where a new event should be inserted;
- to insert the event into the list;
- to remove an event out from the list.

The task:

1. Create an event list class
2. Create unit-tests for all methods of the event list
3. Create a main class that will call all unit-tests

The report contents:

1. The front page.
2. The laboratory work theme.
3. The task.
4. The source code of the event list class.
5. The source code of all unit-tests.
6. The source code of a main class.
7. Conclusions.

The quiz

1. What methods should the event list implement?
2. What fields does the event list have?
3. What is an event?
4. How are events ordered inside the event list?
5. When can the event list become unlocked?

LAB №3
STRUCTURAL AND FUNCTIONAL SCHEME MODEL

Lab objective:
To learn ways to represent the structure and functions of digital circuits in computer applications.

The theoretical background

The SFSM purposes:

- Storing a structure of the simulated circuit: how are elements interconnected;
- Storing states of the simulated circuit: signal values in all circuit nodes;
- Representing functions of the simulated circuit: SFSM should be able to compute output signals in dependence on input signal values;
- Maintaining a list of changed elemets. This allows you to compute output signals at output nodes on changed elements only;
- Generating new events in correspondence with new signals at element output nodes;
- Representing external nodes. This should allow you to pass messages between different SimProcs. External nodes have names;
- Generating of external events (in the case of lazy cancellation);
- Loading of a circuit before the simulation starts;
- Storing current circuit state in a state stack.

The SFSM fields:

- An array of elements. Information about each element:
  - element function (AND, OR, NOT, etc.)
  - element name (for debug purposes)
  - a list of indexes in the global nodes array that correspond to element input nodes
  - a list of indexes in the global nodes array that correspond to element output nodes
- An array of nodes. Information about each node:
  - A list of elements influenced by this node. This is used to define a list of changed elements when an event occurs at this node
  - node name (for debug purposes)
  - signal value (true, false, undefined, upper front, lower front, etc.)
- An array of external input nodes. Information about each external node:
  - node name. This name will be written to the event messages sent between remote processors. This name is used to find an external input that corresponds to an external output node. Corresponding external nodes must have the same name.
  - a local node index (an index in the array of nodes).
- An array of external output nodes:
  - node name. Corresponds to a name of an external input node;
  - a local node index (an index in the array of nodes)
  - receiver processor name
- A list of changed elements. When an event is processed, the elements affected by the event are added to this list. No elements allowed to be added more then once. Every next attempt to add an element that already exists in this list should be ignored.

The SFSM properties:

- Get a number of elements
- Get a number of nodes
- Get a number of input nodes at a given element (be element index)
- Get a number of output nodes at a given element (be element index)
- Get an element name by the index of the element
- Get the total number of external input nodes
- Get the total number of external output nodes
- Get a name of an external input node by it's index (in the array of external input nodes)
- Get a name of an external output node by it's index (in the array of external output nodes)
- Get a name of a node by the node index
- Get a node signal by the node index
- Get an element function by the element index
- Get a flag if a circuit is successfully loaded

The SFSM methods:

- Reset all SFSM fields to zero state;
- Load a circuit from an external file;
- Get an external output node name and a remote processor name connected to that output by element index and index of an output node of that element:
  - this method allows you to learn where to send a message about an external event;
  - you are allowed to store this information cached and prepared at a moment of circuit loading (for optimization purposes);
- Get element output values. using and element index and the element input node index, this method defines a global node index and returns a signal value at this global node;
- Get a global node index by the external input node name
- Set a signal value at a specified node. This additionally adds affected elements to a list of changed elements. This possibility is defined with an additional third argument. For example, while a rollback in an optimistic protocol, this function is not needed.
- A method to reset a list of changed elements
- A method to compute output values at all changed elements. It is important to take into account:
  - output nodes of a single element may not be updated more then once. This is usually true if a list of changed elements do not contain duplicates;
  - This method received LVT as an argument and generates new events in dependence on element delays;
  - new event should not be generated if an output signal is not changed;
  - when this method completes, a list of changed elements should be set to empty.

It is comfortable to have a base class Element with an abstract method to compute output signals, and a set of subclasses for every element function (AND, OR, NOT etc.). There is a sense to make a list of changed elements sorted - to prevent adding duplicates more quickly. Feel free to add as many methods as you may need. For example, a separate method to add an element to a list of changed elements would be useful. In sequential and conservative simulation models the SFSM is used by the computational thread only. Thread synchronization tools must be provided. For example, in an optimistic SE rollbacks are often executed by the main thread (node signals values are changed while a rollback). You should not access your fields from anywhere (including any generic method of the SFSM class) besides specialized methods.

The task

1. Create a structural-and-functional-scheme-model class
2. Create Unit-tests for all methods of the created class.
3. Create a main class that will call all unit-tests.

The report contents:

1. The front page.
2. The laboratory work theme.
3. The task.
4. The source code of the class of the structural and functional scheme model.
5. The source code of all unit-tests.
6. The source code of a main class.
7. Conclusions.

The quiz:

1. What methods should SFSM implement?
2. What fields does SFSM have?
3. What properties does SFSM have?
4. What is the purposes of the SFSM?
5. What will happen if at a given point of virtual time, there will appear an element that will update its output nodes values twice?

LAB №4
A SEQUENTIAL SIMULATION ENGINE

Lab objective:
To study algorithms for sequential simulation of digital devices.

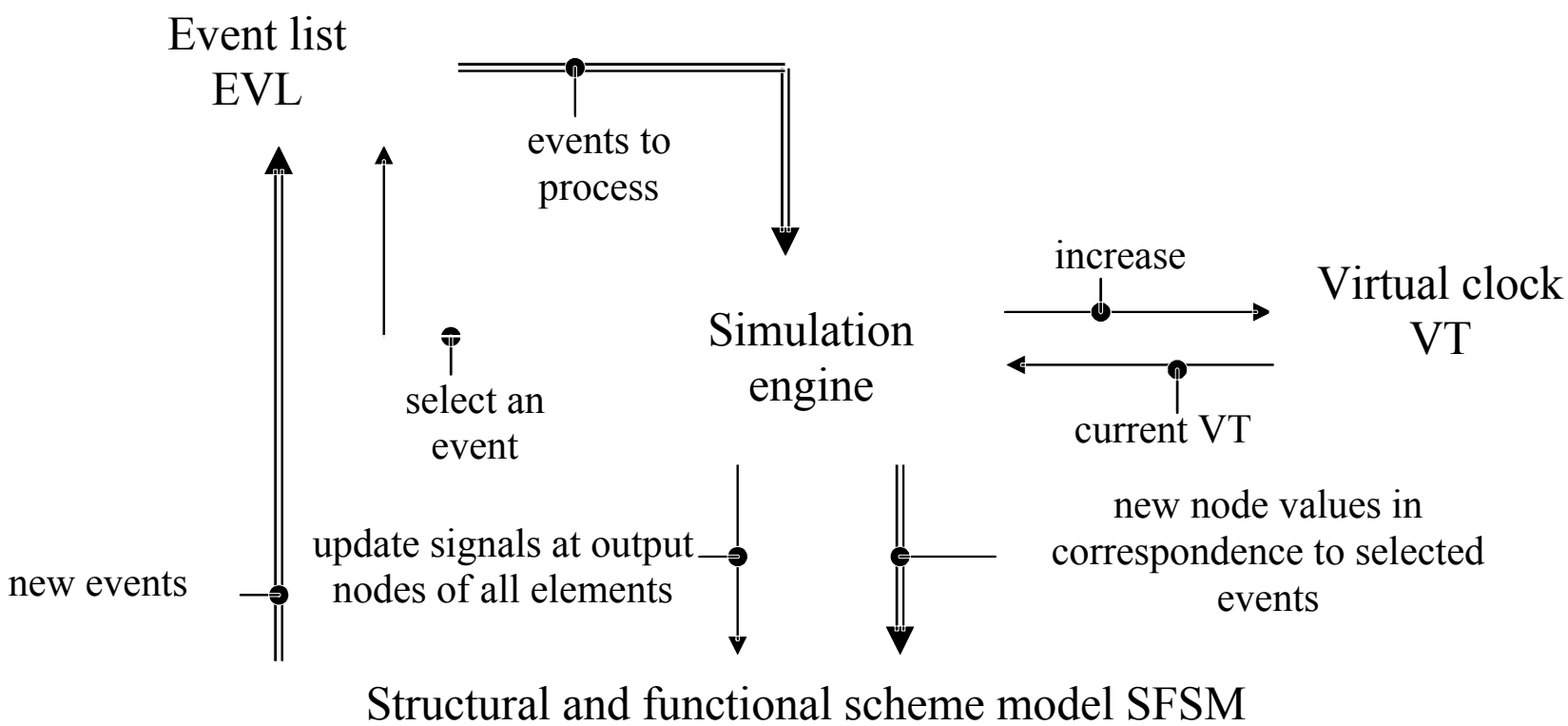
The theoretical background

The sequential simulation processor processes all events in the system sequentially one by one. Simultaneous processing of two or more events is not allowed in the sequential processor. A sequential simulation processor architecture is presented at pic. 4.1.

A sequential simulation processor contains:

- An event list (EVL). This list contain all the events in the system. The events in this list are sorted in order of their appearance in virtual time;
- Virtual clock (VC) shows the virtual time VT;
- structural and functional scheme model SFSM. Contains a structure of a simulated circuit, description of functions of all elements of the circuit, and signal values at all circuit nodes;
- The simulation engine SE manages all other objects of the simulation processor.

The SE selects events schedule for a current moment of VT from the EVL (pic. 4.2), sends a data to set new signals at circuit nodes affected by the selected events. When there are no more events scheduled for a current moment of virtual time, the SE tells the SFSM to update signals at output nodes of all affected elements in accordance to new input signals. New signals will cause new events to be scheduled by sending them to EVL.



Picture 4.1 – Architecture of a sequential simulation processor

```
while LEVL.has_events
begin
  LVT:=LEVL.time_of_next_event;
  while LEVL.time_of_next_event=LVT
  begin
    node:=LEVL.node_of_next_event;
    signal:=LEVL.signal_of_next_event;
    LEVL.remove_next_event();
    if SFSM.node_signal(node)=signal then
      //not changed
      continue;
    SFSM.set_node_signal(node,signal);
  end;
  SFSM.update_output_signals_of_affected_elements();
end;
```

Picture 4.2 – A sequential simulation engine algorithm sketch

The timestamp for new events is defined as the current virtual time plus a positive delay at a corresponding element. The time of a new event is always greater then the current virtual time. When the update process is completed, the SE puts the virtual clock forward to the time of the next event in the system.

The task:

1. Implement a class for a sequential simulation engine.
2. Create unit-tests for all methods of a sequential SE.
3. Create a class that will invoke all unit-tests
4. Create an application for sequential simulation of digital devices.

The report contents:

1. The front page.
2. The laboratory work theme.
3. The task.
4. The source code of all classes created.
5. The source code of all unit-tests.
6. The source code of a main class.
7. The source circuit, stimulus and the simulation results.
8. Conclusions.

The quiz:

1. What objects does the sequential simulation processor contain?
2. Where do events come to event list from?
3. Where does the information about new node signals come to SFSM from?
4. How does SE know when to start updating element output signals?
5. How many times will the outer cycle of the SE be repeated?

LAB №5
CONSERVATIVE COMMUNICATION INTERFACE

Lab objective:
The goal is to study synchronization algorithms for logical processes in distributed data processing systems.

The theoretical background

Conservative communication interface is intended for:
- controlling over the global order of event processing;
- evaluating of global states;
- managing the processes in the deadlock state (detection, recovery and avoidance of deadlocks);
- receiving and sending event messages;
- locking and unlocking of SE;
- computation of LVTH.

Conservative communication interface fields:
- Input buffers - is an array containing a local node index and a time of last event arrived over a given channel (channel clock) for each input channel.
- Output buffers - is an array containing a value and a timestamp of the last event sent over this channel, for each output channel.
- Local virtual time horizon is an integer number. This number is updated automatically when a next event message arrived.
- SimProc color (red or white) is used to detect a deadlock using a marker

Conservative communication interface methods:
- Receive a message. This is an universal method to perform a primary analysis of a message received and to invoke a specialized method in accordance to the type of the message. This method has two arguments: a remote processor name and a string containing a data block.
- Receive an event message. This method updates an input buffer in correspondence to this event. Evaluates a new value of LVTH. Inserts the event into the LEVL. Registers activity (updates the color of the SimProc). This method has two arguments: a remote processor name, and a data block string.
- Receive a marker message. A main marker processing algorithm is implemented in this method. This method has two arguments: a remote processor name, and a data block string.
- Receive a message to advance the LVTH. This method advances the LVTH irrespectively to LVT value. This is needed to recover a system from a deadlock. This unlocks the LEVL. This method has two arguments: a remote processor name, and a data block string.
- Send a message. This is a universal method to send an abstract message. This method is called from specialized methods to send a message over a network, and should not be called directly otherwise. This method has two arguments: a remote processor name and a data block string.
- Send an event message. This method updates a corresponding output buffer, checks whether a signal is changed, registers activity (updates the SimProc color) and creates a string with a formatted ready to send message. This method has 3 arguments: a node name, a timestamp and a signal value. A remote processor name is retrieved from a node name.
- Send a marker message. This method formats a marker message and passes it to the universal method to send messages. This method has three arguments, each argument corresponds to a field in a marker. A remote processor name is retrieved from a global order of processors in a marker circle. This global order is built using a communication system.
- Send an advance LVTH message. Called from the marker processing algorithm, when a deadlock is detected. This method has two arguments: a remote processor name and an LVTH.
- Complete processing of events at a current moment of virtual time. This method is called from the SE when all events scheduled for a current moment of virtual time are processed. This method can be used to send null-messages. This method has one argument – current local virtual time.
- Reset. Resets values of all fields of the object to the beginning state.
- Get LVTH. Returns the current value of the LVTH.
- Register activity. Sets the SimProc color to white.

Messages sent over the network:
- An event message. Fields: a global name of the node, a signal value, and a timestamp.
- A marker message. Fields: a time of the next event in the system, a processor where this event should occur and a total number of red processors visited by the marker.

- Advance LVTH

A message is a data block. A data block is an arbitrary string. That string may contain text or binary data. A method to send a message formats a data string, and a method to receive a message analyzes the string received.

Example of text blocks for different messages:
An event message: event|n1@p1|1|129
“event” – name of the message type
This message means that an event should be scheduled at moment of virtual time 129 at a node with global name “n1@p1”. A new signal value equals 1.

A marker message: marker|150|p3|2
“marker” – name of the message type
This message means that the next event in the system will occur at processor p3 at moment of virtual time 150, and there were 2 red processors visited by the marker.

An advance LVTH message: lvth|150
«lvth» - name of the message type
This message means that a receiver processor should advance it’s LVTH to 150, regardless of LVT value.

The task:
1. Implement a class for conservative communication interface.
2. Create unit-tests for all methods of conservative communication interface.
3. Create a main class to call all unit-tests.

The report contents:
1. The front page.
2. The laboratory work theme.
3. The task.
4. The source code of all classes created in the work.
5. The source code of all unit-tests.
6. The source code of a main class.
7. Conclusions.

The quiz:
1. What methods does the conservative CI implement?
2. What fields does the conservative CI have?
3. What the conservative CI intended for?
4. What are the differences between the conservative and optimistic CIs?
5. How can CI learn about events at remote processors?

LAB №6
COMMUNICATION SYSTEM

Lab objective:
The goal of the work is to study logical processes interaction methods in distributed data processing systems.

The theoretical background

The communication system (CS) is intended to transmit data blocks between processors, to set communication sessions and to identify processors by names. The communication system is an interface between the communication interface and communication tools provided by Java.

The communication system is a universal unit for conservative, optimistic and combined synchronization protocols.

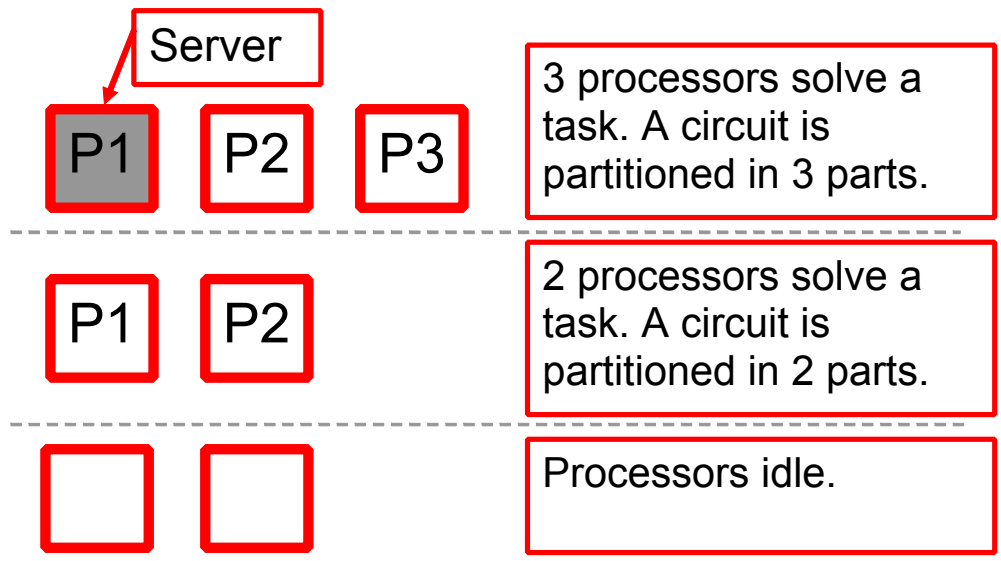
The communication system fields:

- A list of all remote SimProcs in the system. Information about each SimProc: a name (an unique string), IP-address, a number of the port, and a client communication socket.
- Local SimProc name
- A server communication socket

The communication system methods:

- Send a message. This method has two arguments: a remote SimProc name, and a data block.
- Receive a message. This method calls a method in the CI to receive a message and passes a processor name and a data block there.
- Initialize the server socket. This method is called only once at program startup. The number of the port is specified in the program options.
- Initialize client sockets. This method loads a list of all remote SimProcs from options and creates client communication socket for each of them. The connection to old SimProcs is closed when this method is invoked.
- A method to reset all fields to the beginning state: a list of SimProcs is empty, a local SimProc name is not specified.
- Get a list of the next SimProc in the marker circle.

A real simulation system may require an ability to simulate several independent simulation tasks simultaneously. In this case, a special central unit should be provided to manage these tasks (pic. 6.1).



Picture 6.1 – Several simulation tasks are performed in parallel. Processors solving different tasks may have identical names.

The task:

1. Create a task for a communication system.
2. Create unit-tests for all methods of the communication system
3. Create a main class to call all unit-tests.

The report contents:

1. The front page.
2. The laboratory work theme.
3. The task.
4. The source code of all classes created in the work.
5. The source code of all unit-tests.

6. The source code of a main class.
7. Conclusions.

The quiz:

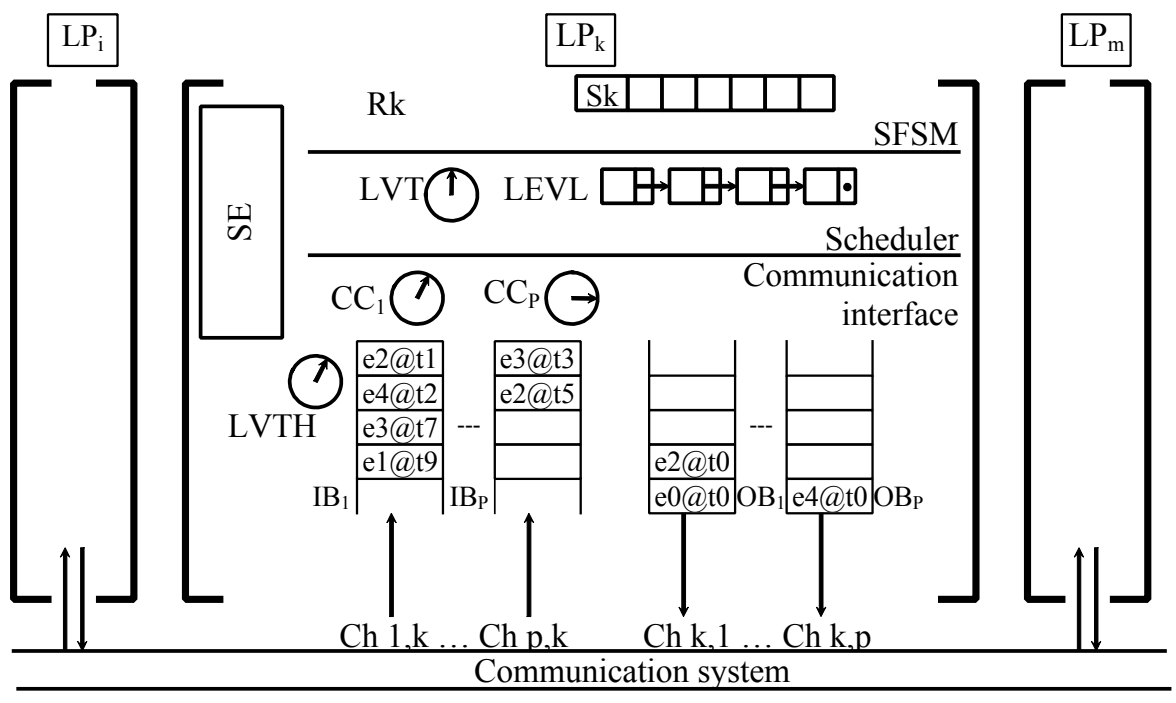
1. What methods does the CS implement?
2. What fields does the CS have?
3. What is the purposes of the CS?
4. How does the CS define the marker circle?
5. How is it possible to reduce a total number of open connections in the CS?

LAB №7
CONSERVATIVE SIMULATION ENGINE

Lab objective:
The goal of the work is to study a conservative algorithm for distributed simulation of digital devices.

The theoretical background

A conservative communication interface contains an Input Buffer IBi and a Channel Clock CCi for each input communication channel. IB stores incoming messages in a FIFO order temporarily. CCi contains a copy of the timestamp of the last event message arrived over this communication channel. At the beginning, CCi is set to zero. LVTH=min(CCi) is a local virtual time horizon. LVT can be safely advanced up until LVTH, and the SE should lock and wait unit LVTH advances, if it needs to advance LVT beyond the LVTH value. Events scheduled at moments of virtual time less then LVTH can not arrive from any processor. CI tells SE to process internal and external events, while LVT<=LVTH. SE processes the events as if it would be a sequential simulation system, but it locks if timestamp of the next event is greater then LVTH (pic. 7.1).



Picture 7.1 – Architecture of a conservative logical process.

Processing events may cause the SE to schedule new events for remote processors in the future. An event message is created for each such event, and is deposited into an output FIFO-buffer OB. The messages are retrieved from the OB by a communication system and then delivered to remote processors. A conservative simulation engine algorithm sketch is at pic. 7.2.

```
while LEVL.has_events() or
      CI.LVTH<? do // external events may occur
begin
  if LEVL.next_event_ts()>CI.LVTH then
  begin
    wait_until_an_event_message_arrives();
    continue;
  end;
  LVT:=LEVL.next_event_ts();
  while LEVL.next_event_ts()==LVT do
  begin
    node:=LEVL.next_event_node();
    value:=LEVL.next_event_value();
    LEVL.rermove_next_event();
    if value==NO_CHANGE then //nothing changed
      continue;
    if SFSM.node_value(node)==value then
      continue; // value is not changed
    SFSM.set_node_value(node,value);
  end;
  SFSM.update_affected_elements();
  CI.send_external_events();
```

```
end;
```

Picture 7.2 – Conservative simulation engine algorithm sketch

The task:
1. Create a class for conservative simulation engine.
2. Provide tools to control deadlocks.
3. Create an application for distributed simulation of digital devices.

The report contents:
1. The front page.
2. The laboratory work theme.
3. The task.
4. The source code of all classes created in this work.
5. The source of all simulation tests
6. The simulation results
7. Screen forms
8. Conclusions.

The quiz:
1. What does the SE do?
2. What are the differences of conservative and optimistic SEs
3. What will happen if a remote event will be added to the LEVL at a moment, when the SE retrieves and applies to the SFSM events scheduled for current moment of virtual time?
4. What will happen if a struggler event arrive?
5. When a conservative SE will lock?
6. When a conservative SE will unlock?

LITERATURE

1. Kshemkalyani A.D., Singhal M. Distributed Computing: Principles, Algorithms, and Systems. - London: Cambridge university press, 2008. – 736 p.

2. Tools and Environments for Parallel and Distributed Systems (International Series in Software Engineering) /A. Zaky, T. Lewis. – London: Springer, 1996. - 320 p.

3. G. Coulouris, J. Dollimore, T. Kindberg, G. Blair. Distributed Systems: Concepts and Design. – London: Addison Wesley, 2011. - 1008 p.

4. Formal Methods for Distributed System Development (IFIP Advances in Information and Communication Technology) / Tommaso Bolognesi, Diego Latella. – London: Springer, 2000. - 424 p.

5. Fleischmann A., Tischer J. Distributed Systems: Software Design and Implementation. – London: Springer, 1994. – 390 p.

6. Bal H. E. Programming Distributed Systems. – NY: Silicon Pr, 1990. – 282 p.

7. Pattison T. Programming Distributed Applications with COM+ and Microsoft Visual Basic (DV-MPS Programming). – NY: Microsoft Press, 2000. – 456 p.