Investigating Process Scheduling

What are the Learning Outcomes of this tutorial?

At the end of this lab you should be able to:

1. Enter source code in the compiler and compile it to executable programs.
2. Load the programs into the CPU simulator's memory.
3. Create processes from programs in the OS simulator.
4. Select different scheduling policies and run the processes in the OS simulator.
5. Explain the differences between pre-emptive and non-pre-emptive scheduling.
6. Locate the CPU register values in a process's PCB when it is in the ready queue.
7. Explain how the CPU register values in PCB are used in Round Robin scheduling.

Why are CPU and OS Simulators used for this tutorial?

The computer architecture tutorials are supported by simulators, which are created to underpin theoretical concepts normally covered during the lectures. The simulators provide visual and animated representation of mechanisms involved and enable the students to observe the hidden inner workings of systems, which would be difficult or impossible to do otherwise. The added advantage of using simulators is that they allow the students to experiment and explore different technological aspects of systems without having to install and configure the real systems.

What is the basic theory behind the tutorial exercises?

The different OS scheduling policies are discussed during the lectures on Process Management. This tutorial is based on these lectures.

Conduct of the tutorial

The following practical exercises are designed to help you better understand the theory covered during the lectures. You will only fully benefit from these exercises if you take note of the following guidelines:

- Familiarize yourself with the basic theory (see the lecture notes)
- Follow the instructions in FULL
- Attempt ALL the question(s) at the end of each exercise
- Complete ALL the exercises in this tutorial
- Work with other member(s) of your group

You are expected to work in a small group of students (two or three). There is strong evidence that learning is best facilitated by working in collaboration with your peers and under the supervision of your tutor.

Preparation for the tutorial

The CPU-OS Simulator software is installed on all the PCs in the lab. Ask your tutor for help if you have difficulty locating it.

Note: From time to time, the CPU/OS simulator software is updated. In this case your tutor may supply you with or direct you to another version to update the one on the drive. Please follow your tutor's instructions on how to do this.

Warning: Sometimes, due to a (yet undiscovered) bug in the simulator software, the simulator may crash. If this happens re-start the simulator and carry on from where you left. Please make sure you save your work, e.g. source code, at regular intervals to minimise the impact of this.

Now, we start the tutorial exercises

Learning outcome 1: Entering source code in the compiler and compiling it to an executable program.

You need to create some executable code so that it can be run by the CPU simulator under the control of the OS simulator. In order to create this code, you need to use the compiler which is part of the system simulator. This compiler is able to compile simple high-level source statements similar to Visual Basic. To do this, open the compiler window by selecting the **COMPILER…** button in the current window. You should now be looking at the compiler window.

In the compiler window, enter the following source code in the compiler's source editor window (under **PROGRAM SOURCE** frame title):

```
program LoopTest
    i = 0
    for n = 0 to 40
        i = i + 1
    next
end
```

Now you need to compile this in order to generate the executable code. To do this, click on the **COMPILE…** button. You should see the code created on the right in **PROGRAM CODE** view. Make a habit of saving your source code.

Click on the button **SHOW…** in **BINARY CODE** view. You should now see the **Binary Code for LOOPTEST** window. Study the program code displayed in hexadecimal format. Provide brief answers to the following questions and then close the window.

Q1. How does the hexadecimal code here relate to the information in the compiler window and the CPU simulator window?

The hexadecimal code relates to the information in the compiler window and the CPU simulator window because the compiler window takes in our code and converts it to hexadecimal so that around CPU can understand.

Q2. Click on the **SHOW INSTRUCTION STATS…** button and make a note of the most used instruction. Does this surprise you? Explain.

The MOV is the most used instruction. This does not surprise me since we are always copying stuff from the register.

Learning outcome 2: Loading the program into the CPU simulator's memory.

Now, this code needs to be loaded in memory so that the CPU can execute it. To do this, first we need to specify a base address (in **ASSEMBLY CODE** view): uncheck the box next to the edit box with label **Base Address**, and then enter 100 in the edit box. Now, click on the **LOAD IN MEMORY…** button in the current window. You should now see the code loaded in memory ready to be executed. You are also back in the CPU simulator at this stage. This action is equivalent to loading the program code normally stored on a disc drive into RAM on the real computer systems.

Q1. Briefly explain what the column headers named **PAdd** and **LAdd** signify. Why are they different?

PAdd is the physical address while LAdd is the logical address. Theyare different because when we load memory to 100 so the physical address points to 100 while the logical address points to the logical address.

Q2. Observe the values of the **PC** and the **BR** registers. What do these values signify?

The value in PC denotes the logical address, and BR is the base register, which is the denotes the physical address

Learning outcome 3: Create processes from programs in the OS simulator.

We are now going to use the OS simulator to run this code. To enter the OS simulator, click on the **OS…** button in the current window. The OS window opens. You should see an entry, titled **LoopTest**, in the **PROGRAM LIST** view. Now that this program is available to the OS simulator, we can create as many instances, i.e. processes, of it as we like. You do this by clicking on the **CREATE NEW PROCESS** button. Repeat this four times. Observe the four instances of the program being queued in the ready queue which is represented by the **READY PROCESSES** view.

Learning outcome 4: Select different scheduling policies and run the processes in the OS simulator.

Learning outcome 5: Explain the differences between pre-emptive and non-pre-emptive scheduling.

Make sure the **First-Come-First-Served (FCFS)** option is selected in the **SCHEDULER/Policies** view. At this point the OS is inactive. To activate, first move the **Speed** slider to the fastest position, then click on the **START** button. This should start the OS simulator running the processes. Now, follow the instructions below <u>without</u> any deviation:

1. Make a note of what you observe as the processes are run (you need to concentrate on the two views: **RUNNING PROCESSES** and the **READY PROCESSES** during this period).

Running Process = p2 & Ready Process = p3,p4

2. When all the processes finished, do the following. Select the **Priority (static)** option in the **SCHEDULER/Policies** view. Then select the **Non-preemptive** option in the **SCHEDULER/Policies/Priority Type** frame. Create three processes with the following priorities (use the **Priority** drop-down list in the **PROGRAM LIST/Processes** view): 3, 2, 4. Slide the **Speed** selector half-way down and then hit the **START** button. While the first process is being run do the following. Create a fourth process with priority 1. Make a note of what you observe.

Process is waiting to complete before going to the one with the highest priority.

3. Now, slide the **Speed** selector to fastest position and wait for the processes to complete (or use the **KILL** button to terminate the processes one by one). Next, select the **Pre-emptive** option in the **SCHEDULER/Policies/Priority Type** frame. Create three processes with the following priorities: 3, 2, 4. Slide the **Speed** selector half-way down and then hit the **START** button. While the first process is being run do the following. Create a fourth process with priority 1. Make a note of what you observe. How is this different than (2) above?

Process stoped what it was executing & went to execute p4.

4. Slide the **Speed** selector to fastest position and wait for the processes to complete (or use the **KILL** button to terminate the processes one by one). Select the **Round-Robin (RR)** option in the **SCHEDULER/Policies** view. Then select the **Non-preemptive** option in the **SCHEDULER/Policies/Priority Type** frame. Create four processes and hit the **START** button. Make a note of what you observe.

Every process runs for a specific period of time, then gives control to the next process.

5. Wait for all the processes to complete (or kill them). Go to the compiler window (use the **COMPILER…** button in the **GO TO** frame for this). Click the **NEW** button in the **PROGRAM SOURCE** view and enter the following source code:

```
program LoopForeverTest1
    n = 0
    while true
        n = n + 1
    wend
end
```

   Compile this code and load it in memory as previously described above (Use **Base Address** 200 in this case). Go to the OS window (use the **OS…** button for this). You should now see an additional entry in the **PROGRAM LIST** view titled **LoopForeverTest1**. Select this entry by clicking on it. We are now going to create processes of this program but this time we will assign a lifetime (in seconds) to each. We'll create three processes with the following lifetime values (use the **Lifetime** text box for the values and also select the **Secs** radio button): 10 seconds, 32 seconds, 6 seconds. Now, we are going to select the time slot value. To do this select 4 seconds from the drop-down list in **SCHEDULER/Policies/RR Time Slice** view. Hit the **START** button and wait until all processes finish.

   Open the **OS Activity Log** window by clicking the **VIEW LOG…** button in the **SCHEDULER/Views** tab. Observe the relevant log entries and check out the sequence of the running processes and note the time spent by each process during each running state. Record this information in the box below.

After every 4 seconds a new process takes control. p3 is 6 seconds so it finishes first, then p1, and finally p3.
P1 = 4 seconds, P2 = 4 seconds, P3 = 4 seconds, P1 = 4 seconds
………

Learning outcome 6: Locate the CPU register values in a process's PCB when it is in the ready queue.

Learning outcome 7: Explain how the CPU register values in PCB are used in Round Robin scheduling.

Now, go to the compiler and enter the following code in the source code editor. You need to click on the **NEW** button to start a new source code tab in the editor:

```
program LoopForeverTest2
    while true
        n = n + 1
        i = i + n
        p = n + i + 5
    wend
end
```

Compile this new source and load in memory (use **Base Address** 300 in this case). Go to the OS simulator. You'll observe a third entry in the **PROGRAM LIST** view titled **LoopForeverTest2**. Click on the entry **LoopTest** and create a process. Click on **LoopForeverTest2** and create a process of it. There should now be two processes in the ready queue. Make sure the **Round Robin** scheduling is selected; the priority type is **Non-preemptive** and that the **RR Time Slice** is set to 10 ticks (you can select this from the drop-down list).

1. Select each process in turn and click on the **PCB…** button. Observe the values of the **PC Registers** in each case and make note of these in the box below. Also, note down what other information is in the PCB (select three examples) and briefly explain why this information is required.

The value for PC register is at 0 because we have not started our program. So at the initial state it'll be at the starting position. There is also the program name which is important so we know what program it's running. There is also the parent id which is required so we know what comes before. And finally there is a process state which is required so we know what state each process is at

2. Select the **Suspend on run** check box in the **READY PROCESSES** view.
3. When the currently running process is put back in the ready queue, select it and click on the **PCB…** button in the ready queue. Make a note of the **PC Register** value and the values of the registers R00 to R05.

PC register is at 53 and r00 = 0, r01 = 0, r02 = 0, r03 = 1, r04 = 40, r05 = 1

4. Click on the **RESUME…** button and when the second process is put back on the ready queue make a note of the **PC Register** and the R00 to R05 register values in its PCB in the above box.

PC register is at 51 and r00 = 0, r01 = 1, r02 = 1, r03 = 2, r04 = 2, r05 = 2

5. Now, select the **Suspend on run** check box in the **RUNNING PROCESSES** view. Slow down the OS simulation to minimum. Click on the **RESUME** button and when the queued process is put back in running state the simulation will be automatically suspended. Click on the **RESUME** button and immediately click on the **SUSPEND** button (this is the same as the **RESUME** button). Make a note of the values in the **PC** register and the registers R00 to R05. Compare these values with the ones recorded in (3) and (4) above.

PC register is at 53 and r00 = 0, r01 = 0, r02 = 0, r03 = 1, r04 = 40, r05 = 1
PC register is at 51 and r00 = 0, r01 = 1, r02 = 1, r03 = 2, r04 = 2, r05 = 2 . They are the same from 3 and 4 above.

6. Click on the **RESUME** button and kill the currently running process. Click on the **RESUME** button and immediately click on the **SUSPEND** button. Make a note of the values in the **PC** register and in the registers R00 to R05. Compare these values with the ones in (3) and (4) above. Briefly explain what has been happening.

When we resume the process it starts at the same PC value again. Since we're clicking resume and suspend at the same time, we're not letting the program to execute. Thats why we have the same pc register and values in the register.