# Operating Systems – Laboratory

**Hristo Trifonov**

*7/March/2019*

*For SDip in Embedded Systems Engineering*

## Laboratory Assignment #4

**OBJECTIVES:**

The objectives of this laboratory assignment are as follows:

1) Learn how to use **named** pipes for IPC (inter-process communication)
2) Learn how to write shell script program to manage concurrent processes
3) Learn how to write a simple Bash shell function to **trap** signals

**INSTRUCTIONS:**

- Students will provide individual submissions (however, learning/study cooperation is encouraged).
- A short report **document** file must be submitted to describe the operation of your program and to comment on any problems etc. (see Addendum for details).

**SUBMISSION:**

Students will submit via SULIS (EE5012 page) by 23:55 hours, Thursday 14th March 2019. Late reports will not be accepted. The submitted files will be the:

- a **report** file (pdf) – see format in the Addendum
- two scripts (**proc_A** and **proc_B**)

Please put all of the above in a folder named **"Assignment4_yourIdNumber"** and compress/archive with **zip**, **tar** or whichever program you prefer before submitting to SULIS.

The student name and ID number is to be on the heading comments of any script file. The programs are to be commented for readability. Individual submissions only will be accepted. The student can be asked to demonstrate the working programs in the lab.

**Assignment assessment weightings:**

| |
|---|
| Assignment #1 10% of module |
| Assignment #2 10% of module |
| Assignment #3 10% of module |
| **Assignment #4 10% of module …. this assignment** |
| Assignment #5 10% of module |
| *There will be a compulsory exam question in the final exam based on the laboratory assignments.* |

# INSTRUCTIONS

Please complete the following exercise.

---

**EXERCISE: an exercise based on UNIX pipes**

**STEPS:**

1) Study the UNIT 4 – Tutorial (Named pipes)
2) Write a solution for the two-pipe example scheme as described below

**Instructions:**

UNIT 4 – Tutorial shows an example where two processes, **process1** and **process2**, communicate via a single named pipe, called **pipe1**.

Study the programs and run these script programs on your computer to satisfy yourself that they do work properly.

Modify the two programs as follows, where the two modified programs will now be named as **proc_A** and **proc_B**:

*The Figure 1 below shows a diagram for the modified system.*

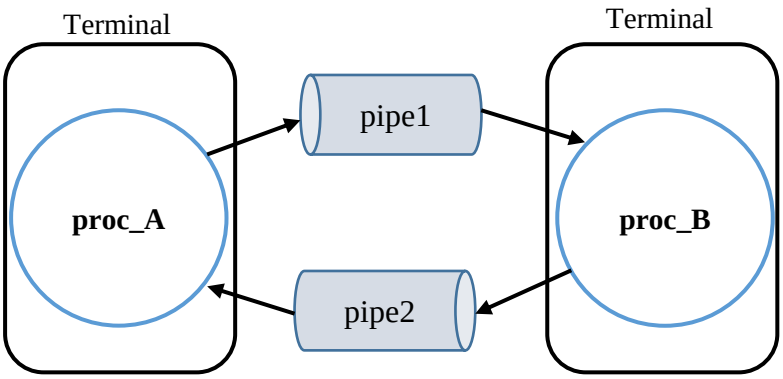


*Figure 1. Two pipe example.*

- **proc_A** starts first and sends a message on **pipe1** 5 times. Then sends the "Last message" (same as the example process1 and process2).
- **proc_B** starts in a separate terminal window as shown in figure 1.
- When **proc_B** receives the message "Last message" on **pipe1**, it then creates a pipe named **pipe2** and sends its **PID** over **pipe2**. After that it prints to its terminal "Proc_B looping" every second.
- The **proc_A** will then read **pipe2** and write the message that was received to the screen (i.e. standard output).
- **proc_A** will send a "HUP" signal to **proc_B** using the **PID** that was received over **pipe2**. It will then remove **pipe1** and **exit**.
- The **proc_B** process will use a **trap** function to intercept the HUP signal and it will also lower its priority with 5.
- The **proc_B** will remove **pipe2** and then **exit**.

# *Hints: exercise 3*

The **pipe2** is created just like **pipe1** in the tutorial example - so there is no catch here, the aim of the exercise is to simply get some experience in using a named pipe.
You will notice that a program can block on writing to a pipe (if the pipe is full) and can block on reading a pipe (if the pipe is empty).
This can be frustrating in getting the program to work, even for a simple program like this. However, if you use the **bash debug mode** then you will have a better opportunity to see what is happening in your scripts.

You will need to obtain the PID for **proc_B** so that you can increase its **nice value** (decrease priority).
You might go to great efforts to find out this PID, by using the **ps** command etc. However, there is no need for this complexity. All you need to know is this issue can be easily solved by knowing that the shell variable **$$** always represents the PID of the current process that was started.
Do use something like: *echo "The PID of the current process is: $$"*

Don't forget to use the *renice* command to adjust the priority of a running process as it was shown on the lecture.

## ADDENDUM

### The document file

The submission for this laboratory assignment will include a document. Note, the document does not at all need to be very long and wordy – but must be of good quality, to the standard of a small technical report, and presented as listed below:

1) The file will be submitted as a **PDF** file.

2) The document will have the following information and sections

**Front page**
Title page with student name, ID, date, module code, assignment number (e.g. *Assignment #1*)

**Requirements**
Briefly summarise the assignment requirements from the assignment instructions in the handout.

**Description of solution**
Describe your solutions noting any special problems or issues.

**Testing and results**
State how you tested your program and record any results, using **screenshots** to show actual outputs.

**Statement of completion**
Briefly make a statement saying that you have completed all of the requirements, or summarise any aspects that you could not complete.

**Source code**
Include your source code as part of this document. You will also submit a separate plain text source code file or files as stated on the cover page of this assignment.

**NOTE: A student can lose up to 30% of assignment marks for a bad report.**