**ECS518U Operating Systems**

**Lab 6: Monitoring processes using SystemTap**

**This exercise is not assessed (but you must get it ticked off by the demonstrators) Preliminary**

# Aims

The aims of this lab exercise are to:

1. Get familiar with the use of SystemTap to trace the OS’s behaviour.
2. Understand how processes are scheduled.

# Preliminaries

1. Log in to [(https://vspace.comp-teach.qmul.ac.uk/)](https://vspace.comp-teach.qmul.ac.uk/) and connect to the dedicated virtual desktop available there (more detailed instructions on the Lab6 section on QMPlus).
2. You are logged in as user “Gollum” in the dedicated desktop. The **sudo password** which you need to run the System Tap scripts **is 123456**.
3. You can use the Chrome browser in the visrtual desktop to download any files you need from QMPlus.

You can find more detailed information about SystemTap on QMPlus. However, you can still complete the lab work without looking there.

# Step 1: Burst Length Distribution

## Gathering Burst Length Data

|  |
| --- |
| The SystemTap scripts are run using the stap command. **The scripts operate in two modes:**   1. Monitoring an interactive process that is already running (e.g. chrome, gedit, etc.):   sudo stap –v burst\_length1.stp *command\_name*  where command\_name is the name of a command as it appears in the ‘CMD’ column of ‘ps –a’ or ‘ps –e’. Examples are ‘gedit’, ‘chrome’, etc. After at least 5-10 seconds, press the keys Ctrl and c in the window from which the script was run to stop the collection of data by aborting the System Tap command. A report will then be printed. **Note** that the process must already be running (i.e. start the process, then monitor it using the script).   1. Monitoring a process that runs for only a short time (e.g. ls, ps):   sudo stap –v burst\_length1.stp –c ”ls –l > /dev/null”  In this mode, the command (here ‘ls -l’) is run by the script which stops when the command completes. The output of the command also appears – we have prevented this above by redirecting it to the null device. If you do not know already, try to find out what  /dev/null is, and what redirection (using <, >) is in Linux. Run the script with and without the redirection, see the difference. |

The SystemTap script ‘**burst\_length1.stp**’ is available in the scripting folder to download. Its purpose is to gather data about the ‘burst length’ – the length of time a process runs each time it is scheduled to execute. Download it (it is easier to run the browser in the dedicated desktop and download all scripts directly there or transfer the files via the Guacamole menu).

**Note:** the ‘-v’ option stands for ‘verbose’ and causes some debugging information to be printed. You will see that the system script is processed in five passes; the final pass should be: “Pass 5: starting run.”

**DO NOT COPY/PASTE THE COMMANDS FROM THE PDF DOCUMENT AS THIS CAN CAUSE PROBLEMS, TYPE THEM IN THE SHELL YOURSELVES!!!**

To gather information about the scheduling of a process, the steps are:

1. Connect to the dedicated virtual desktop using your standard QMUL credentials.
2. Run the SystemTap scripts as shown in the grey box above. Once SystemTap runs in the terminal, no further commands can be entered in that same terminal.

 **If you ran stap without ‘-c’ (mode 1 in the grey box)** then first start and use the program to be monitored (e.g. chrome), perhaps in a second terminal window, or by selecting it from the Applications -> Internet menu. Then in another terminal window run the stap script to monitor that process. Interact with Chrome, browse, etc. After some seconds, stop the collection of data by aborting the stap command by pressing ‘Ctrl-C’ in the window from which stap was run. A report will then be printed.

An example session including the report is shown below:

|  |
| --- |
| sudo stap -v burst\_length1.stp chrome  *debugging information omitted (steps 1-4 omitted)*  Pass 5: starting run.  ^C  Process 14725 has 9040 bursts: min 1 avg 195 max 10816 us/burst  Process 14725 has 9040 bursts (the most) Burst length distribution (microseconds)  value |-------------------------------------------------- count 0 | 0   1. | 10 2. |@@@@@@@ 264   4 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 1831  8 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 1374  16 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 1458  32 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 1502  64 |@@@@@@@@@@@@@@@@@@ 681  128 |@@@@@@@@@@@@@@ 525  256 |@@@@@@@@@@@@@@@ 559  512 |@@@@@@@@@@@@@@ 534  1024 |@@@@ 162  2048 |@@ 77  4096 |@ 44  8192 | 19  16384 | 0 32768 | 0  Pass 5: run completed in 0usr/20sys/22529real ms. |

The session shows that a single process with PID=14725, was observed. This process was scheduled to execute 9040 times during the observation. The minimum length of execution was 1 microsecond, the average length of execution was 195 microseconds and the maximum was 10,816 microseconds. The overall duration of the observation was 22,529 msec.

The graph shows a histogram of the ‘burst length’ (the time the process executes for each time it is scheduled). The burst lengths ‘b’ are on the left, using intervals b=0, 1 ≤ b < 2, 2 ≤ b < 4, 4 ≤ b < 8, 8 ≤ b < 16 etc. Notice that each interval is twice the width of the previous.

To collect the session in a file, you can either use cut and paste or redirect output to a file (you would still need to press Ctrl-C to stop the script after some time):

sudo stap –v burst\_length1.stp *command\_name* > *file\_name*

# Experiment (the main aim of the lab)

Observe several commands, including both simple ones (e.g ‘sort’, ‘ps’, etc.) and programs such as chrome, gedit, etc.). Why not try some of the programs you have written in other modules and see how the scheduler deals with them? Maybe your shell from Lab 5?

Make sure you also use the testProgram (download from the scripting folder on QMPlus), information below.

A program ‘**testProgram**’ is available. It takes one argument – a number - and often returns the number one greater. However, some input values (e.g. 230) cause the program to loop endlessly. You may wish to run this program at the same time as e.g. chrome to see what the scheduler does. How would a simple round-robin pre-emptive scheduler behave? (this is not the scheduler used here though)

To use the program, you will have to change its permissions and make it executable, e.g. chmod 744 testProgram. Run testProgram in one terminal in an endless loop (./testProgram 230). Run a stap script to monitor testProgram in another terminal (e.g sudo stap –v burst\_length1.stp *testProgram*). Press Ctrl-C after some time to see the burst distribution.

In your various experiments answer the following:

* Describe the distribution of burst times (for example: “they are all roughly the same size”, “they look wildly different”, etc.) for the ‘testProgram’, chrome and for at least a couple of other examples of shorter commands, but feel free to experiment more. Explain your findings. If you find them different, why are they different?

Text

Description automatically generated with medium confidence

Chrome: 19 bursts were recorded for the main process. The burst lengths are similar, ranging from 32-2048. Count for each row is similar. A high number of bursts are recorded because this is an interactive process, often waiting for user input.

Text

Description automatically generated

Ls: Only 2 bursts is recorded, which are in the 128us &1024us row. The program doesn’t care about user input, and the task complexity is low, so it only have single short bursts.

A picture containing text

Description automatically generated

testProgram: 4 bursts are recorded: 4us, 64us and 512us. This program is also not interactive. It should also be less computationally intensive than Ls, therefore the burst lengths are shorter.

Text, table

Description automatically generated with medium confidence

testProgram 230: 411 bursts: Lengths ranging from 16-262144us, with the majority being 1024-65536. This program is an endless loop, so the scheduler tries to allocate as many CPU cycles as it can before forcing a context switch.

Text

Description automatically generated

Ps -aux: 269 bursts recorded. It has a lot of bursts in the 2-64us range, suggesting that the smaller bursts might be a result of ps printing line by line. It is also more intensive than ps.

Text

Description automatically generated

Ps -aux > /dev/null: 4 bursts recorded. We know that it didn’t print anything and as a result there are much fewer bursts.

* The maximum burst length value will be rather different when you monitor chrome compared to when you monitor testProgram. Why is that the case?
  + Chrome: 3589us. testProgram: 620us.

It is because the testProgram has nothing much to do, whereas the chrome process needs to render the page and run JavaScript VMs, therefore requires more CPU cycles before triggering a context switch.

* Repeat your experiment with testProgram 230 by modifying its niceness level with the nice command (e.g. set the nice level to 19). Do you observe any differences? If so, explain.

Text

Description automatically generated

* + The burst lengths are similar, but the number of bursts has decreased.
* While SystemTap collects data for testProgram 230, start using another program (e.g. chrome – open some tabs, browse some web sites, etc.). Conduct separate experiments to monitor the bursts for testProgram when you are interacting with chrome and then when you are not interacting with chrome. Try to explain the differences.

Table

Description automatically generated

* + The average and max burst length are greatly shortened.
* Why do very short burst times occur for some processes? Think of at least 3 reasons.
  + Waiting for user input
  + Frequent system interrupt calls
  + Other computationally intensive processes running in the background
* If you monitor Linux commands using the –c option (mode 2 in the box in Page 1) you will see that they complete after a couple of bursts (e.g. try ls –l). A couple of other commands to try that will keep for a bit longer are ping and find. Try these

examples in mode 2: ping –c10 8.8.8.8 and find /usr –size -10c

(what do these commands do? Find out). Can you think of any way that you can modify the burst behaviour that you observe for these commands without changing the command parameters?

* Change the niceness

# Step 2: Burst Length Time Line

A second version of the SystemTap script, burst\_length2.stp, produces data in a form suitable for a spreadsheet. It is best to redirect the output to a file with extension ‘csv’ that can then be processed by a spreadsheet.

sudo stap –v burst\_length2.stp *command\_name* > *file\_name.csv*

The script first produces a row for each context switch event, giving the number in the sequence, the process id and the burst length. For example:

Number,PID,microseconds

0,16932,1540

1,16932,7254

2,16932,850

3,16932,763

4,16932,9884 5,16932,2954 etc …

This is followed at the end by a summary

PID,Count,Min,Average,Max

16932,1230,9,7123,10749

Use a spreadsheet to draw a graph (bar chart) of the burst times for a couple of the experiments you did in Step 1 (no need to do it for all, choose 2):

* Draw a bar graph of the burst lengths. Can you see any patterns in your data?

Chart

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

* Functions like the following in Excel may help in drawing the graph: COUNTIF, FREQUENCY. If you have not used these functions before there are plenty of online resources that explain how to use them with examples. There may be other ways to obtain the graph.

o As a hint you can define ranges of burst values (x-axis) (e.g. 0-100, 101-200, etc. but create your own ranges based on the data you collected) and count how many ‘rows’ have values that fall within each range (y-axis).

Most bursts are in the first column

**Optional:**

There is another script available to you: cycle\_thief.stp.You can run it as sudo stap –v cycle\_thief.stp –x pid to monitor burst times for process with pid (e.g. 45678), as well as length of periods off the CPU, the interrupt signals that have stolen time off the CPU for the process and the list or processes that have got CPU time apart from the process we monitor. You could use this script to dive deeper into your investigations for context switches in Step 1.

The script sched\_switch.stp is also available to download and experiment with. You can run it as sudo stap –v shced\_switch.stp name process\_name or sudo stap –v sched\_switch.stp pid process\_pid. For example:

sudo stap –v sched\_switch.stp name chrome or sudo stap –v sched\_switch.stp pid 45678

The script will trace the process based on pid/name and print the scheduler switches happening with the process. If no arguments are passed, it displays all the scheduler switches. This can be used to understand which tasks schedule out the current process being traced, and when it gets scheduled in again. Output is rapid as you can imagine so it may be best to redirect. Look at the script for what the output actually is.

No specific tasks are prescribed for you here, plenty of room for improvisation and diving into what the scheduler is doing, just explore. **Get in touch with the lecturers during the labs if you have tried this out and want to discuss what you observe.**

**Optional:** For those curious enough, there is a page with many stap scripts available, feel free to explore and try things out. We will be happy to discuss things with you.

<https://sourceware.org/systemtap/examples/keyword-index.html>

# Answer Sheet

You do not need a specific answer sheet for this lab, but for any experiments you do in Step 1 present the output in a document that you can discuss with your demonstrator or the lecturers.

For example, in Step 1 (Experiment) there are a number of questions. Present data captured from the script to justify your answers to the questions.

For Step 2 just prepare the graphs as required and be able to discuss what you observe.