# OS Warmup

CS453: Operating systems

## Overview

We are going to create a logging program named **ps-logger** that logs the percentage of multithreaded programs on the underlying Linux operating system.

The command **ps augx** lists details for all processes under Linux. Play with the command **ps augx** on Linux to see the format of the output. In particular, the 8th column shows the state of the process. Check the man page for **ps** to find out how the state of the process encodes whether it is multithreaded or not.

You must use the **ps augx** command so that we can get some practice with parsing in C. There are other options to **ps** that will simplify the parsing but we will not use them in this project. Similarly we can use filters on command line such as **awk**, **grep**, **wc**, etc to get the answer but that again is not the point of this project!

Write your program to read in the output from **ps** in the log file **ps.log** generated with the **ps augx** command that your program will run with the **system** command. We will be parsing this file, checking if the state column contains the special character that determines if the process is multithreaded or not. We will count the number of multithreaded and the total number of processes and print them out (as well as the percentage).

Next, combine it with a loop so that your logger program sleeps for user specified (via a command line argument) number of seconds and then reruns the **ps augx > ps.log** command to capture its output and parse it again to print the output for that point in time. It keeps doing this forever. You will need to use the system calls **sleep()** and **system()**. Note that in general you do not want to use the system() function call you want to use fork/exec. However, this is intended to be a very simple warm up project to shake the cobwebs off your C programming skills so your instructors have allowed the use for this program only.

## Sample Output

```
[user@localhost user(master)]$ ps-logger
Usage: ps-logger <sleep interval (seconds)>

[user@localhost user(master)]$ ps-logger 0
ps-logger: Specified too small an interval. Must be at least 1 second.
```

```
[user@localhost user(master)]$ ps-logger -2
ps-logger: Specified too small an interval. Must be at least 1 second.

[user@localhost user(master)]$ ps-logger 2
ps-logger: Using a default sleep time of 2 seconds.
ps-logger: 72/252 processes are multithreaded. [28.57%]
ps-logger: 73/253 processes are multithreaded. [28.85%]
ps-logger: 73/253 processes are multithreaded. [28.85%]
ps-logger: 72/252 processes are multithreaded. [28.57%]
ps-logger: 71/251 processes are multithreaded. [28.29%]
ps-logger: 71/251 processes are multithreaded. [28.29%]
ps-logger: 71/251 processes are multithreaded. [28.29%]
ps-logger: 71/252 processes are multithreaded. [28.17%]
ps-logger: 71/251 processes are multithreaded. [28.29%]
ps-logger: 72/252 processes are multithreaded. [28.57%]
ps-logger: 73/255 processes are multithreaded. [28.63%]
ps-logger: 73/256 processes are multithreaded. [28.52%]
ps-logger: 71/254 processes are multithreaded. [27.95%]
ps-logger: 74/258 processes are multithreaded. [28.68%]
ps-logger: 83/267 processes are multithreaded. [31.09%]
ps-logger: 83/267 processes are multithreaded. [31.09%]
^C
[user@localhost user(master)]$
```

## Valgrind

Run your program under valgrind and verify that there are no memory errors or leaks. Note that your program is running in an infinite loop, so you will have to kill it with **Ctrl-c** to get output from Valgrind. Valgrind may generate an error as a result of using Ctrl-c but you can safely ignore it. A great way to increase your class participation is to post the Valgrind stack trace that can result (**HINT: The valgrind error doesn't happen 100% of the time due to something called a race condition that we will learn about later in the semester)**

## Submission

There are two places to submit for this project 1) you must submit your code to backpack and 2) you must submit the SHA1 hash to blackboard.

## Submit to Backpack

Required files to submit through git (backpack)

1. Makefile
2. Ps-logger.c
3. README.md

*Push your code to a branch for grading*

### Run the following commands

1. make clean
2. git add <file ...> (on each file!)
3. git commit -am "Finished project"
4. git checkout -b os-warmup
5. git push origin os-warmup
6. git checkout master

### Check to make sure you have pushed correctly

- Use the command **git branch -r** and you should see your branch listed (see the example below)

```
$ git branch -r
  origin/HEAD -> origin/master
  origin/master
  origin/os-warmup
```

## Submit to Blackboard

You must submit the sha1 hash of your final commit on the correct branch. Your instructor needs this in order to troubleshoot any problems with submission that you may have.

- git rev-parse HEAD

## Grading Rubric

Provided via backpack.