Extra Credit Assignment

RESOURCE & ACTIVITY PROFILING FOR A PROCESS

This assignment targets resource and activity profiling for processes. This assignment will contribute 2 points towards your overall course grade. You should attempt this extra credit assignment ONLY after you have successfully wrapped up HW5 (which accounts for 10% of your course grade) and are looking for a challenge. The 2-day late submission window will not be available for this assignment.

The assignment should be implemented in C and you will be gaining expertise in how to perform File I/O in C.
DUE DATE: Wednesday November 1st, 2023 @ 8:00 pm

1 Description of Task

The program you are developing can be thought of as a simplified activity monitor. The program will be provided a process ID (of a currently executing process e.g., a web browser) as an argument. The program will then report on a slew of process execution statistics. These include:

1. CPU utilization
2. Memory utilization
3. Elapsed time
4. Page faults

To report these statistics, you will work with files located in the **/proc** directory, which houses runtime information for the system. For a running process, you can find data concerning it in the **/proc/[pid]** directory, where **[pid]** is its process ID. The documentation for **/proc** provides specific details on the information contained in these files.

2 Requirements of Task

The program should execute on machines in the CSB-120 lab. These statistics should be retrieved by accessing files in the following directories: **/proc** and **/proc/[pid]**. No other tools or utilities should be used to retrieve this information; doing so, will result in an automatic zero.

The **/proc/[pid]/stat** and **/proc/[pid]/statm** files have similar formats. Both contain space delimited data about the system. In the man page, **man proc**, the order and types of the fields in these files are shown. For example, here are the first two entries from the **/proc/[pid]/stat** section of the **proc** man page:

1. pid %d
   The process id.
2. comm %s
   The filename of the executable.

Given a pid, these first two values from can be read in this way:

```
FILE *proc_file = fopen("/proc/[pid]/stat", "r");
pid_t pid;
char exec_name[128];
fscanf(proc_file, "%d %s ", &pid, &exec_name);
```

Another way to read these fields is by using **strtok**, which would allow the space-delimited tokens to be read individually, in the order shown in the man page.

The **/proc/[pid]/stat** file contains status information for a running process. Given a process ID, the following fields should be read from this file (0.2 points each):

1. Executable name
2. Process state
3. Number of page faults
4. Parent Process PID
5. Process Group ID
6. Time spent in Kernel Mode
7. Time spent in User Mode

The **/proc/[pid]/statm** file contains memory information for a running process. Given a process ID, the following fields should be read from this file (0.2 points each):

1. Virtual memory size
2. Resident pages
3. Shared pages

Error Handling:

1. If the program is specified an invalid ID or the ID of a process that is currently not active, the system should gracefully return an output indicating that the process is not currently active.

Restrictions and Deductions:

[R1]. You should not use ANY external library to solve this problem. There is a 2-point deduction if you do so.

[R2]. You should not leak any memory that you have allocated within your program. There is a 1-point deduction if you do so.

3 Example Outputs

Print out of what the outputs of the executing programs look like (note that your outputs may differ).

```
> ./profiler $(pgrep -n chrome)
[./profiler] checking pid 561735
Executable:      (chrome)
ppid:            561529
pgrp:            560780
State:           S
User mode:       0.000 sec
Kernel mode:     0.000 sec
Virtual memory: 296813862 bytes
Resident pages: 16750
Shared pages:   12628
Page faults:     0
```

```
> ./profiler $$
[./profiler] checking pid 178525
Executable:      (bash)
ppid:            178504
pgrp:            178525
State:           S
User mode:       0.010 sec
Kernel mode:     0.010 sec
Virtual memory: 67011 bytes
Resident pages: 2312
Shared pages:   1460
Page faults:     0
```

Nota Bene:

The **sysconf** function defined in **unistd.h** may be useful for working with system dependent units.

4 What to Submit

Use the CS370 *Canvas* to submit a single .tar file that contains:

- All C files related to the assignment (please document your code),
- a Makefile that performs both a *make clean* as well as a *make all,*
- a README.txt file containing a description of each file and any information you feel the grader needs to grade your program.

Filename Convention: You should call the executable that drives your program profiler. You can name your other supporting C files anything you want. The archive file should be named as <LastName>_<FirstName>_ExtraCredit.zip. E.g., if you are Cameron Doe and submitting for this ExtraCredit, then the tar file should be named Doe_Cameron_ExtraCredit.zip.

5 Grading

This assignment would contribute a maximum of 2 points towards your final grade. The grading will also be done on a 2-point scale. The points are broken up as follows:

You are required to work alone on this assignment.

6 Late Policy

Click here for the class policy on submitting late assignments.