Program 3 Analysis

Task 1

For task1, we created several applications that we also applied to later code. We ended up with some interesting and unexpected results. First of all, we created task1.c, which calculates an expensive function that takes several seconds to run, and meanwhile it looks at the memory usage by reading from meminfo. After a while we decided to create a separate piece of code, freemem, which just reads the available memory. We can then run a process in one window and use another to run freemem. With the original task1.c, it usually allocated somewhere in the 150-200 range. Interestingly, memory usage went down when running ping commands, and it happened consistently enough that it didn’t seem to be coincidence. It was also tested with a number of bash commands, all of which used more memory. Freemem has also evolved to be usable for all of the tasks in some way or another.

Task 2

Like task 1, task 2 had strange and unexpected results. The first forked process sometimes resulted in a change in memory usage, but the 2 following forks didn’t seem to change anything, and they both returned 0 for their individual process usage, and the free and available memory counts remained unchanged. Sometimes there was no reported change in memory, so it seems that a number of our test processes, aside from memhog had relatively low effects on memory. This was true for both identical forked processes and execv forked processes. However, we created a program called memhog, that forks 15 children and does seem to make a large dent on memory usage. This was especially true when we ran multiple copies of it. In task2.c we tested identical children, and in task2b we tested children using execv and there was no noticeable difference. Interestingly enough, memhog recently stopped finishing its execution before dying. It seems that our vms have started killing the process before it is able to fully finish executing.

Task 3

Task 3 was difficult. At first, I had problems using the method Zach used to read from /proc/meminfo. I constructed the get\_file\_value function, which simplified and corrected the way to read the memory values from meminfo.

At first, for over a day, the file ran as expected; it updated the percentage of memory used, as well as the amounts of memory free and total memory. However, at some point it stopped working correctly. I didn’t notice it until it was too late to determine what environment variables changed. I don’t know why it doesn’t update the memory usage; it did, and should.

The same thing happened with memhog, as Zach mentioned. It seems to create a few child processes, but these never usually take up very much memory. When “top” is executed, the processes don’t appear to take any significant amount of memory at all, and when I check the meminfo file, it looks something like [this](http://puu.sh/hBhjP/d5ec331eea.png). This was taken seconds after memhog was run, and seconds before its output completed. Sometimes, though, the amount of free memory is in the 5-digit range, but isn’t detected by task 3.

Task 4

Task 4 was almost exactly similar to task 3 up until a point. It changed when I had to implement what the program did when 85% memory usage was reached. Task 4 has memory hogging done by memhog, and when the limit is reached, pkill is used on memhog. As in task 3, task 4 had trouble getting memhog to hogmem (bad joke). The process killing seemed to work fine, but I couldn’t measure the memory change after killing because of the aforementioned problems.

Overall Analysis

We learned a lot about memory and memory allocation through this assignment. Unfortunately, there was a variety of results so it makes it difficult to make any definite conclusions about anything, but there is still much to be taken from it.

Aaron:

I learned a lot of how linux stores data about running processes, which is very different from Windows. The virtual file system is unique. I’m guessing that our memory hogging process isn’t able to hog memory like we want it to because the kernel is preventing over-allocation.

I learned that processes are described in their own folder in proc, and their folder titles are their process IDs. Information about each process is stored in each folder.