How to monitor memory consumption - a *short* tutorial

July 18, 2017

Ever wonder how to measure memory consumption while running a program on Eustis?

Here's a short tutorial on **`top`** which provides a very useful memory consumption monitor.

To test your own code, put memLeak.c big40.c, and big40.h in the same folder on Eustis. Compile with the following command (you can, of course, rename the output if you so choose):

```
gcc big40.c memLeak.c -o memLeakCheck
```

Open a second Eustis terminal using whatever method you already use for connecting to Eustis (Putty, MobaXterm, etc.). In the second terminal run the following command:

```
top -d 0.5 -u [your Eustis username] -o -COMMAND
```

In **`top`**, you can see the amount of memory each process is using. The -d flag sets the refresh rate (in this case to every half-second), the -u flag shows only the processes started by a specific user (in this case you), and the -o flag sorts the output based on the field specified (in this case we will sort by COMMAND). The specified field is case sensitive, and the minus sign preceding COMMAND sorts a-z (top to bottom).

In the first terminal, run your program. While it is running, **`top`** will show it as a.out (or whatever you named it) under COMMAND, and the memory it is using (in kB) will be listed under VIRT. If you have no leaks in your, memory used should be a steady number based on your program's size. If you do have a leak, you will see that number climb ever higher and higher until the program eventually crashes. If you don't want to wait for the program to crash or if you feel like being considerate of the other Eustis users, you can force stop the endless loop by going back to the first terminal and pressing ctrl-c.

Here's the best part, you can use this to check any code you want for memory leaks. All you have to do is place the code inside the while loop, and make sure you call you functions (still inside the loop) for destroying whatever the code creates. If **`top`** shows a memory leak then you will know to look in the tested code and the destroying functions.