Dan Smith and Joe Holston

**TASK 1**

* Task1a
  + Foreground: 976980 kB
  + Background: 976987 kB
* Task1b
  + Foreground: 540 kB
  + Background: 832 kB
* Task1c
  + Foreground: 540 kB
  + Background: 832 kB
* Analysis
  + Linux allocates memory based on the size of the process that is being run. For example, task1b is a simple addition program that does not use much memory. Because of this, it only receives less than a 1MB. This shows the contrast to task1a, which uses a whole gigabyte of memory. It is based on the size of the process. There is also another factor to consider when deciding how much memory to allocate. When a process in running in the background, then more memory is allocated to those processes because it is not the main priority. Both size of the process and what else is running play into how much memory is allocated.

**TASK 2**

* Analysis
  + When Linux allocates memory to a child that is identical, it allocates almost the same amount to each identical child. When testing task2.c several times, the children all received 85 - 92 kB, with the parents having more than the children do. This shows that identical children all receive almost identical amounts of memory allocation.
  + When Linux allocates memory to a child that is not identical, or uses an execv to load another process, then the size of the memory allocated depends on the size of the process that is called. For example, we passed in the task1a process in the execv call and it allocated almost 1 gigabyte because that is the size of the process when it runs on its own.

**Testing Procedures:**

* Task 1
  + Test 1 (test how much memory is allocated for a process when it is the only process running)
    - Run task1a and record how much memory is allocated
    - Run task1b and record how much memory is allocated
    - Run task1c and record how much memory is allocated
  + Test 2 (test how much memory is allocated to a process when it is in the background)
    - Run task1a (with & command) so it is running in the background
    - Run task1a and record how much memory is allocated
    - Run task1a (with & command) so it is running in the background
    - Run task1b and record how much memory is allocated
    - Run task1a (with & command) so it is running in the background
    - Run task1c and record how much memory is allocated
* Task 2
  + Test 1 (test memory allocation for identical child processes)
    - Run task2 and record how much memory is allocated per child
    - The higher numbers are the parent’s memory and the lower numbers are the child’s memory.
  + Test 2 (test memory allocation for non-identical child processes)
    - Run task2b and record how much memory is allocated (the process passed into the execv command will report how much memory allocated towards it.)
* Task 3 (Provides an error message any time memory usage goes above threshold- even if it drops down and goes back above)
  + Run sysinfo() and calculate current percentage through (total RAM – free RAM) / total RAM
  + Run test 1
    - Allocates 2GB to a pointer in memory
  + Calculate current RAM usage with sysinfo()
  + Run test 2
    - Allocates 1.5GB to a pointer in memory
    - Calculate current RAM usage with sysinfo()
    - De - Allocates 1.5GB from a pointer in memory
    - Calculate current RAM usage with sysinfo()
  + Run test 3
    - Allocates 1GB to a pointer in memory
  + Calculate current RAM usage with sysinfo()
* Task 4 (Provides an error message any time memory usage goes above threshold and begins to kill processes to attempt to lower memory usage below threshold)
  + Run sysinfo() and calculate current percentage through (total RAM – free RAM) / total RAM
  + Run test 1
    - Allocates 2GB to a pointer in memory
  + Calculate current RAM usage with sysinfo()
  + Run test 2
    - Allocates 1.5GB to a pointer in memory
  + Calculat e current RAM usage with sysinfo()
  + Run test 3
    - Allocates 1GB to a pointer in memory
  + Calculate current RAM usage with sysinfo()

**Overall Analysis and Summary**

* For the first task, something that we learned was processes that run in the background are allocated more memory than ones that run in the foreground. This was not what we had expected when we started working on this project.
* The second task followed mostly what we had expected. The child processes that were identical to the parents received the same amount of memory allocation that the parents got. This was not the case with children that called other processes using execv. Memory was allocated based on how much memory the called process needed.
* For the third task, we learned how to calculate system memory from inside a C program. There are many bash commands that provide a large amount of information about system conditions and running tasks. For example, vmstat and ps u will provide information about the vm conditions and the current running processes. However, there is no logical way inside of a program to call these, so we found the sysinfo() struct and command that returns the memory stats in the program. This allows us to track and alert the user of high memory usage.
* For task four, it was very similar to task three; however, instead of alerting the user, we killed tasks. We did find out that pthread\_kill() does not end a process, but rather alerts the process with a signal. Pthread\_cancel returned errors when canceling more than one, so we used pthread\_exit(0) to exit the running task if it caused the system to go over the threshold.