Name: Georges Junior Naddaf
Section 001 PA 1 Report

1- a) Address 1 addresses belong to local variables whereas Address 2 addresses belong to the dynamically allocated variables. This is because the stack usually occupies higher addresses compared to the heap and the stack holds local variables. When looking at the addresses, we notice that the stack grows downwards (Address 1 decreases) as it should on most systems and the heap grows upwards (Address 2 increases).

b) The size of the stack when waiting for the user input is 2132 KB.

c) The size of the heap when waiting for user input is 132 KB.

d) The address limits for the stack are 7fffffeea000-7fffffff000. The address limits for the heap are 00602000-00623000.

e) execve(): Executes and runs prog1.

mmap(): This function is mainly used to dynamically allocate variables to the heap in the virtual address space.

open(): Returns a file descriptor and is mostly used to reference libraries used by prog1.

access(): Checks user permissions to run the file.

fstat(): Returns information about the file, most probably used when it shows in the console that it's running the program (or if errors show up).

close(): Close file with specific file descriptor, used on opened libraries.

read(): Reads commands in executable.

mprotect(): Make sure that memory accesses follow the rules and do not cause any violations.

arch_prctl(): Unclear what the use of it is but it probably sets architecture-specific thread state for the virtual-address space.

munmap(): Similar to mmap() but takes different arguments.

write(): This allows the program to output to the command line (Address 1/2 text for example).

brk(): Sets the location of the program break so that the program does not unintentionally overwrite occupied addresses.

exit_group(): Terminates the program and exits all threads.

2- a) For the 32 bit executable:
(i) Total size of compiled code: 1691 Bytes
(ii) Size of code during run time: 4296 KB

(iii) Size of linked libraries: 124 + 4 + 4 + 1604 + 8 + 4 = 1748 KB

For the 64 bit executable:
(i) Total size of compiled code: 2183 Bytes
(ii) Size of code during run time: 7524 KB
(iii) Size of linked libraries: 128 + 4 + 4 + 1576 + 2048 + 16 + 8 = 3784 KB

b) The statement that caused the error is on line 6 : int x[300000]; The error showing up is a segfault meaning that the program simply cannot allocate more memory on the stack.

c) If we look at the stack limits, we notice that they are 7fffff48b000-7fffffff000. Then we look at the addresses that are output by the program and we see that the last non-dynamic variable allocated would go below the 7fffff48b000 limit (From 0x7fffff48bf20). And so, we simply get a segfault error because the stack does not have anymore space for more variables.

(d) We use stack info of two consecutive stack frame to determine the size of each frame. We get that the size is 0x124FB0 or 1200048 in decimal. This makes sense since we are allocating 300000 integers and each interger is 4 bytes. The stack is 12009472 bytes big (from stack limits). The recursive function should be called 10 (stack size/frame size) times before a segfault happens. This represents what is happening in our program.

(e) A frame usually contains local variables, return addresses and temporary parameters. The bulk of the data in the frames are local variables. Their size is 1200000 bytes. The 48 remaining bytes are present for the return address and the count temporary parameter.

3- (a) For the 32-bit executable:
(i) Total size of compiled code: 1984 B
(ii) Size of code during run time: 3196 KB
(iii) Size of linked libraries: 124 + 4 + 4 + 1604 + 8 + 4 + 160 + 4 + 4 = 1916 KB

For the 64-bit executable:
(i) Total size of compiled code: 2556 B
(ii) Size of code during run time: 7660 KB
(iii) Size of linked libraries: 128 + 4 + 4 + 1576 + 2048 + 16 + 8 + 524 + 2044 + 4 + 4 = 6360 KB

(b) Using valgrind, we can deduce that the statement causing this problem is at line 13: b = malloc(pow(10,r)*sizeof(int)); From /proc, we know that heap addresses should range from 00601000 to 00622000. However, the program and specifically this statement are exponentially mallocing more and more memory to the point where they go outside the heap (because of r incrementing) and into addresses related to libraries, which is a huge red flag.

(c) The difference between this error and the prog2 error is that this one is about the heap not finding enough memory to dynamically allocate and the prog2 error is about allocating too many local variables to the point where the stack runs out of memory.

Name: Georges Junior Naddaf
Section 001 PA 1 Report

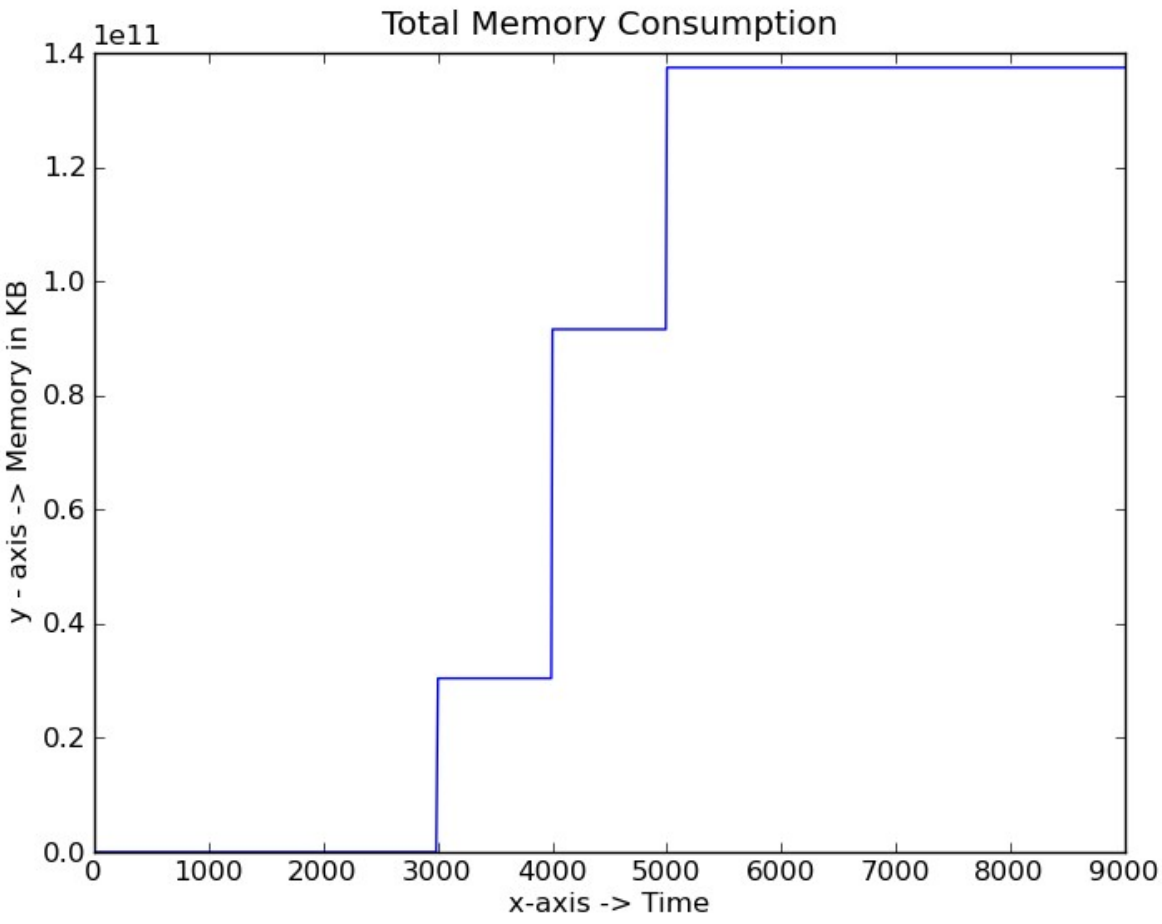
4- (a) Through valgrind, we notice that there were many more allocations (mallocs) compared to deallocations (frees). This is definitely the cause of the large amount of lost memory. To fix this, we simply make sure that for every malloc, there is a free that accompanies it and also make sure to not leave any hanging pointers by pointing them to NULL.

(b) The difference between system CPU time used and user CPU time used is how long the process stays in kernel or in user mode (making/not making system calls). A process's resident set size is the amount of memory that it acquired and is currently present in RAM, Signals received may occur because of a trap issued by the CPU or an interrupt caused by the user. A voluntary context switch is usually initiated by the thread itself if it wants to wait for a certain event to occur for example. An involuntary context switch is usually forced by the system scheduler upon the thread and changes the process control to another thread to allow fair scheduling and usage of the CPU.

5- a) The shell script deletes the data.txt output and the memory_utilization.png graph if they exist. That way, the new data that is outputted does not overwrite already present info.

b) The difference between my graphs and the ones included is that my graphs have a precision of 1s whereas the included graphs have a precision of 100ms. This explains why the included graphs are more precise than my graphs.

Prog51:



Name: Georges Junior Naddaf
Section 001

PA 1 Report

Prog52: