CMPSC 473: Project 1 Report

1) **Stack, Heap, and System Calls:**
    a. The second address is where the dynamically allocated variables are located and the first address is for the local variables. You can tell since the second address is at a lower position in the virtual memory stack, making it part of the heap. The heap is growing downwards in memory, meaning each new variable added to it has an address that is greater than the variable added before it. The stack is growing in an upward manner in memory, meaning that as more variables are added to it the address gets smaller.
    b. The size of the process stack is 2132kB.
    c. The size of the process heap is 132kB.
    d. The addresses for the stack lie between 0x7fffffdea000 - 0x7ffffffff000 with all of the local variables lying within those addresses. The heap has addresses 0x00602000 - 0x00623000 with the dynamically allocated variables lying within those addresses.
    e.
        i) The execve system call starts running the chosen program.
        ii) The brk system call sets the end of the address space for the data.
        iii) mmap creates a new mapping showing the address for the virtual memory space.
        iv) access checks if the process running has permissions to a file.
        v) open grants the process access to a file
        vi) stat returns the data from a file
        vii) fstat gets status information about the file specified by the file descriptor.
        viii) close is used to close an opened file
        ix) read returns the data within a file
        x) mprotect changes the protections of a memory mapping. Prevents memory from being accessed in a way it is not meant to be.
        xi) arch_prctl stores information on the current running thread in the FS register
        xii) munmap deletes a specific memory mapping
        xiii) write puts data into the chosen file
        xiv) exit_group closes down the process

2) **Debugging Refresher:**

a.

i) The size of the 32 bit executable is 1520 bytes, and the 64 bit executable is 1722 bytes. Ask for about decimal or bytes

ii) The size of the code during runtime is 9352K for the 32 bit executable and 11432K for the 64 bit executable.

iii) The size of the linked libraries for each executable is 1824 kbytes for the 32 bit and 3876 kbytes for the 64 bit.

b. The error for the 32 bit executable is when trying to dynamically allocate memory with malloc in line 11 of prog2.c. The error for the 64 bit executable is when trying to read the count variable when starting the allocate function on line 4 and then the signal is raised when the program reaches line 5. The variable is in a memory location that cannot be read from.

c. When checking /proc to see the limits of the stack and heap of the 64 bit program, we noticed that the count variable lies at 0x7FFFFF7FAFDC. The stack for the program starts at 0x7FFFFF9…. which is after that address. The heap also ends at the address 0x7FFFFF7B….. which is before the count variable. This means the count variable is outside of the stack and heap which is the cause of the error. The 32 bit executable had an error with available memory. The program was trying to allocate memory in the heap that was not available, causing a segmentation fault.

d. Each stack takes up 1200048 different address spaces in memory. There are about 7208960 address spaces in the whole program stack. In theory, this means the allocate function can iterate 6 times. When executing both the 32 and 64 bit executables, we get 7 instances of the allocate function iterating, where the 7th one produces an error.

e. The frame contains the data for the process running such as arguments. local variables, and the address of where to return to when finished. For the recursive allocate function, the frame contains the argument variable count, local variable x, the address of the dynamically allocated space in c, as well as the place to return to once it is finished (at the start of the allocate function).

3) More Debugging:

a.
   i) The 32 bit executable is 1747 bytes, and the 64 bit executable is 2017 bytes.
   ii) The 32 bit executable is 437552K while executing, and the 64 bit executable is 4413212K while executing.
   iii) The size of the libraries for each executable is 1824 kbytes for libc and 264 kbytes for the libm in the 32 bit version. For the 64 bit version we got 3876 k bytes for libc and 3080 kbytes for libm.

b. For the 32 bit version we find that malloc has a possible “fishy” value for its size parameter. This occurs on line 13. We also find that there is an invalid write of size 1 on line 18 which is the cause of the error. When checking the 32 bit program with /proc, we notice that some of the heap values are outside of the actual address space for the heap. For the 64 bit version the statement that causes the error is on line 18 which is the line that calls the memset function. We find that the error is an invalid write of size 1. The cause of the error is that memset is setting memory in the heap that is not within the range of the heap.

c. For program 2, the stack was filled up and could not hold all of the required variables. Program 3 did not have enough space in the heap to allocate the variables, therefore allocating them to a random memory address.

4) **And Some More:**

   a. The error occurring in prog4 is memory leaks. The function allocate is only freeing the allocated memory if it passes through the func functions from check.c. The function allocate1() also does not work as it will only free the first instance in the loop. allocate2() on the other hand, will always free the memory. The way to fix this issue would be to call allocate2() since both allocate() and allocate1() will not free all of the memory and allocate2() will free all of the memory that gets allocated.

   b.

      i) The user CPU time for prog 4 was on average 1.9 seconds

      ii) The system CPU time was on average 0.5 seconds. The difference between user and system CPU time is what is being measured. The user CPU time measures the time it takes the processor to run the code. The system CPU time is the time it takes for the Operating System kernel to run the code.

      iii) The maximum resident set size for the program was 1814756 bytes. The maximum resident set size is the maximum amount of physical memory that the program used while running.

      iv) We receive 4 signals. They are SIGSEGV, SIGBUS, SIGILL, SIGFPE. The process receives these signals from the operating system. The SIGSEGV is coming due to the program trying to write out of the memory space. SIGBUS occurs when a memory address is trying to be accesses that cannot be. SIGILL occurs when the program tries to read or write outside the allocated memory space. Finally SIGFPE occurs when there is a computational error in the program.

      v) The number of voluntary context switches have been between 1 and 3.

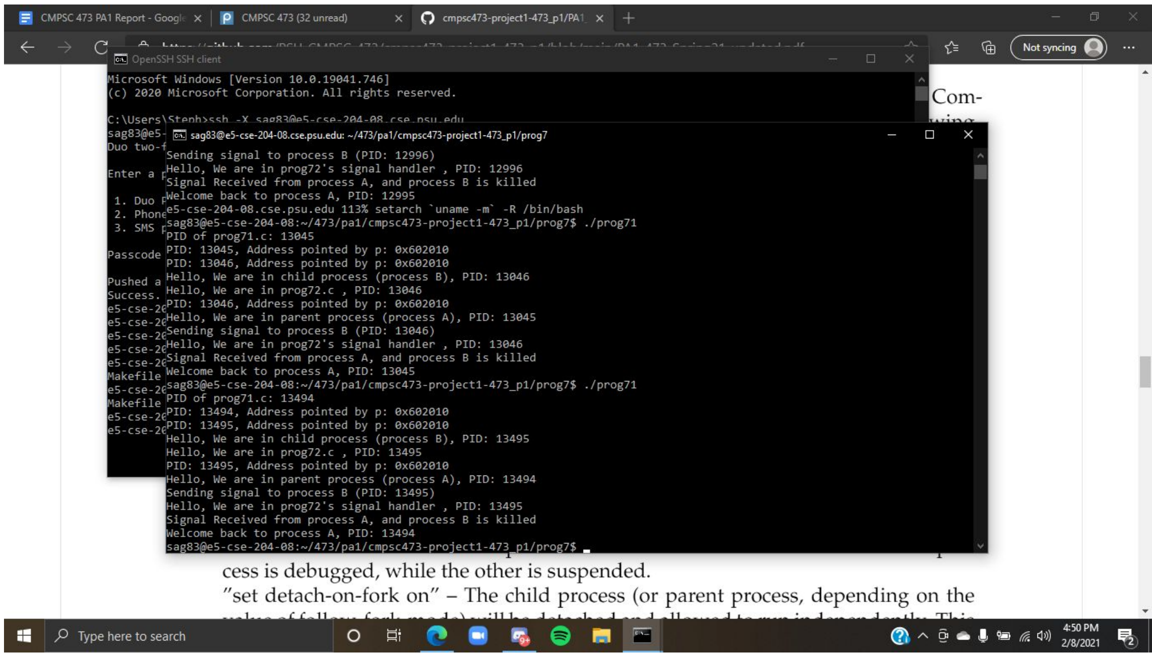
      vi) The number of involuntary context switches have been around 10. The difference between voluntary and involuntary context switches are that voluntary is when the process is completed and involuntary is when there are other processes waiting on the CPU.

5) Multi-Process Program that uses Fork, Exec, Wait, Kill, Exit Calls:

a.

i) The PID of the parent process is always 1 less than the child process (Parent = 13045 and Child = 13046)

ii) The address for both the parent and child process are the same at 0x602010


CMPSC 473 PA1 Report - Google
CMPSC 473 (32 unread)
cmpsc473-project1-473_p1/PA1
Not syncing
OpenSSH SSH client
sag83@e5-cse-204-08.cse.psu.edu: ~/473/pa1/cmpsc473-project1-473_p1/prog7
Type here to search
4:50 PM
2/8/2021

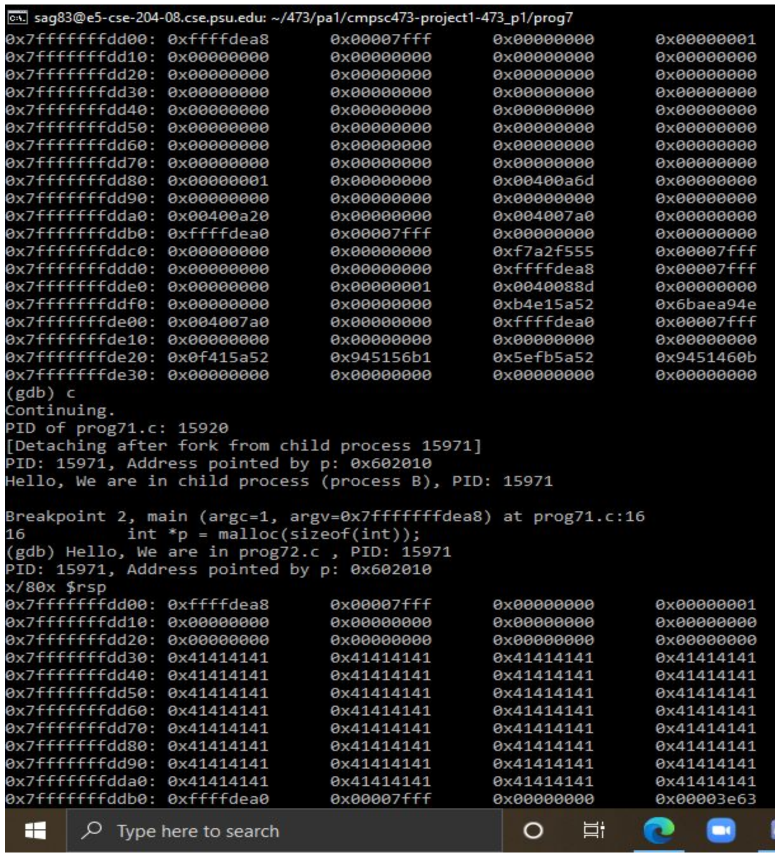
```
Microsoft Windows [Version 10.0.19041.746]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\Steph>ssh -X sag83@e5-cse-204-08.cse.psu.edu
```

```
Sending signal to process B (PID: 12996)
Hello, We are in prog72's signal handler , PID: 12996
Signal Received from process A, and process B is killed
Welcome back to process A, PID: 12995
e5-cse-204-08.cse.psu.edu 113% setarch `uname -m` -R /bin/bash
sag83@e5-cse-204-08:~/473/pa1/cmpsc473-project1-473_p1/prog7$ ./prog71
PID of prog71.c: 13045
PID: 13045, Address pointed by p: 0x602010
PID: 13046, Address pointed by p: 0x602010
Hello, We are in child process (process B), PID: 13046
Hello, We are in prog72.c , PID: 13046
PID: 13046, Address pointed by p: 0x602010
Hello, We are in parent process (process A), PID: 13045
Sending signal to process B (PID: 13046)
Hello, We are in prog72's signal handler , PID: 13046
Signal Received from process A, and process B is killed
Welcome back to process A, PID: 13045
sag83@e5-cse-204-08:~/473/pa1/cmpsc473-project1-473_p1/prog7$ ./prog71
PID of prog71.c: 13494
PID: 13494, Address pointed by p: 0x602010
PID: 13495, Address pointed by p: 0x602010
Hello, We are in child process (process B), PID: 13495
Hello, We are in prog72.c , PID: 13495
PID: 13495, Address pointed by p: 0x602010
Hello, We are in parent process (process A), PID: 13494
Sending signal to process B (PID: 13495)
Hello, We are in prog72's signal handler , PID: 13495
Signal Received from process A, and process B is killed
Welcome back to process A, PID: 13494
sag83@e5-cse-204-08:~/473/pa1/cmpsc473-project1-473_p1/prog7$
```

cess is debugged, while the other is suspended.
"set detach-on-fork on" – The child process (or parent process, depending on the

b. Before the exec command is called, the stack for both the parent and child process look like this with data at addresses ..ffdd30 to ...ffdda0 being 0:

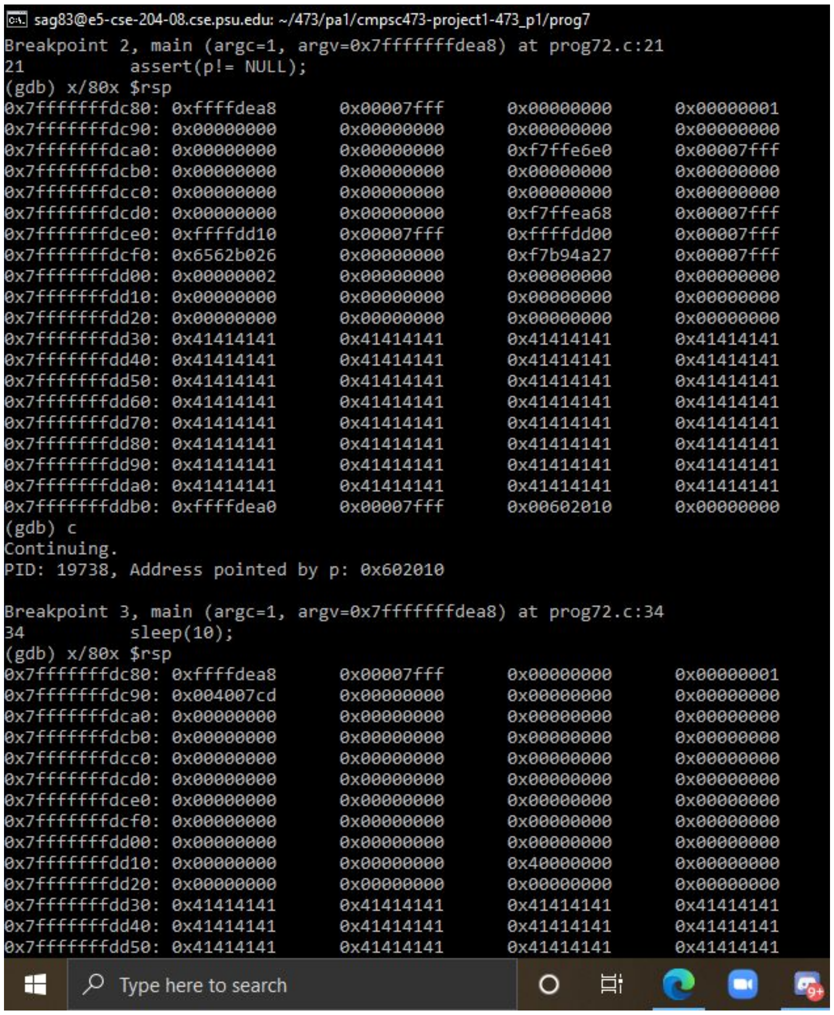


```
sag83@e5-cse-204-08.cse.psu.edu: ~/473/pa1/cmpsc473-project1-473_p1/prog7
0x7fffffffdd00: 0xffffdea8      0x00007fff      0x00000000      0x00000001
0x7fffffffdd10: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdd20: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdd30: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdd40: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdd50: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdd60: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdd70: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdd80: 0x00000001      0x00000000      0x00400a6d      0x00000000
0x7fffffffdd90: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdda0: 0x00400a20      0x00000000      0x004007a0      0x00000000
0x7fffffffddb0: 0xffffdea0      0x00007fff      0x00000000      0x00000000
0x7fffffffddc0: 0x00000000      0x00000000      0xf7a2f555      0x00007fff
0x7fffffffddd0: 0x00000000      0x00000000      0xffffdea8      0x00007fff
0x7fffffffdde0: 0x00000000      0x00000001      0x0040088d      0x00000000
0x7fffffffddf0: 0x00000000      0x00000000      0xb4e15a52      0x6baea94e
0x7fffffffde00: 0x004007a0      0x00000000      0xffffdea0      0x00007fff
0x7fffffffde10: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffde20: 0x0f415a52      0x945156b1      0x5efb5a52      0x9451460b
0x7fffffffde30: 0x00000000      0x00000000      0x00000000      0x00000000
(gdb) c
Continuing.
PID of prog71.c: 15920
[Detaching after fork from child process 15971]
PID: 15971, Address pointed by p: 0x602010
Hello, We are in child process (process B), PID: 15971

Breakpoint 2, main (argc=1, argv=0x7fffffffdea8) at prog71.c:16
16          int *p = malloc(sizeof(int));
(gdb) Hello, We are in prog72.c , PID: 15971
PID: 15971, Address pointed by p: 0x602010
x/80x $rsp
0x7fffffffdd00: 0xffffdea8      0x00007fff      0x00000000      0x00000001
0x7fffffffdd10: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdd20: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdd30: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdd40: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdd50: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdd60: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdd70: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdd80: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdd90: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdda0: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffddb0: 0xffffdea0      0x00007fff      0x00000000      0x00003e63
```

As you can see, after exec is called the data in the stack at addresses ...ffdd30 to ...ffdda0 are filled with 0x41414141.

c. When checking the stack of prog72 before and after the signal handling, we can see that the data in the address spaces ..ffdca0 through ..ffdd10 change. As you can see, after the signal handling has been done, data in that region of the stack is removed.



sag83@e5-cse-204-08.cse.psu.edu: ~/473/pa1/cmpsc473-project1-473_p1/prog7

```
Breakpoint 2, main (argc=1, argv=0x7fffffffdea8) at prog72.c:21
21          assert(p!= NULL);
(gdb) x/80x $rsp
0x7fffffffdc80: 0xffffdea8      0x00007fff      0x00000000      0x00000001
0x7fffffffdc90: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdca0: 0x00000000      0x00000000      0xf7ffe6e0      0x00007fff
0x7fffffffdcb0: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdcc0: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdcd0: 0x00000000      0x00000000      0xf7ffea68      0x00007fff
0x7fffffffdce0: 0xffffdd10      0x00007fff      0xffffdd00      0x00007fff
0x7fffffffdcf0: 0x6562b026      0x00000000      0xf7b94a27      0x00007fff
0x7fffffffdd00: 0x00000002      0x00000000      0x00000000      0x00000000
0x7fffffffdd10: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdd20: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdd30: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdd40: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdd50: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdd60: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdd70: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdd80: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdd90: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdda0: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffddb0: 0xffffdea0      0x00007fff      0x00602010      0x00000000
(gdb) c
Continuing.
PID: 19738, Address pointed by p: 0x602010

Breakpoint 3, main (argc=1, argv=0x7fffffffdea8) at prog72.c:34
34          sleep(10);
(gdb) x/80x $rsp
0x7fffffffdc80: 0xffffdea8      0x00007fff      0x00000000      0x00000001
0x7fffffffdc90: 0x004007cd      0x00000000      0x00000000      0x00000000
0x7fffffffdca0: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdcb0: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdcc0: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdcd0: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdce0: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdcf0: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdd00: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdd10: 0x00000000      0x00000000      0x40000000      0x00000000
0x7fffffffdd20: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdd30: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdd40: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffdd50: 0x41414141      0x41414141      0x41414141      0x41414141
```