MemoryLab – Answer Document

Program memory\_layout.c

1. Study, build, and run memory\_layout.c to improve your understanding of the four memory sections of a program. Copy/paste your running here.
2. What are the memory sections of a running program? Describe the contents of each section. Draw a diagram of the sections.
3. Compare the memory sections of a running program to the program segments on disk.
4. The program on disk does not have segments for stack and heap. Why?
5. Modify memory\_layout.c to experiment with the protections on the sections. There is some code commented out that is an experiment, attempting to write to memory in front of the global section.  
       //int \*p = &global - 400;

    //\*p = 5;

You can perform a similar experiment for heap.   
 Another experiment is to add a pointer to constant, and attempt to modify it. What does   
 the compiler do with this pointer?  
          //int \*p = &constant;

    //\*p = 5;

Copy/paste your running here.

# Program heap.c

1. Study, build, and run heap.c to improve your understanding of the heap and C’s API to access heap memory. Copy/paste your running here.
2. Describe the three C API heap access functions used in heap.c?
3. When you run heap, can you still access heap memory that you have returned?
4. When you return heap memory, followed by allocating the same size returned, what memory address is returned?
5. Can you access memory that is in front of the address returned by malloc/calloc?

# Program stack.c

1. Study, build, and run stack.c to improve your understanding of the stack. Copy/paste your running here.
2. stack.c executes the maps command with the following code. maps shows the memory layout of a process.  
    char command[50];  
    printf("\n\n /proc/%ld/maps \n\n", pid);  
    sprintf(command, "cat /proc/%ld/maps", pid);

system(command);

1. What does the following tricky C code accomplish with back: and &&back.  
      back:

    printf(" func1: retaddr %p \n", &&back);

1. Discuss (a) how a stack is implemented, (b) how the memory layout of a stack relates to the heap, globals, and code, and (c) what a stack overflow is. Which tool creates the code that implements a stack? Can you write assembly code functions that carve out a stack? Provide rationale.

# Virtual Memory - Paging

1. Study this algorithm until you understand it. No answer required.
2. Does each process have its own page table? Why/Why not?
3. In this example, what is the length in bytes of a page table?
4. What is a technique that helps reduce the size of page tables?
5. A computer system will not be efficient if every memory access requires two accesses. Describe the technique that is used that allows paging to be efficient.
6. Discuss how creating virtual memory with protections helps computer security.
7. **Option 1**: Design a page table data structure and write a program that implements the above algorithm.

# Heap Memory Management - malloc and free

1. Study, build, and run xv6\_malloc.c and kr\_malloc.c. Both programs are rather terse and require quiet time to study. Copy/paste your running here.
2. Draw a diagram that shows the data structure used to implement the heap.
3. What is the sbrk function used by the algorithms?
4. What is a more modern version of sbrk?
5. There is a function named morecore. What is the origin of the name morecore?
6. The kr\_malloc.c has some interesting C code. What does the following do?

unsigned size;

1. **Option 2**: Design and implement your own version of malloc and free and write a program that uses them.