1. What is a program?

2. Identify at least three stages that a program goes through in-between (and exclusive of) editing source code and execution.

3. Why does an executable file not have a stack or heap?

Consider the three command lines:

% gcc my\_prog.c

% ./a.out < in1 > out1 & ./a.out < in2 > out2

% ./a.out < in3 | ./a.out > out3

4. In response to the second command line, are there one or two programs running? Are there one or two processes running?

5. In response to the third command line, are there one or two programs running? Are there one or two processes running?

6. What is the difference between the second and third command lines?

Kernel mode / User mode. Circle **one or both** of K and U, as applies.

7. K / U In this mode add instructions can be executed.

8. K / U In this mode only a subset of the instructions can be executed.

9. K / U In this mode only a subset of the physical memory addresses can be accessed.

10. What should happen if a user program attempts to execute a privileged instruction?

11. Consider the changes in the memory image of a process during a buffer overflow attack, as illustrated below. Identify a way in which the attack can be prevented using a memory protection scheme that differentiates instruction fetches from data reads and writes. (diagram source: stackexchange)

Buffer overflow example in stages.
Stage a: stack layout before function call.
Stage b: stack layout after function call with return address to main program. There is a small buffer allocated as a local variable for the function.
Stage c: stack layout after buffer overflow, where the return address has been replaced with a pointer to code that has been read into the buffer.

execution starts in main() main() calls A() buffer overflow after a call to strcpy()

returns to A(); the return address to

main() has been overwritten