The following test is *closed book*; you may neither give nor receive assistance during this exam. Do not discuss the content of this exam until it is turned back in class. Read each question *carefully* and answer it in the space provided. If you require more room, continue on the back of the page after clearly labeling what question the answer goes with.

Name: _____

1. Explain *limited, direct execution*. What, exactly, is **limited** and how is it limited? What, exactly is **direct** about it? (10)

2. Process virtualization is a spectrum from full-on emulation to unlimited, direct execution.
   (a) Describe two *advantages* of unlimited, direct execution of arbitrary user code. (5)

   (b) Describe two *advantages* of full-on emulation or running processes in a fully virutalized machine. (5)

3. What hardware feature(s) is(are) necessary to support *preemptive scheduling*? (5)

4. Why is hardware like the `k-stack` necessary in an operating system that has interrupts that can happen at any time while a process is running? (Alternatively: What does the `k-stack` actually *do*?) (10)

5. Is it possible to have multiple processes running the same program? In terms of operating system data structures, how could that work? (5)

6. Draw a state diagram for a process as we have pictured it in the operating system. Clearly label the events that trigger the state transitions. (5)

Points earned: _____
out of a possible 45 points

7. Consider FIFO, SJF, and STCF scheduling. Which, if any, require the operating system to be able to *preempt* the running process? (5)

8. Dr. Ladd said something to the effect that SJF is the *worst of both worlds* between FIFO and STCF scheduling. Considering the worst-case work loads for each scheduling policy, **support** or **refute** his statement. (5)

9. Imagine a program that spins up a child process and then loops forever while the child process spins up a "grandchild" process. The parent process loops forever after creating the child; the first child prints " Potsdam" after starting its child and then terminates; the grandchild prints "SUNY" and then terminates. (10)

   Write the `main` function (you write **only** one function) that runs as described above: start a child and loop, child starts a child, prints, and terminates, and the second generation child prints a string and terminates. For sake of clarity/concision, you do not have to handle errors.

   The three processes are summarized below:

Parent	Child P	Child S
`start child P`	`start child S`	`print "SUNY"`
`loop forever`	`print " Potsdam\n"`	

10. Consider `execl`, which takes a variable number of C-style strings as parameters as in the following:

Points earned: _____
out of a possible 20 points

```
execl("/bin/man", "/bin/man", "-k", "wait", nullptr);
```

(a) What happens to the process that executes the above line? In terms of the operating system, PIDs, and context, what happens? (5)

(b) If you were at the shell prompt and wanted to execute the same program with the same parameters, what would you type? (5)

(c) Assuming `/bin/man` is written in C/C++, where would its `main` find the parameters passed by the call to `execl`? (5)

11. What part of an address is used as in index into the **page table**? What part of an address is found inside a **page table entry**? Clearly answer both questions for full credit. (5)

12. Assume 100 copies of `/bin/man` are running in our operating system simultaneously. How many **page tables** do those processes (and only those processes) use? Explain how you arrived at your answer. (5)

13. What does it *mean* when a process translates a virtual address with a **page table entry** with a 0 **present bit**? (5)

14. How can multiple processes in a *paged* memory system **share** code pages? What must the memory system do in the page tables of those processes? What (if any) protective measures must the OS take in this situation? (5)

Points earned: _____
out of a possible 35 points