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)

   Solution: *OSTEP Chapter 6, Process Lecture and Slides, Scheduling Worksheet Question 9*

   Limited, direct execution is a compromise between letting user programs run any arbitrary code without any operating system intervention and requiring operating system intervention for every instruction.

   The running of those instructions designated as *unprivileged* by the CPU is **direct** in that a user program fetches and executes such instructions on the CPU at CPU speeds. Unprivileged instructions run directly on the CPU.

   The user program is **limited** to those instructions that are unprivileged. Taking the action of a privileged instruction requires a system call or some other type of system interrupt so that OS code can grant or deny permission for the action.

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)

   Solution: *OSTEP Chapter 6, Processes Lecture*
   Primary advantage: speed. No system call to run privileged instructions. Also: user program can do *anything* making the program easier to write.

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

   Solution: *Scheduling Worksheet Question 10, Processes Lecture*
   Process safety; no process instruction is run without being checked by the virtual machine. Emulation of different hardware, ability to debug while running, ability to snapshot the VM.

Points earned: ______
out of a possible 20 points

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

> **Solution:** *Scheduling Worksheet Question 4*
>
> Privilege bit (permits user/system execution separation). Kernel-mode register saving (the `kstack` or equivalent; saves and restores user mode registers intact). Interrupts into kernel space and, in particular, timer interrupts (to permit changing what user program is executing at the end of a quantum).
>
> 1 for one mechanism; 2 each for next mechanisms. -2 for any incorrect mechanism (including *registers*, *context*).

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)

> **Solution:** *OSTEP Chapter 6, Processes Lecture and Slides, Slide 59ff in particular*
>
> When user process A is interrupted, *all* of its registers must be saved **by the hardware**. This is because the operating system is just machine code. For the OS to service the interrupt the operating system code must *execute* and to execute it must use registers.
>
> The `k-stack` (or kernel-only copies of registers) is where the hardware moves A's registers before the OS interrupt handler runs. The OS can then copy and/or replace the contents of the kernel stack to handle a context switch or it can service the interrupt without disturbing the kernel stack and return control to A when the OS is done.

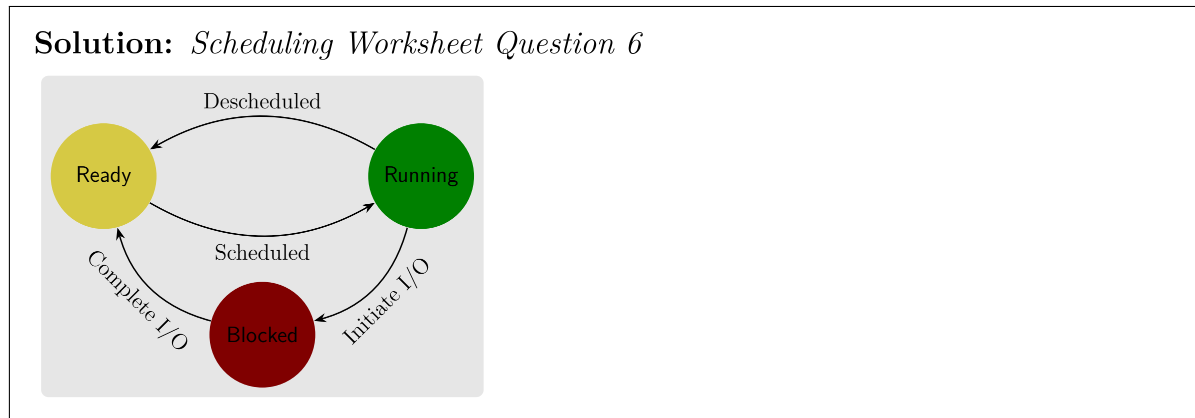
5. Is it possible to have multiple processes running the same program? In terms of operating system data structures, how could that work? (5)

> **Solution:** *Processes Lecture*
>
> Of course. Each process running the same program has its own *context* so each could be executing at a different point in the code. This means each has its own register values, memory, etc. Thus each has its own *process control block* and, since PCB are found through the process table, each gets its own PID, too.

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 25 points

Solution: *Scheduling Worksheet Question 6*



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

Solution: *OSTEP Chapter 7, pp67-67*

Shortest time to completion requires interrupting a long-running process that has only just begun when a new, shorter process arrives. This is preemptive in that the running process does not cooperate with the OS in this policy decision.

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)

Solution: *OSTEP Chapter 7*

He seems correct. SJF behaves as FIFO in the face of a very long job arriving first; since it is not preemptive, the convoy effect happens to all jobs (including short ones) that arrive after the long job.

If one or more short jobs arrives before a longer-running job, if shorter jobs arrive at a rate of about 1 per time of a short runtime, the longer-running job will be starved. This is similar to STCF starving a long job, just without the preemption used in that scheduler.

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

Points earned: ______
out of a possible 20 points

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"`	

Solution: *OSTEP Chapter 5, Scheduling Worksheet Question 5, VSSH-b Program*

Creating two new processes: means there are two calls to `fork()`: once in parent's context to create process P; once in process P's context to create process S.

```
int main() {
  if (fork() != 0) {    // parent of fork()_1; parent
    while (true);       // loop forever
  } else {              // child of fork()_1; child P
    if (fork() != 0) {  // parent of fork()_2; child P
      cout << " Potsdam\n";
    } else {            // child of fork()_2; child S
      cout << "SUNY";
    }
  }
}
```

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

```
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)

Solution: *OSTEP Chapter 5, VSSH-b Program*

The memory content of the process is replaced with that in the file `/bin/man`. The PID remains the same and the context is reset to that of a newly started program.

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

Solution: *OSTEP Chapter 5, VSSH-b Program*

`/bin/man -k wait`

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

Solution: *C/C++ Programming, Sample Code*

In `argv[0-2]`

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)

Solution: *OSTEP Chapter 18, p180-181, Paging & TLB Slides, Paging Worksheet, PTE Size and PT Size Questions*

The *page table* is an array of *page table entries*. The array is **indexed** by the virtual address's *page number*; each PTE contains the physicial *frame number* corresponding to the page number that indexes it (along with some meta data).

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)

Solution: *OSTEP Chapter 18, Paging Slides*

Every process has, as part of its *context*, a *page table*. Thus each of the processes running `/bin/man` has its own page table; 100 page tables for 100 such processes.

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

Solution: *OSTEP Chapter 18, p184, Page Table Lecture*

The *page* containing the address is not currently mapped into physical memory; there is no *frame* in physical memory containing the page. The OS must take a *page fault interrupt* and **load the page** from disk, put the page into a frame, update the process's page table to reflect the presence of the page, and then restart the instruction.

Points earned: ______
out of a possible 20 points

Note that the I/O is slow so the process is **blocked** until the page load completes (the OS moves it from **blocked** to **ready**) after updating the page table. The "instruction is restarted" automatically when the process is scheduled: the PC was not advanced when the page fault occurred so the same instruction is fetched and executed.

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)

Solution: *Paging Lecture*

Corresponding PTE contain the same *frame number*. Thus addresses into the corresponding pages address the same physical RAM. The shared code pages should be *read-only* since code sharing processes are not meant to communicate through the shared RAM.

Points earned: ______
out of a possible 5 points