Instructions:

- Print your name in the space provided below.
- This examination is closed book and closed notes. No calculators or other computing devices may be used.
- Answer each question in the space provided. If you need to continue an answer onto the back of a page, clearly indicate that and label the continuation with the question number.
- If you want partial credit, justify your answers, even when justification is not explicitly required.
- There are 12 questions, priced as marked. The maximum score is 100.
- When you have completed the test, sign the pledge at the bottom of this page and turn in the test.
- Note that either failing to return this test, or discussing its content with a student who has not taken it is a violation of the Honor Code.

Do not start the test until instructed to do so!

Name ______Solution______________________________
printed

Pledge: On my honor, I have neither given nor received unauthorized aid on this examination.

____
signed

1. [8 points] Suppose that when a process requests an I/O operation, the process is moved to a system state reserved for such processes. Would it make sense to use a pure FIFO protocol to manage the processes in that state? Explain why or why not.

   No. It is certainly possible that I/O requests for two or more processes may be carried out concurrently, especially if they involve different hardware devices. It is also the case that the time required to service I/O requests varies widely. So, there is no reason to believe that I/O requests will be completed in the same order in which processes make those requests. Given that, using a FIFO strategy would impose unnecessary waits on some processes that were unfortunate enough to be caught behind processes with I/O requests that take a long time to complete.

---

2. [8 points] Suppose that a process is executing on a multitasking system, and that the current process instruction requires modifying the value of a variable that is stored in primary memory, but not in a data register in the CPU. Suppose that the scheduler is implemented so that a context switch is triggered every time a process behaves in this manner. Describe the implications of that design decision.

   Typical processes spend a great deal of time modifying the values of variables, and it is common for a relatively large number of variables to be in scope at any time during process execution. Therefore, the scenario described above is highly likely to occur. If a context switch was required every time, there would be a number of nasty consequences:

   - **There would be far more context switches, increasing the scheduling overhead and decreasing performance.**
   - **Since memory accesses are very fast, this situation is not at all similar to the handling of a disk I/O request, in which case blocking the requesting process is justified.**
   - **Performing a context switch involves backing up the values in a significant number of data and status registers. That will surely take more time than is required to simply load a variable into a register from primary memory.**
   - **Unless s somewhat careful strategy is adopted, the required variable will still not be in a register after the process is suspended and again reaches the running state.**

---

3. [8 points] Suppose that the processor in a system uses D data registers and S status registers to execute user processes. Assume that each register contains B bytes, and the time required to store a single byte from register to physical memory is W. Derive an algebraic formula for the total time required to back up the execution context of the current process.

   Time = $(D + S) * B * W$

4. [8 points] Assuming the processor for the system described in the previous question does not provide any duplicate hardware that might reduce the overhead, when a context switch from one user process to a second user process occurs, how many execution contexts must be copied (either into or out of the CPU registers)? Justify your answer.

   The execution context for the first use process must be copied out, and the execution context for the second user process must be copied in.

   However, the scheduler must execute in order to select the next process to run, and so its execution context must be loaded and unloaded as well.

   So, a total of 4 such operations are required.

5. [6 points] What kind of hardware support could be provided to reduce the cost of performing a context switch?

   Several ideas were presented in class and/or in Nutt:
   - **provide two (or more) complete sets of data and status registers so that loading and unloading can be carried out in parallel**
   - **embed the scheduler code in hardware, eliminating the need to load/unload it**
   - **use a second processor, dedicated to executing only kernel mode code**

6. [8 points] Suppose that a system uses batch scheduling, scheduling processes on the CPU in the order they arrive, and running each process to completion before scheduling the next on the CPU. Assume that on average new processes arrive at a rate of 3 per minute, and that the average execution time (time in the CPU) for each process is 15 seconds. Assume that the scheduling overhead uses 10% of the total available CPU time. On average, what percentage of the time will the CPU be idle? Justify your conclusion.

   In each 60 seconds of system time, an average of 45 seconds will be used to run three processes and 6 seconds (10% of the available minute) will be used for scheduling overhead. If we disregard other possible time costs, about which no information is given anyway, that leaves 9 seconds of idle time for the CPU.

   So, the CPU will be idle 9/60 or 15% of the time, on average.

7. [6 points] Describe two ways in which the processor may discover that a device has sent an interrupt.

   The processor can poll the hardware devices in the system, checking each to see if it is in an interrupt state.

   Each device can be connected to a dedicated interrupt flag (or register of flags) in the CPU, and simply turn on the interrupt flag (or its particular interrupt register bit) when the device generates an interrupt. Checking of the interrupt flag can then be embedded efficiently into the fetch-execute cycle.

8. [8 points] Suppose that a multitasking scheduler uses a preemptive scheme with an interval (quantum) timer. Suppose that the time quantum (time between interval timer interrupts) has been set to an relatively large number (say several minutes). In answering the following questions, consider issues like turnaround time and responsiveness.

What is the effect of this on the performance of CPU-bound processes?

On the one hand, all processes will see an increase in the average delay before entering the run state.

On the other hand, CPU-bound processes will then be likely to use their entire, large quantum before being interrupted by the quantum timer, and so CPU-bound processes are very likely to run to completion in less total time.

What about interactive users?

Interactive users care most about IO-bound processes (most GUI interactions are such). IO-bound processes will usually not benefit at all from an increased quantum since they will usually generate an IO request long before the quantum is over. On the other hand, IO-bound processes will certainly suffer from the increased delays due to CPU-bound processes retaining the processor for longer times.

---

9. [12 points] Consider the simple hold/ready/run/block state transition scheme for process scheduling. Draw the state diagram, showing all the possible transitions. Label each transition with a brief, precise description of an event that might cause a process to make that transition.

The answer to this question is in the specification for Project 2.

10. [8 points] Suppose a multitasking system loads the entire executable image of each process into physical memory as soon as the process becomes eligible to run, and keeps it there until the process terminates. Is it practical for the linker to generate addresses that will not require re-mapping, either by a loader or at runtime? Explain why or why not.

As described, it is clear that process images will never be relocated after they are loaded, so that is not an issue.

Since the system uses multitasking, we must expect that several process images will be in physical memory at the same time. Therefore, process images may be loaded at any base address. So, if the linker simply generates absolute addresses, those will be useless without re-mapping at load or runtime. On the other hand, there is no way for the linker to know what base address will be selected at load time, so the linker cannot generate the correct physical addresses.

There is one possibility: what if the linker specifies a base address and the loader is required to load the process image only at that location? Technically, this could work. However, it would mean that each process image could only be loaded at one, fixed base address. That would drastically increase the wait times for almost all processes, since the loader would have to wait until an appropriately large free space is available starting at the right base address. While this is feasible, it is not practical.

---

11. Recall that shared hardware and software resources may be space-multiplexed or time-multiplexed.

a) [6 points] Explain the difference between the two types of multiplexing.

Space multiplexing involves allowing two or more processes to use different parts of the same resource at the same time.

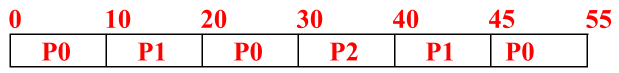
Time multiplexing involves allowing two or more processes to use the same part of (or all of) the same resource at different times.

b) [4 points] Is it possible for a hardware resource to be both space-multiplexed and time-multiplexed? If yes, give an example.

Yes. One example would be a file on a storage device. Different processes can safely use the entire file at different times, and different processes could safely use different parts of it at the same time.

12. [10 points] Suppose a system uses round-robin processor scheduling, with a quantum of 10. The processes shown below arrive at the specified times and require the specified service times. Draw a Gantt diagram to show how these processes will be scheduled, and fill the specified statistics.

Process	Arrival time	Service time	Finish time	Turnaround
P0	0	30	55	55
P1	5	15	45	40
P2	15	10	40	25



Note: when P2 arrives, at time 15, P0 has already moved from the Running state back to the Ready state, so P0 is ahead of P2 and P0 is selected to run when the quantum expiration occurs at time 20.