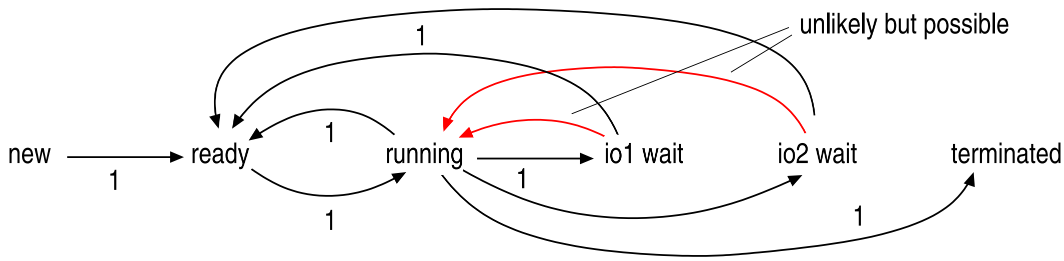
4 problems. 40 points. 30 minutes *Closed book. Closed notes. No electronic device.* *Write your name above.*

1. [6 points] An OS has 1 cpu, 2 io devices (io1, io2), pre-emptive cpu scheduling, and no multi-threaded processes. A process is terminated only by itself. The possible states of a process are given below. Draw the possible transitions (and omit the impossible ones).

new	ready	running	io1 wait	io2 wait	terminated

Solution **[6 pt]**



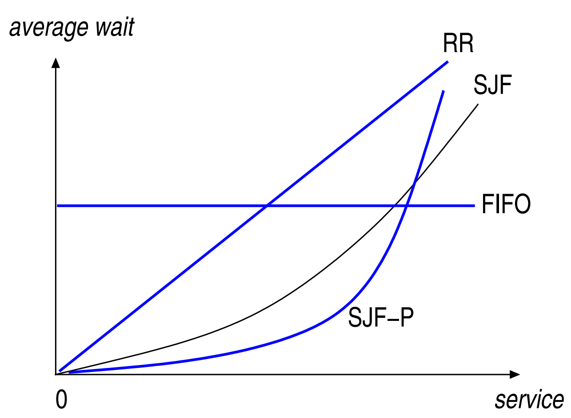
Points as shown above.

Because a process is terminated only by itself, there are no transitions to terminated from new, ready or io wait. -1 pt for each such transition.

2. [6 points] A collection of cpu-bound processes are scheduled on a cpu. The curve in the graph below shows the average wait vs service for SJF (shortest-job first, non-preemptive) scheduling.
(Recall: the *service* of a process is the total cpu time it requires; the *wait* of a process is the total time it spends in the ready queue; the *average wait* for service s is the average wait of all processes with service s.)

Draw on the same graph the expected curve for FIFO (instead of SJF). Repeat for SJF-preemptive. Repeat for RR (round robin). (So your answer is three curves on the same graph.)

Solution **[6 pt]**



2 pt for each curve.
1 pt if the curve is wrong but non-decreasing.

3. [12 points] A multi-cpu shared-memory machine has a swap instruction (and no other "read-modify-write" instructions). Specifically, `swap(x,y)` atomically exchanges the contents of register `x` and memory location `y`.

Implement a (weak or strong) spin lock using the swap instruction. Specifically, give code chunks (at a level of detail as in the os-process slides) for

- lock definition
- lock `acq()`
- lock `rel()`

***Solution* [6 pt]**

`swap(x,y)`, with `x` true, has the same effect as `test&set(y)`. So the solution is almost identical to a test-and-set solution.

Here is a weak lock.

- `Lock lck:` [3 pt]
```
acqd ← false
```

- `lck.acq():` [6 pt]
```
register tmp ← true
while (tmp)
  swap(tmp,acqd)
return
```

- `lck.rel():` [3 pt]
```
acqd ← false
return
```

Max 6 pt for a solution that uses the test-and-set instruction. Less if solution is not correct.

Max 5 pt for a solution that uses a pcb queue. Less if solution is not correct.

Several of you gave a solution that uses test-and-set but implemented the latter using the swap instruction. This is fine if your implementation is correct. Usually, it was wrong: the test-and-set function was not atomic. This got max 6 pts.

4. [16 points] You are given a multi-cpu machine with spin locks. Give an *efficient* implementation for a lock whose acquired durations can be long (e.g., seconds or minutes). Specifically, give code chunks (at a level of detail as in the os-process slides) for

- lock definition
- lock `acq()`
- lock `rel()`

***Solution* [6 pt]**

Because the lock can be acquired for long durations, the solution must use a pcb queue and a spin lock to protect the queue. So the answer is the one titled "Lock: spin, pcb, multi-cpu" in the os-process-slides.

- `Lock lck:` [5 pt]
```
boolean lckAcqd              [2 pt]
spinlock lckSplock           [2 pt]
PcbQueue lckQueue            [2 pt]
```

- `lck.acq():` [6 pt]
```
lckSplock.acq()
if (not lckAcqd)             [2 pt]
   ...
   return
else                         [4 pt]
   rrSplock.acq()
   ...
   scheduler()
```

- `lck.rel():` [4 pt]
```
lckSplock.acq()
if (lckQueue empty)          [2 pt]
   ...
else                         [2 pt]
   ...
```

Max 8 pt for a busy-waiting solution

Max 8 pt for not using a spin lock.

Max 8 pt if lock acquire never blocks.