1. (+25 points) To implement deadlock avoidance, the Banker's Algorithm is applied to the following system, where the total number of resources available for each of four resource types is R0=6, R1=4, R2=4, and R3=2. Is the system in a safe state? If not, then explain why not, or if so, find a safe sequence. Show your work.

Maximum Claims:

	R0	R1	R2	R3
P0	3	2	1	1
P1	1	2	0	2
P2	1	1	2	0
P3	3	2	1	0
P4	2	1	0	1

Current Allocation:

	R0	R1	R2	R3
P0	2	0	1	0
P1	1	1	0	0
P2	1	1	0	0
P3	1	0	1	0
P4	0	1	0	1

First, we compute the current needs and currently available resources:

Current needs:

	R0	R1	R2	R3
P0	1	2	0	1
P1	0	1	0	2
P2	0	0	2	0
P3	2	2	0	0
P4	2	0	0	0

Currently Available resources.

R0	R1	R2	R3
1	1	2	1

This is safe in the following sequence:

(after allocating to)	R0	R1	R2	R3
P2	2	2	2	1
P3	3	2	3	1
P4	3	3	3	2
P1	4	4	3	2
P0	6	4	4	2

2. (+25 points) Suppose we have 5 dining philosophers and 5 shared chopsticks, and that the philosophers are circularly deadlocked. Assume also that there are two serving spoons, one allocated to philosopher 1, and the other available. Assume that philosopher 4 wants a serving spoon. Show how this situation can be modeled according to the various data structures, e.g. Allocation and Request matrices, used in the Deadlock detection algorithm. Run the Deadlock detection algorithm on your model to prove that the dining philosophers are in deadlock.

Current allocation:

	Chopstick	Spoon
DP1	1	1
DP2	1	0
DP3	1	0
DP4	1	0
DP5	1	0

Max allocation:

	Chopstick	Spoon
DP1	2	1
DP2	2	1
DP3	2	1
DP4	2	1
DP5	2	1

Since P4 wants a serving spoon then our current needs are:

	Chopstick	Spoon
DP1	1	0
DP2	1	0
DP3	1	0
DP4	1	1
DP5	1	0

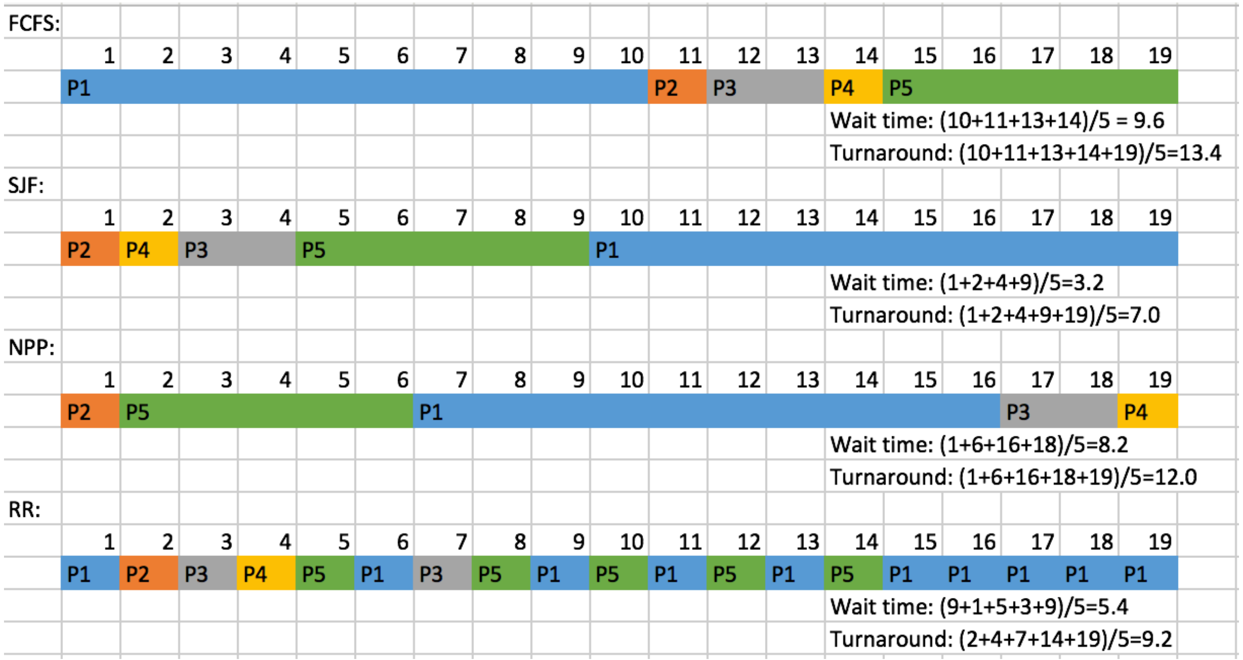
We can see that Spoon is already available but we have no chopsticks available:
Chopsticks: 0
Spoons: 1
So all of the DP require an unavailable chopstick! Obviously, this commonly known problem is deadlocked since none of the philosophers can get the additional chopstick required. To solve this deadlock is trivial as we've seen in the previous homework.

3. (+20 points) Consider the following set of processes, with the length of the CPU execution time given in seconds:

Process	Execution Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

Draw four Gantt charts that illustrate the timeline of execution of the processes when scheduled according to FCFS, SJF, non-preemptive priority (priority 1 is most important), and round robin (time slice=1). Which one has the lowest average wait time? Which one has the lowest average turnaround time?

Looks awfully like that exam problem!



FCFS:
Wait time: (10+11+13+14)/5 = 9.6
Turnaround: (10+11+13+14+19)/5=13.4

SJF:
Wait time: (1+2+4+9)/5=3.2
Turnaround: (1+2+4+9+19)/5=7.0

NPP:
Wait time: (1+6+16+18)/5=8.2
Turnaround: (1+6+16+18+19)/5=12.0

RR:
Wait time: (9+1+5+3+9)/5=5.4
Turnaround: (2+4+7+14+19)/5=9.2

Lowest avg. wait time: SJF

Lowest avg. turnaround time: SJF

4. (+30 points) Suppose the Completely Fair Scheduler (CFS) algorithm is applied to the following scenario. There exist two processes P1 and P2 that need to be scheduled. P1 has two threads T1 and T2 that are implemented as kernel threads. P2 has three threads T3, T4, and T5 that are implemented as user space threads. We apply round robin scheduling to all schedulable tasks, where each task schedulable by the kernel receives a time slice of size T seconds.

a. (+20) Draw the table of wait times (times owed on an ideal CPU) for one round robin of this scenario, i.e. what are the wait time balances as each time slice is allocated to a kernel-schedulable task, and what is the final wait time for each such task? See for example slide 21 of the Chapter 6.3 lecture slides.

Wait times:

	T1	T2	P2
T1	-2T/3	T/3	T/3
T2	-T/3	-T/3	2T/3
P2	0	0	0

(Adapted from slide 21 on lecture 6.3)

b. (+10) Is this system fair at the level of schedulable tasks? Is this system fair at the level of processes? Justify your answers.

After one round of round robin, the balances owed are all 0 so every task receives its fair share of CPU time. However, at the level of processes it's not fair since P2 needs to share its resources across 3 threads (in the user space) while T1 and T2 are kernel threads who don't have to share.