Michael North
Lab 2 Findings

3.1: Running the part one specs, I observed the following output:
PPPABCBCAABCABCBCAABCABCBCABCAABC
This is due to the fact that the main process, which has higher priority, will be able to create all three processes (and print all three P's) until it voluntarily gives up control to the lower priority processes.

3.2: With the main process running at a lower priority than it's children, we get this output:
PAAAAAAAAAAPBBBBBBBBBBPCCCCCCCCCC
This is because in the static priority queue, the lower priority process (main) will never regain control until all higher priority processes have voluntarily given up control. In this case, that means that each spawned printloop completely locks out main until it has run it's course, despite using the CPU for many quantums.

3.3: PPPCCCCCCCCCCBBBBBBBBBBAAAAAAAAAA
This occurs because, in a static priority scheduling system, the highest priority processes will always execute until they voluntarily give up control or return. In this case, this means that main runs it's course, then the next highest priority – C, then B, then A.

When we change the main process' priority to 6, we get:
PAPBBBBBBBBBBPCCCCCCCCCCAAAAAAAAA
This is, again, an expected artifact of static priority scheduling. Because main and A are tied, they will alternate for control in round-robin fashion, until main creates the higher priority process B, which will run until completion, and C, which will again run until completion. At that point control would again alternate between main and A, but main is done and has voluntarily given up control, so we see A finally finish uninterrupted.

3.4: With null process priority set to INITPRIO:
PPP
This is because the null process has a higher priority than any of the spawned child proccesses, and thus they will never be reached.

With the null process priority set to 30:
Nothing happens past initialization. With the null process set higher than the main process, the main process cannot gain control.

3.5: With sleepms(1) and null process printing 'N':
Mostly N, printed for dozens of lines, but there are some interesting parts:
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
PPPAB#C[1;31m-C-B-A-------CA-B-------B-A-C-------BC-A--------BC-A-----
CA B __   CA_B_   ___AB_C_   _ BC _A   _ NNNNNNNNNNNNNNNNNN

I'm not sure what is printing the other characters, but control is eventually passed to main and it acts similarly to 3.1. However, with sleepms instead of a loop, the quantum is always exceeded and the same character never prints twice in a row.

With sleepms(0):
Again, mostly N printed continuously, with this interesting bit:
NNNNNNNNNNNNPPPABCABCABCBCABCABCABCABCABCANNNNNNNNNN
Note that it is again very similar to 3.1, but because of sleepms(0) control is always given up immediately after each letter is printed. However, the other random characters are gone. Those are still a bit of a mystery to me.