Chris Cohen
CS354 Spring 2020
Lab 2

3.2

a) main()'s priority is INITPRIO (20), so I hypothesize that neither of these sendch() processes will ever run, since their priority is 10, and in XINU, lower priority numbers mean less important. After running the code, my hypothesis is wrong. The characters A and B are outputted continuously and infinitely, switching off equally every 30ms (in a round-robin style) as specified with the QUANTUM constant. This is because the main() process reaches the end after starting the two other sendch() processes, then it is killed. After that, the only processes left in the queue are the two equal priority sendch() processes, so they run in a round-robin style.
b) In this experiment, the same thing happens as in part (a), but A and B switch off at a faster rate, given 5ms each as specified by the newly-changed QUANTUM constant.
c) In this experiment, the same thing happens with main() reaching the end and being killed after starting processes A and B. Neither of the processes run before main() is killed because they still have a lower priority than main(). Once main() dies, processes A and B are the only ones left in the queue, and since process B has a higher priority than process A, process B is the only one that runs, infinitely printing 'B'.
d) In this experiment, main() never reaches the end because process A has a higher priority than main(), so as soon as process A is created, it's scheduled with that higher priority and infinitely prints 'A'. Process B never executes either, because A has a higher priority.
e) In this experiment, the same output as (d) occurs, but the scheduling is a bit different. In this case, process A and main() have the same priority, so when process A is scheduled, it is scheduled in a round-robin style with main(). Therefore, 'A' is printed infinitely while main() finishes. After process B is scheduled and main() is dead, only process A is left executing, infinitely printing 'A', because process B has a lower priority than process A.
f) This modification to make sure that no process can be changed to have a priority value less than or equal to 0 is important because that would mean the NULL process does not have the lowest priority. The NULL process is supposed to always have the lowest priority, because we want it as the last lifeline for the OS (that does nothing and never exits).
    i) Also, that other process created with a negative or 0 value would never run anyways because the NULL process disables interrupts and infinitely loops in order to keep the system alive. This means that, since it never ends and can't be interrupted, those processes with lower priority than 0 will never run.

4

a) getppid() → **system/getppid.c**
b) gettmslice() → **system/gettmslice.c**

5.1 - 5.5 → wgetprio.c, wgetpid.c, intr.S (_Xint33)

BONUS → wgetppid.c, intr.S (_Xint33)