Assignment 3

Question 1

Xinu's implementation of a queue as a series of elements into an array uses less space. In Xinu's implementation, a given process' PID is implicitly declared via its location in `queuetab[]`. The PID is the same as the index of `queuetab[]` at which a given process may be found; in the doubly-linked list implementation, another field in the `qentry` struct is nececessary because the PID of a process can no longer be referenced simply via `queuetab[i].next` or `queuetab[i].prev`. PIDs and pointers in our version of Xinu are the same length, 32 bits, so the extra field `pid` in `qentry` adds an extra 32 bits of memory per process. Assuming *n* processes, that is an extra *4n* bytes total of memory used by implementing doubly-linked lists.

Even though it uses more memory, a doubly-linked list implementation may very well take less time for FIFO operations, as traditional `enqueue()` and `dequeue()` operations add onto the end and take off the start, respectively, and each element in the list contains references to its direct neighbors.

On the other hand, Xinu's array implementation is very likely faster at non-FIFO operations such as the priority queue, because accessing an arbitrary element of an array is a constant-time operation, while a doubly-linked list requires traversing the entire list (or entirely reordering the list by a non-FIFO heuristic, or creating a new list that operates by that non-FIFO heuristic).

In systems which do not have a hard limit on the number of processes (or, more likely, which have a very large number as the limit), a doubly-linked list implementation may be more appealing because of its dynamic size. At any given time, given *n* active processes, the list need only contain *n* elements and only take up the memory needed for those *n* entries. An array implementation, on the other hand, automatically takes up the maximum amount of memory because of its static size. However, it eliminates the need for an explicit PID field in the `qentry` struct, saving some memory and making references to particular processes clearer, and non-FIFO operations require less overhead.

Question 2

A valid queue ID in Xinu is an integer less than the number of elements in `queuetab[]` and, generally, greater than 0. More specifically, a valid queue ID really should be greater than or equal to NPROC, to allow for sufficient entries in `queuetab[]` and to prevent accidentally accessing an arbitrary process instead of the head or tail of a particular queue.

Question 3

The original `resched` function's .o file contains 49 instructions. My `resched2.o` contains 53.

The extra commands between the two come from the extra checks needed to ensure the next state is an eligible state, whether the state should remain PR_CURR or change to PR_READY if the process does remain eligible, and assigning the state of the current process within resched2 instead of outside the function (as with the original resched). `resched2` uses an extra register compared to `resched` for the argument `nextstate` that is now passed in instead of referenced from the current process' `procent` struct.

Final Commits

My final commit (prior to backing this report up on github) for each branch may be found at the following URLs:

- master: https://github.iu.edu/htdoughe/OS-P536-F17/commit/6af974d8bc6de432fac7a0ff0e3b2f2518c04183

- pointer_list: https://github.iu.edu/htdoughe/OS-P536-F17/commit/e6bfba0847be1f1cc019d747fe29f66903e6be37

- validate_list: https://github.iu.edu/htdoughe/OS-P536-F17/commit/5cdf45b7fbb6cc5b63c4f3ffde2fbf9f098bcfe0