Documentation Notes

User Processes P1:

- Expected output:

G012_test: START
G012_test: total 6 tests
G012_test: test 6 OK
G012_test: test 5 OK
G012_test: test 4 OK
G012_test: test 2 OK
G012_test: test 3 OK
G012_test: test 1 OK
G012_test: 6/6 tests OK
G012_test: 0/6 tests FAIL
G012_test: END

- Proc1 takes up all the memory to test blocking on resource
- Proc2 is a basic test to check if a memory block on its own can be requested
    - Proc2 is designed to be blocked and later unblocked
- Proc3 checks a standard request/release pair
    - Proc3 is designed to be blocked and later unblocked
- Proc4 is designed with pre-emption in mind giving a higher priority to Proc6
- Proc5 changes its own priority and checks that it is changed correctly
- Proc6 ensures that our checks around releasing memory disallow releasing something which is not a memory block that we manage
- We have custom functions to print out the result of each test and to check when tests are complete to print the post amble

User Processes P2:

- proc1 HIGH reg command / command will exit
- proc2 HIGH reg command / check argument correct->exit
- proc3 HIGH clock / does not interact with anything else
- proc4 HIGH always ready
- proc5 Medium send delayed message to self, ask for all memblocks, release them upon receive the delayed message
- proc6 Medium ask for one memblock, then release memblock.

- 1,2,3 set self to LOWEST upon finishing
- 4 set self to LOWEST if #proc finished is 3
- 5,6 set self to LOWEST upon finishing

- run proc 1,2,3 (block on receive)
- while waiting proc 4 (non-block)

- 5 sends a delayed message to self, takes all available memblocks
    - upon receiving, it releases all memblocks it holds.
- 6 ask for one memblock (should be blocked when 5 is not yet started releasing)
    - then release that block immediately.

Our Notes

- We keep track of our memory blocks with a stack
    - Requesting memory pops off
    - Releasing memory pushes on
    - Stack makes it so that we don't have to keep track of both the front and the back; in general the math just becomes a lot easier
- We have two arrays of queues
    - One array keeps track of the ready queues for the different priorities
    - One array keeps track of the blocked queues for the different priorities
- We have ProcessNode as a queue member wrapper around PCB which allows us to have next and previous for each PCB without changing the structure of the given PCB
- We implemented a memory block queue as well which we don't use as we instead go with a stack, but we kept it for reference purposes in case we need it at a later time