I have chosen the fields from rusage that are currently supported by linux and analyzed a couple of them. One thing I noted is that as each send and receive uses system calls in its respective functions, it increases the System CPU time used proportionally. The User CPU time only increases when there are requests by clients to connect, which makes sense as rusage gives us the data from the main processes’ thread and not the threads created to handle the clients after (worker threads). When running 8 simultaneous clients, the maximum resident set size used increased slightly to 996kB (in comparison to ~800kB for one client) and this is most likely due to the malloc-ed struct containing the necessary information for the worker thread (which has not been freed yet). I have yet to see any hard page faults but this merely tells me that the linux “garbage collector” isn’t clearing the servers used memory (though it’s too small to make a difference). The page reclaims on the other hand makes me wonder that the memory allocated to messages that had been freed were referenced again (possibly the kernel had yet to flush it), so it merely reclaims the pages (not too sure about this). The number of block output operations on the other hand has to do with writing to the log, indicating the number of write ‘chunks’ that had been done. Voluntary context switches on the other hand is always in the hundreds range and as seen from the log file, the thread scheduler uses a sort of round-robin method as the log entries are not consistent in term of who runs after the other.

I shall be running 20 simultaneous clients connected to the server on both digitalis and the NectarCloud VM.

Performance and Usage Statistics

(Digitalis) (Nectar)

Number of successful connections: 20 20

Number of successful guesses: 0 0

User CPU time used: 0.2999s 0.32000s

System CPU time used: 0.21996s 0.56000s

Maximum resident set size used: 1140 kB 3104 kB

Page reclaims (soft page faults): 352 178

Page faults (hard page faults): 0 0

Number of block output operations to log: 72 64

Voluntary context switches between threads: 890 780

Involuntary context switches between threads: 7 69

There is not a big difference between the User CPU times and perhaps the System time was higher for Nectar because of Ubuntu’s extra layer of checks for their system calls. The maximum resident set size is much bigger for Nectar and I’m sure digitalis allocates a smaller memory size since there’s many users (students). Involuntary context switches between threads are surprising for Nectar as it’s almost 10x the number of switches of digitalis. Maybe this is how Nectar deals with system calls, i.e. it is supposed to return control back to a particular thread, but forces a different thread to run instead.