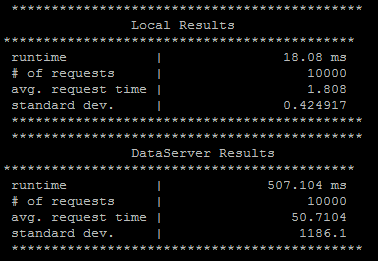
**Machine Problem 3:**

Part 1:

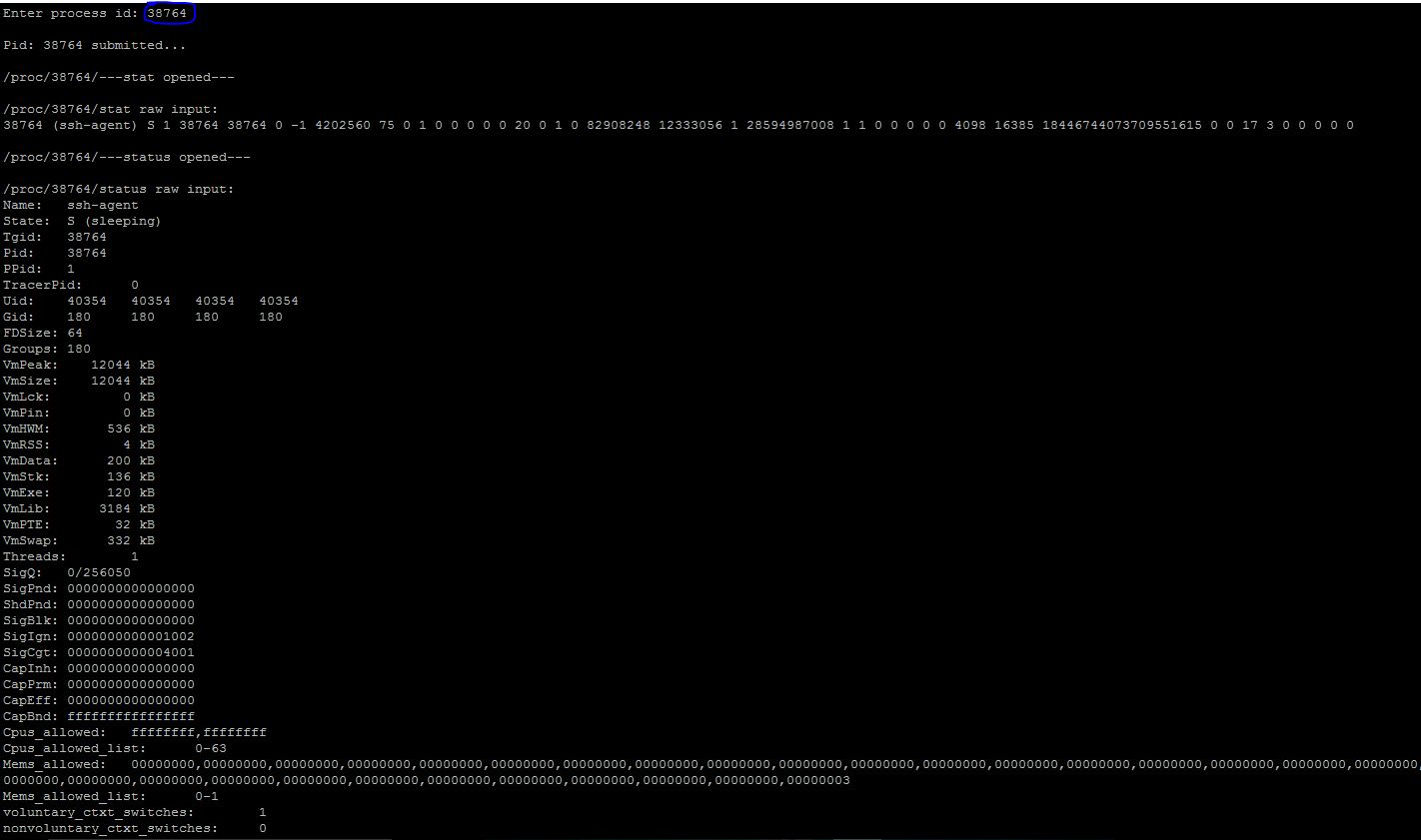
For part one, we developed the client.cpp program to run with dataserver. cpp by running fork() and a system call to allow dataserver to run from the child process. For measuring the time to send a request and get a response, we used the “chrono” library. We took the system time before and after the request, subtracting the former from the latter to obtain the total time.

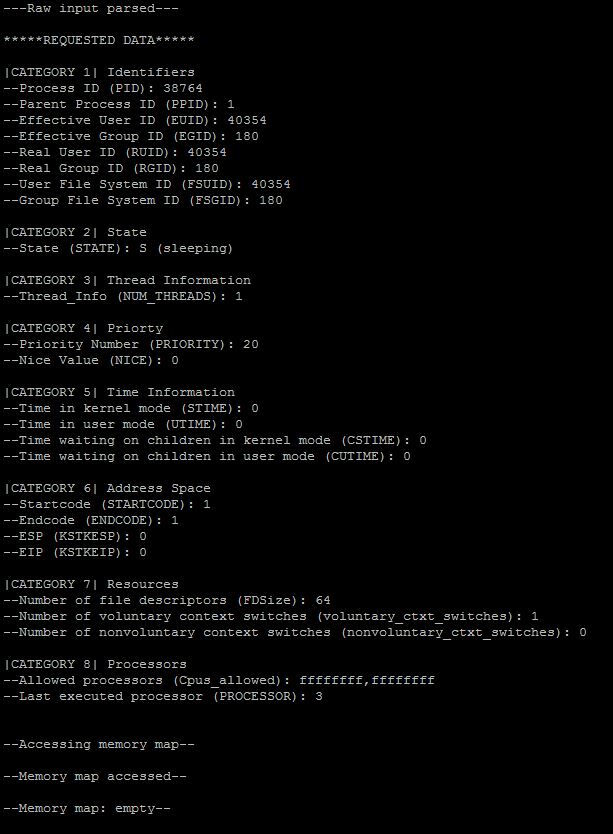
Our results were gathered after collecting runtime information from 10,000 requests to both the server and local requests. The topmost table depicts the data collected when handling requests with local functions. This proved to be the faster option and took a fraction of the time needed to handle requests with the server. The time difference is due to the time the request spends between the client and the dataserver.

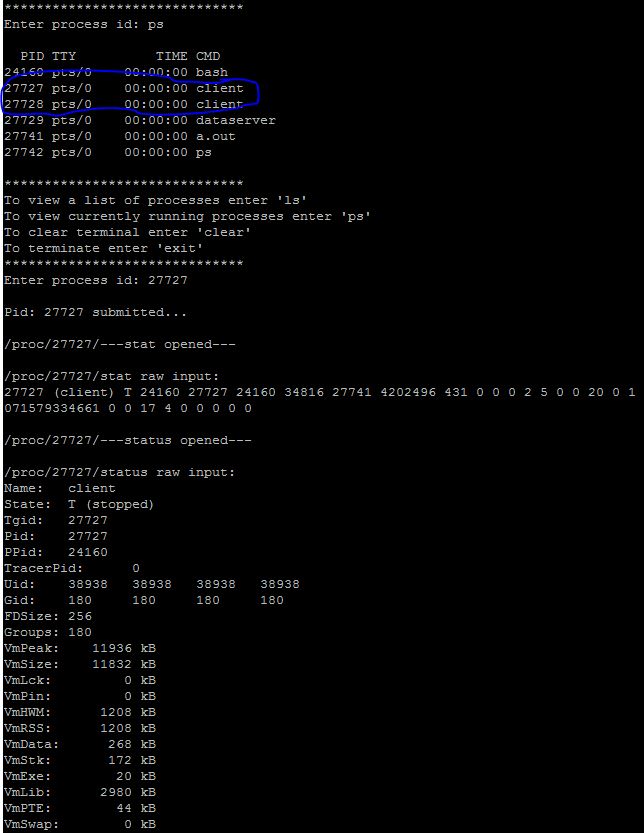
The bottommost table lists data collected on the requests sent to the dataserver. The total runtime was 507.104 milliseconds. For 10,000 requests, the average request time was found to be 50.7104 microseconds. The standard deviation was 1186.1microseconds.

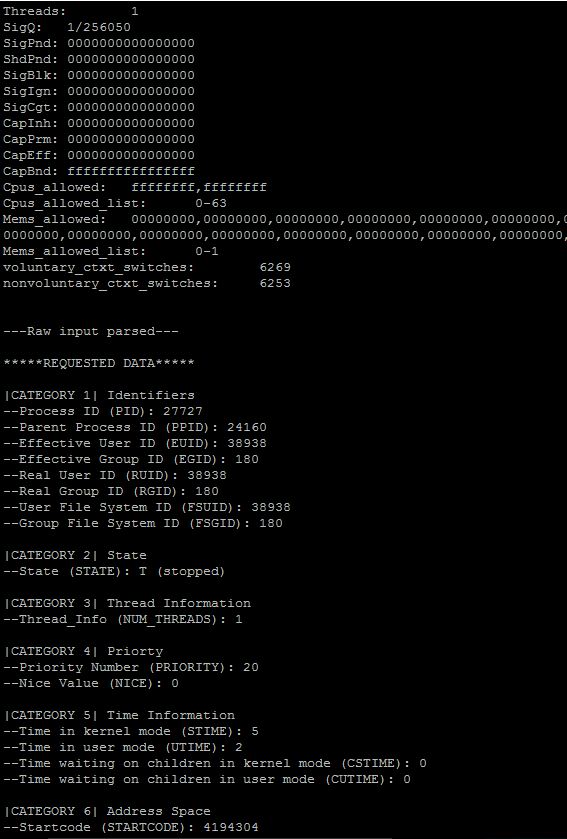


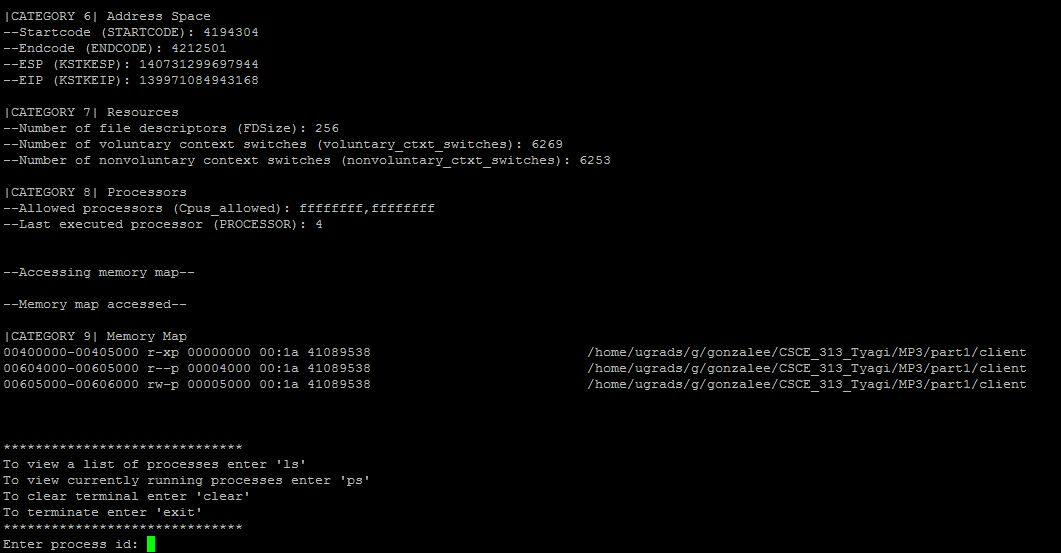
Part 2:

1. 



1. 





1. The kernel usually only checks the effective user ID. When a process tries to open a file, the effective user ID is checked by the kernel which then decides to grant access to the file or not. The real user ID matters when one wishes to change the effective user ID of a running process. A situation where the real user ID and effective ID are different is when a process is running as root and a particular user sends a rquest to access a file. In this situation, the effective user ID will change from root to the user and check to see if they have access to the desired file. After all operations have been carried out, the effective ID returns to its original value using the real user ID.
2. Most files in /proc are read only because changing them would alter information that the kernel relies on, potentially causing problems.
3. The task\_struct is allocated by the slab allocator to provide object reuse and cache coloring. Most kernel code dealing with processes works directly with task\_struct, so it’s crucial for the computer to quickly look up the task\_struct of the process that is currently running.