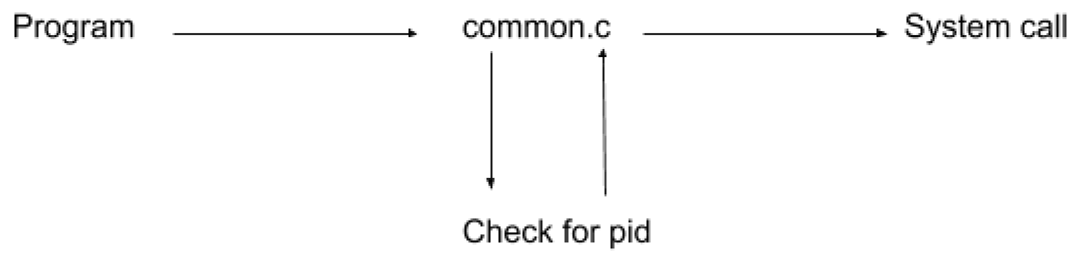
Daniel Roh

Project 2 Design Document (initial v1.2)

Requirements:
- *A system to track which processes are not allowed to use system calls*
- *A data structure to store “banned” processes but is fast enough to not slow down the system when multiple system calls are being made.*
- *Implement system call functions block and unblock.*
- *Implementation of 4 userspace programs to test the system calls*

The data structure that I plan to use for holding the banned processes will be a hash table. Given that the kernel will be processing many system calls per second, chances are that a linked list will be too slow to use leading to delays. I decided to use a hash table as it is a fast data structure and due to the fact that my skip list from project 1 was a disaster. A potential back up in the case that the hash table does not work the way I hope would to switch to a binary search tree.

For the kernel code, given the hint of looking at entry_64.s. A function named “ENTRY()” is commented as the only entry point into the kernel from userspace. Given this entry point, this function would be the best location to insert the kernel code to check the process ids. However, this function deals with coding in assembly which would be a bit annoying. After looking around a bit; a file common.c seems to have the call “do_syscall_64” in line 283 which appears to an entry point before the assembly. This would be my guess as to add the function to block system calls adding a check in line 285 before the initialization of this call would do the trick.



To control what pid’s are blocked, 3 system calls are asked to be implemented, block, unblock, and count. These system calls would access the hash table to edit or return information.

For testing a program run will also be implemented that would allow a user to run a program with the permissions of a given user. With that program, four separate programs will be implemented that use the block/unblock function calls to test the implementation of blocking.