eid: vc6246
Vaibhav Chattree

Project #3A - Adding NULL pages to xv6 address spaces

This project was an intro to virtual memory for the xv6 system. The code had to be updated so that an exception is thrown when a null pointer is dereferenced. Since the first page is invalid, the entry point had to be changed to 0x1000 in the Makefile. Exec() also had to be updated to start at pgsize when loading the program into memory. The function mprotect and munprotect had to be added to the syscall files as is standard for new methods. Also, sysproc had to be updated to retrieve the args. For mprotect, the length had to be checked, as well the address to make sure it is page aligned. The walkpagdir is used to get the physical address of the page table using the virtual address. Then writable bit of the page table entry must be cleared with the & operator to disable writing. This is repeated arg len times. The lcr3 function is then used to make sure the hardware knows the page table was changed. For munprotect, the same logic as above was applied. However, the writable bit was set using the | operator to allow writes.

Each test case was tested to ensure it passed. For test 1, I had to be sure the program entry point was not 0. For test 2, a null pointer is dereferenced causing a trap and nothing to output. Test 5 checks for good arguments passed to mprotect which passed. Test 7 returns a value from mprotect where main is mapped, so the page fault trap must occur. Therefore the output under the trap does not print. Overall after initially writing the code, I had to run all the tests and see which ones I was failing. I failed quite a few tests initially, and had to update my code to be sure it could handle each test. I had set the writable bit incorrectly at first. I did fail some hidden test cases. Overall this was a tough project but it taught me a lot about the xv6 system.

Project #3B - xv6 Kernel Threads

In this project we added Kernel Threads to the xv6 system. Clone was created to create a new thread with the same address space as the initial process. This new process used args stack for the user stack. A few things had to be checked for validity. The size of the current process minus the stack had to be less than the page size. The stack had to be aligned with the page size. The current process page dir and size were copied to the new process. Allocproc is needed to create a slot in the process table. Clone's new thread is in the same address space as the parent. I used an array of size 3 for the user stack which had the fake return PC and arg1 and arg2.

The next call to add was join. Join searches the table for exited children. When the child exits, its state becomes zombie. So once the state of zombie is found, the pid is retrieved. Then the child process is freed. This id repeated until no child threads remain. Thread create is the next call needed. This creates the new stack and creates the child thread with clone. I had to check to be sure the memory is page aligned. I also have to acquire the lock, then malloc for the stack, then release the lock. Thread join is next call created. This calls join, and frees up the stack. The lock has to be used during the freeing of the stack. The calls lock acquire, lock

eid: vc6246
Vaibhav Chattree

release, and lock init were written. These implement the lock with a ticket system. This uses the fetch and add method.

At first I was failing many of the tests, and had to update my code to get the tests to pass. The first test simply checks for multiple threads with different pids. Test 2 passes in invalid args to clone. I had to fix my clone to check for invalid args. Test 6 tests clone and join along with locks. This took some debugging to get the address space working correctly. Test 8 tests the ticket lock. My ticket lock worked.

Overall, the code has to be carefully written to ensure the threads do not interfere with each other. The function calling convention and stack must be understood for this to work. Also, the lock must be used to ensure a value is not being modified incorrectly. This project taught me a lot about multiple threads and the code needed to handle them.