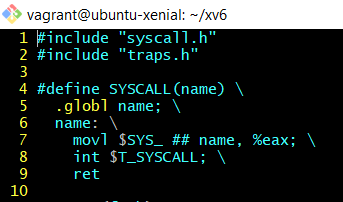
**STUDENT NAME: Nikhil Sarika**

**CS 450 Fall 2021 Assignment #02**

Due: **Monday, October 11th, 11:59 PM**

**Problem 1:**

usys.S has the assembly code defined to execute the system calls.



The assembly code has a function defined called SYSTEM which takes in an argument called name to execute the specific system call.

For the close function the line number 7 is executed as

movl $SYS\_close, %eax;

A “syscall.h” file is defined with all the system call number for all the system calls.



For close 21 is allocated as the system call number.

As close() is a system call and the user doesn’t have the privilege to execute it. The system calls are executed by the kernel with the help of Traps.

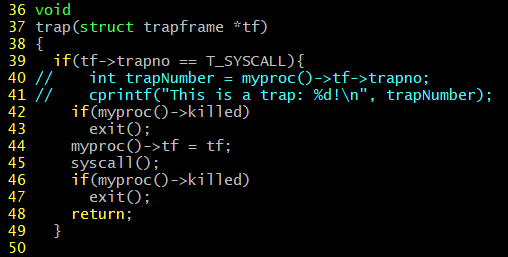
In X86 the traps are executed with help of an assembly instruction called “int”. The trap function passes.

As we can see in the first image the int instruction is executed in line number 8.

During the execution of this “int” instruction the cpu context is saved on the kernel stack.

The “alltraps” assembly function saves the data within the registers and other data that is being used by the current program.

After saving the data, the alltraps function transfers the control to trap(tf) which is implemented in trap.c file.



The trap(tf) function checks for the trapno that is saved in the trapframe struct.

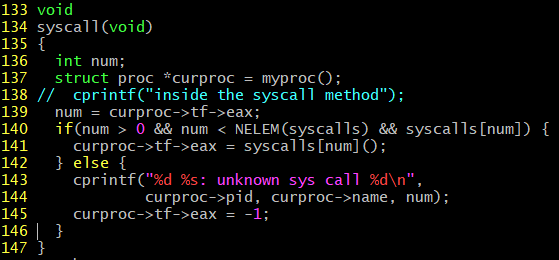
The trap numbers are saved in the traps.h hearder file



The trap number for system calls is defined as 64.

If the trapno is a match in line 39, The systemcall() function in line 45 is executed.

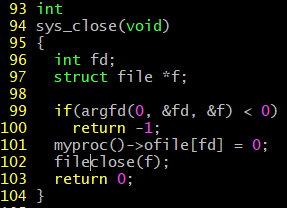
Systemcall() function is defined in the syscall.c file.



The line 139 fetches the eax register value which is stored on the kernel stack.

As 64 is a valid value for the system call the systemcall for close is executed.

The close system call is located in the sysfile.c folder.



The sys\_close function executed the the close systemcall.

The sys\_close validates the argument with the help of argfd() function defined in the samefile.

As we have passed an invalid value to argfd this method would return -1. The returned value is used to set the eax register in the syscall()’s method line number 141.

The last line in the usys.S’s method takes this eax register value and returns it to the user. As -1 indicates a error response. The ret instruction returns the value and does the necessary error handling.

**Problem 2:**

**Steps to Implement a SystemCall:**

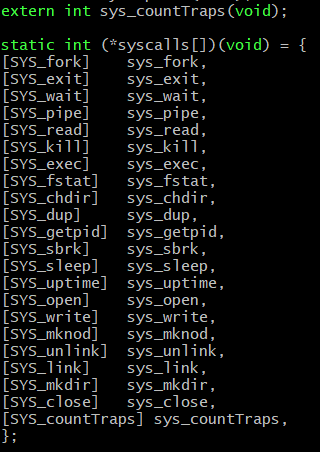
1. Modified user.h file to include method signature i.e the method definition and the return type.



2. Modified syscall.h file to include the system call number. For countTraps() system call we have incuded a system call number 22



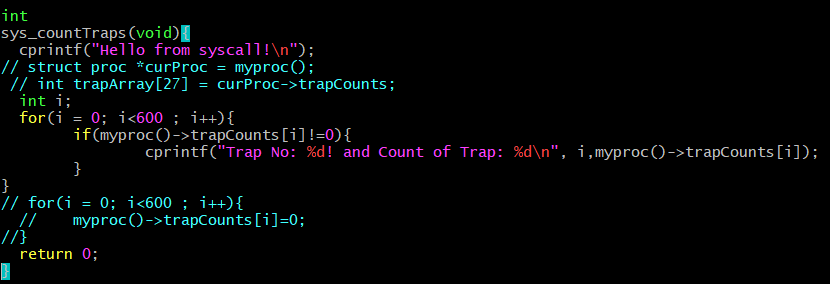
3. Modified syscall.c file to include countTraps() in the array that stores all the system call numbers.



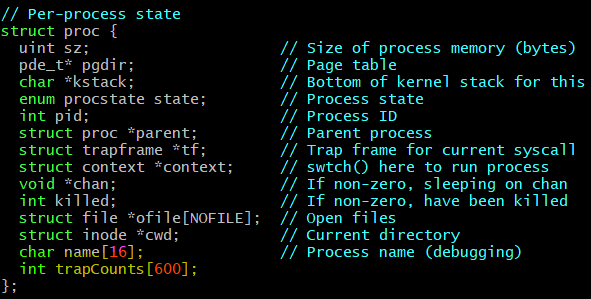
4. Modified usys.S assembly file to invoke the system call for our countTraps()

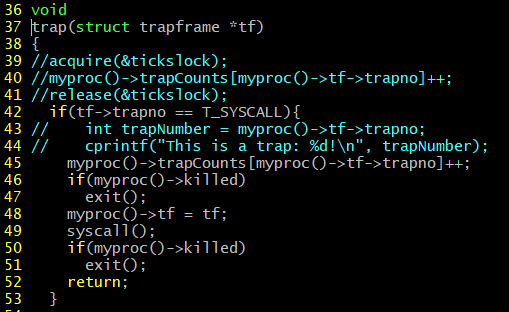


5. Modified the sysproc.c file to implement our countTraps() system call.

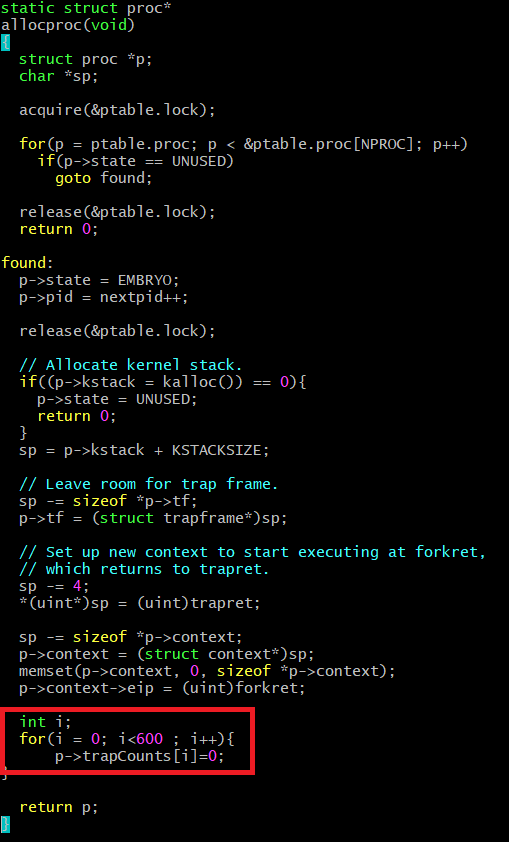


6. Modified proc.h to edit the proc stuct to include a custom array that holds the count of all the traps that occurred in a user process.



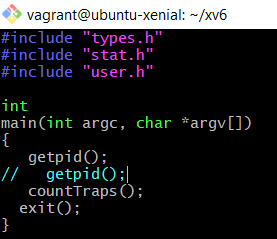
7. Modified trap.c file to increment the trapCounts array that we have included in the proc struct. The entries are updated based on the trapno that is part of the trapframe. Line 45 is added to increment the entries in trapcounts array which represents the number of traps.

8. Modified the proc.c file to edit the allproc() method. The countTraps array is reset to default values before allocation of memory for each user process. The values of countTraps are set to zero before each allocation.



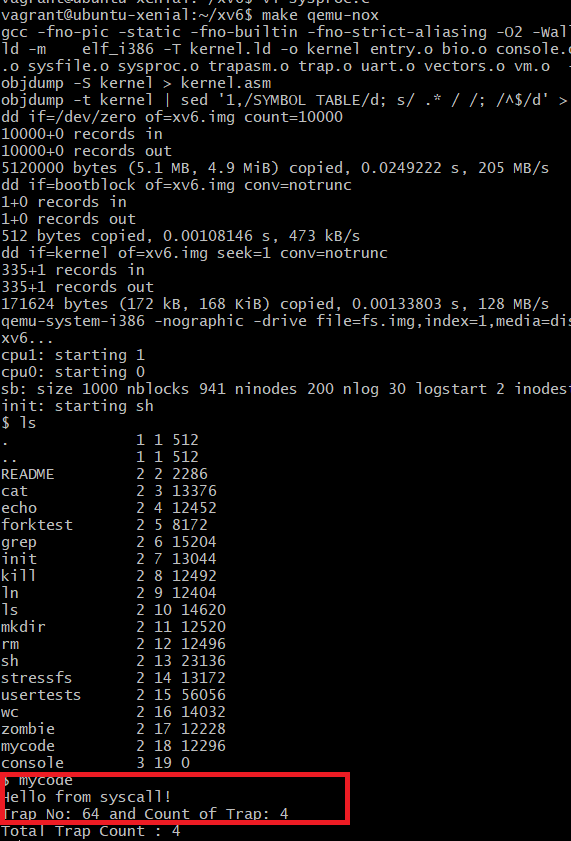
**Testing of Implementation:**

**1.Implemented mycode.c file to test the count of traps.**

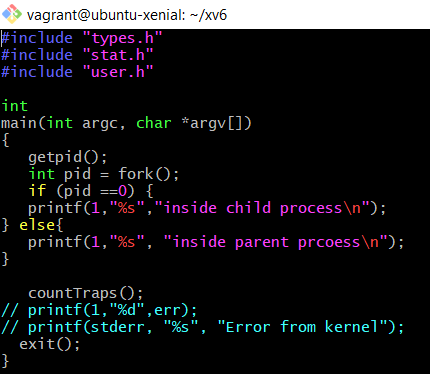


a. Included two system calls to get the count of number of traps in the program. The program is included in the make file.

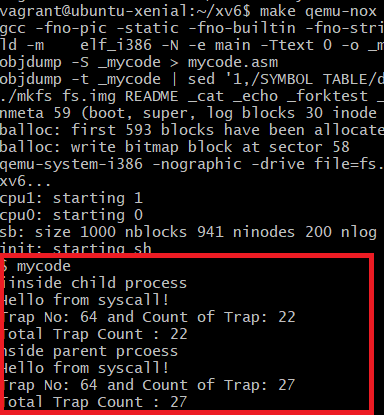
Test 1 output:



2. Modified the mycode.c file to include fork() system call. The test case is similar to the first one except that the fork starts a child process.



Test Case 2 Ouput:



The second test case has additional traps that are a result of the fork() system call.