CS3500 LAB 2 REPORT - CS19B012

Problem 1:

This problem is analogous with echo with a single argument (string aka argv[1]).

Created a user file test_problem_1 & called the system call inside main. Added the new system call prototype to user/user.h. Added corresponding entry("echo_simple") to user/usys.pl file & adding file to Makefile

Added #define SYS_echo_simple 22 to kernel/syscall.h file

Added extern uint64 sys_echo_simple(void); & [SYS_echo_simple] sys_echo_simple inside syscalls array to kernel/syscall.c file

Defined the function uint64 sys_echo_simple(void) in the fole kernel/sysproc.c, which just prints the passed string to stdout using argstr (reading the string argument)

Problem 2:

Just extending on the previous problem, so all the common steps aren't mentioned. The only main difference is the definition of the required function, where we pass argc & *argv here for the sys_echo_kernel function.

First, I loaded the value of argc into variable n using argint() & the address of argv to argv_address variable using argaddr() function. Then, I defined a argv[] array to store the address of each string (argv[i]). Iterate from I = 1 to n & store the address of the string in argv[i] & then read the string from this stored address argv[i] into str string. Then print it in each iteration.

Problem 3:

Just doing all the common stuff as the previous 2 problems. But here add trace_mask field to struct proc in kernel/proc.h file. And for the definition in sysproc.c, we load the the value using argint to the trace_mask field for the process.

And coming to syscall.c, after doing the general additions, we define a new array call_names that maps the correspong [SYS_fun] with the corresponding name strings, which is used in printing the system call name later when printing the required details (using the corresponding num -> name map). In the syscall() function, add an if conditional to check for trace_mask field of the process and print the required details (process id name, return value)

Problem 4:

Extension to previous problem by printing arguments for the system calls in addition.

Defined an array of integers arg_count[] where each [SYS_fun] is mapped to the corresponding number of arguments that the system call takes. So, the modification to that of the 3rd one is that we print args after each system call is printed (as the above question) using the above defined array, we get the value of arguments using argraw() and number of arguments to be printed is equal to the corresponding value of that function in arg_count[] array. Value stored in a0 regoster is saved in a temp variable before it gets overwritten by return value of system call.

Problem 5:

After completing all the common steps just as we've done earlier, define sys_get_process_info() in sysproc.c file. Store the address of the struct object in info using argaddr(). Then called myproc() function to get the process & then stored the corresponding values of pid, sz & name (last one using strncpy) into obj. Then using copyout, copied the values from this obj to the user destination "info".

PS:

The Assignment was submitted ~5 hrs after the deadline as my VirtualBox Ubuntu crashed and was unable to boot into it again, so I had to delete the files and redo everything from scratch (including installing the entire OS & shaktitools, including having to download .iso file). Will make sure that this'll not happen again

Video Proof of System crash