Operating Systems - Monsoon 2021

Sambuddho Chakravarty, Arani Bhattacharya

November 12, 2021

Assignment 2 (Total points:135)

Due: Time:23:59 Hrs. (Hard Deadline)

1 `Exec` family system call and basic IPC using `signals` API(Total points: 65).

The first exercise tries to familiarize you with `exec` family system calls and signals API. In this program you are required to create a child process, lets call it S1. You need to save the PID of the process just created using a local variable. The body of process S1, should register a signal handler for SIGTERM which could be raised by any other process, using the `kill()` system call.

Thereafter, following the creation of process S1 the main (parent) program should fork two more child processes, ST and SR. Both ST and SR should register signal handler for the signal SIGALRM. Both of them should setup appropriate timers using the `setitmerval()` system call (that should invoke a timer for both the processes, individually) at every predecided interval (*e.g.* 1s, 2s, *etc.*).

Functionality of SR: Corresponding to SR, the SIGLARM handler should read a random number from the CPU using the the RDRAND instruction (using inline assembly). Thereafter, SR needs to enqueue the random number and the SIGTERM and deliver it to S1's SIGTERM handler (using `kill()` system call). S1's SIGTERM handler should print the random number supplied by SR.

Note 1:*Functionality of ST:* We require a similar SIGALRM handler for ST. The ST process should read the CPU timestamp counter (using inline assembly to run RDTSC instruction), and coverts into a human readable string (*i.e.* representing the current date and time), at every interval. The interval can be set using the `setitimer()` system call, that raises then SIGALRM signal everytime the timeout elapses, that is trapped by an appropriate signal handler. The random number should be obtained (using the inline RDTSC instruction), in the SIGALRM handler function. Much like SR, the ST's SIGALRM handler should enqueue the string along with the SIGTERM signal and thereafter send it to S1 (using `kill()`). ST's SIGTERM signal handler should print the string supplied by ST.

Using* `exec` *family system calls to launch the two programs SR and ST: Also, *it must be noted* that the entire functionality of ST and SR should be embodied in two individual programs. In other words, there should be two executable programs E1 and E2 corresponding to SR and ST respectively, that should accept the PID of S1 as an argument. These programs, *i.e.* E1 and E2, must be launched from SR and ST respectively using the `exec` family of system calls, especially those that allow launching programs with arguments.

What to submit/rubric

- All the three program sources, corresponding to the main() function, program E1 (which is invoked from SR) and E2 (which is invoked in ST). These programs should compile successfully. **[Successfully compilation of all the three programs: 25 points. No points for programs that do not compile.]**
- Correctly functioning programs achieving the required functionalities. **[All the three programs correctly functioning: 25 points. Some of the programs working correctly, but not all: 12.5 points. Programs that seem logically correct but none of them work as expected (*provided*, it compiled successfully: 6.25 points.]**
- A single Makefile to compile the above three programs.**[Fully functioning Makefile: 5 points.]**
- README/Write-up describing the program logic used for achieving the above (no more than one page).**[10 points]**

2 Kernel memory copy (`kernel_2d_memcpy()`) (Total points: 70)

This exercise aimed to test your understanding how system calls work. You need to write a system call, `kernel_2d_memcpy()`, which copies one 2-D floating point matrix to another. You would require using kernel functions like `__copy_from_user()` and `__copy_from_user()` to read data bytes from user space and write back to user space. In other words, this is a version of `memcpy()` that relies on the kernel to do the necessary copy operations, which are otherwise usually done directly in the user space (using the standard C library routines).

What To submit/rubric.

- The `diff` between the stock kernel and the kernel with your appropriate system call. This `diff` could be patched into the stock kernel code so as to achieve the required functionality (*i.e.* the system call) and used eventually. **[Successfully compilation of the patched kernel: 50 points. No partial points.]**
- A sample program to test the above system call. In the program you could hard-wire the source 2-D matrix (*i.e.* no need to take input from user at runtime or via a file). **[Program that correctly calls the system call**

with all the appropriate parameters and does the copy, provided (1) above works *i.e.*:10 points. Program compiles (as well as (1) above works), but copying is unsuccessful: 5 points.]

- README/Write-up describing the program logic used for achieving the above (no more than one page). **[10 points]**