Assignment-5

CS342: Operating System Lab

General Instruction

• Assignments should be submitted through shared link.

• Assignments will not be accepted after the due time.

• Markings will be based on the correctness and soundness of the outputs. In case of plagiarism zero marks will be awarded.

• The assignments must be written in C language. Proper indentation & appropriate comments (if necessary) are mandatory in the code. Further,

In today's lab session, you will learn about how a computer boots, and write a dummy operating system. Further, the goal is to demonstrate the usage of signals.

You may refer to given link for signal assignment:

1. http://www.alexonlinux.com/signal-handling-in-linux
2. http://www.cs.princeton.edu/courses/archive/spr06/cos217/lectures/23signals.pdf

Booting Unraveled

When a computer starts, a special program called the Basic Input/output System (BIOS) is loaded from a chip in to the main memory. The BIOS detects connected hardware devices, resets them, tests them etc. and also looks for the special sector (the boot sector) on available disks to load the operating system. The BIOS reads the first sector of each disk (one by one) and determines whether it is a boot disk (a disk with an operating system). A boot disk is detected via a magic number 0xaa55, stored as the last two bytes of the boot sector of a disk.

Convert assembly (mnemonics) code to binary using the following:

$nasm boot_sect1.asm -f bin -o boot_sect1.bin

If you want to see what is exactly inside the bin file, the following command will help you.

$od -t x1 -A n boot_sector1.bin

The above binary can be used to setup (copy to) the first 512 bytes (the boot sector) of a disk. Instead of writing this boot sector to a physical hard disk, we can use an emulator. QEMU is a system emulator that provides a simple and nice method to run your boot sector directly from the bin file.

$qemu-system-i386 boot_sector1.bin

The above command emulates a system using the file provided as the attached disk (which in our case has the first 512 bytes of interest).

Problem 1a: The boot_sector1.asm file, in the myos directory, shows a sample assembly code that is supposed to do something. The idea is that this program produces machine instruction that would be copied on the boot sector and the computer powered-on.

Compare the outputs of the booting process using the two programs, boot_sector1.asm and boot_sector2.asm, and justify your results. Submission should contain binary files and screen shots of QEMU along with an explanation.

Problem 1b: Let's do something slightly more interesting. On boot, our custom OS should print out a message. Write a program, hello.asm, that prints "Hello" on the screen during boot-up.

To print a character on the screen, us the following code with appropriate repetitions and changes.

mov ah, 0x0e: set tele-type mode (output to screen)

mov al, 'H': one ascii character hex code in register AL

int 0x10: send content of register to screen via an interrupt

Setup hello.bin as the input file for QEMU to use for booting and test output (capture screen shot in hello.png.

Problem 2a: Write a program (p1a.c) to handle the SIGINT and SIGTERM signals. The process should print a custom message asking a question about whether the program should really exit, and exit only on confirmation. Look up accompanying files for sample outputs.

Problem 2b: Write a program (p1b.c) that spawns a number of child processes, using the following command:

./p1b

where n is the number of child processes. Each child process sleeps for a random duration and exits. Override the SIGCHLD signal handler in the parent process and print the sequence of exits of the child processes.

Problem 2c: What would happen if a child process forked another (grand) child process, which eventually terminated. Recreate this situation (p1c.c), check for output and reason about it. Two situations to test—when the grandchild process terminates before the child and when the child terminates before the grandchild. Write your observations and reasoning as part of the submission. (Note that this requires handling the SIGCHILD signal.)