CSCI/ECEN 5573: Advanced Operating Systems
Fall 2024

Programming Assignment Zero

Review of Fundamental Concepts

Due Date: 09/20/2024

The main goal of this programming assignment is to review OS programming background typically learned in an undergraduate course in operating systems. Before doing this assignment, please go over the undergrad OS material posted on Canvas carefully and make sure that you understand every line of code in the sample programs included there. Experiment with those programs by making changes to gain deeper understanding.

1. Gain experience with using *fork( )*, *exec( )*, interprocess communication (*shared memory* and *pipe*), POSIX semaphores (*sem_overview*), *kill( )* and *sleep ( )*.

   Write two separate programs (parent_pgm and child_pgm). The parent_pgm creates two child processes, each of which run the child_pgm. The process running the parent_pgm is named "Parent", one child process is named "ChildOne" and the other child process is named "ChildTwo". The child process names are communicated to the child processes using *exec* parameter. The three processes share a shared memory segment that's created by the parent and the key communicated to the child processes using pipes.

   Parent starts by reading a string from the terminal and writing that string in the shared memory segment prefixed with "Parent: ".

   The three processes then repeatedly take turns starting with ChildOne, to read a string from the shared memory segment, write this string in a file named "assignment zero output", read a string from the terminal and then write that string in the shared memory segment prefixed with "*xxx*: ", where *xxx* is the process name (Parent, ChildOne or ChildTwo). The *assignment zero output* file is created and opened by Parent. When the Parent reads a string "TERMINATE" from the terminal, it kills ChildOne and ChildTwo, closes the *assignment zero output* file and then terminates.

   As an example, here is the output in the *assignment zero output* file for a sequence of strings entered in the input terminal:

Input Strings	**Output in *assignment zero output* file**
One two 3	ChildOne: One two 3
4 five 9	ChildTwo: 4 five 9
TERMINATE	Parent: TERMINATE
Nine	ChildOne: Nine
TERMINATE	ChildTwo: TERMINATE
10 4000 seven	Parent: 10 4000 seven
Four eight	ChildOne: Four eight
Ten eleven 12	ChildTwo: Ten eleven 12
Ten four	Parent: Ten four
TERMINATE	

CSCI/ECEN 5573: Advanced Operating Systems
Fall 2024

2. Assignment about virtual memory layout, symbol table, page tables, sbrk, brk, mmap

   The goal of this assignment is to explore how the virtual memory layout of a process changes when you use dynamic memory. Write a C program that includes multiple calls to *malloc( )* and *mmap( )* and prints the addresses of some variables to determine the boundaries of heap, stack, text, static variable, etc. Your program should include two command line arguments. Answer the following questions:

   - What is the size of your process' virtual address space?
   - Identify the locations where *argv* and *argc* are stored.
   - What are the addresses of code, static data, stack and heap?
   - In what directions do the stack and the heap grow? Explain your answer by using the virtual memory layouts at different execution phases.
   - Your dynamic memory (from *malloc( )*) may not start at the very beginning of the heap. Explain why this is so.

3. Write a C program that includes a function foo ( ), which creates a child process to run another C program. Answer the following questions:

   How does the virtual memory layout of the child process compare to the virtual memory layout of the parent process before the child process executes *exec* ( )? Specify address ranges etc in your observations.

   How does the virtual memory layout of the child process compare to the virtual memory layout of another process that runs the other C program from command line? Again, specify address ranges etc in your observations.