COMP 3430 – Operating Systems – Lab 4
Introduction to Memory Management

Purpose

The goal of this lab is to introduce you to some real-world Memory Management facilities available in Linux/Unix.

Introduction

Your lab will again take place in E2–420 EITC (where you will physically sit to do the lab) and will again use the CS Linux machines via remote login to `aviary` using ssh.

How to Prepare for the Lab

Your first step in preparation for the lab should be to read over the lab problems described below and make sure you understand them generally. The lab demonstrator will be available to answer specific questions during your actual lab. Your second step should be to review any relevant background material (e.g. class notes, handouts, and web links specified below) as well as to read any man pages referred to in this document. Additionally, you will want to download the program `mmap.c` from the Examples section of the course homepage in advance and make sure you understand it. Your final step in preparation for the lab should be to quickly review the relevant notes on memory management covered in class, to make a note so that you remember to bring the notes with you to your lab, and to print a copy of this lab document and the hand–in document (if you have not already done so) to bring with you to the lab.

What to do?

Answer the questions below on the hand–in sheet. Be sure to hand in your completed sheet to the demonstrator before you leave the lab. This is how you will be graded!

Shared Memory

Linux, like most Unix-based operating systems, supports IPC through shared memory. This is useful if you want to share data structures between processes rather than between threads. You can limit the memory that is shared whereas with pthreads, for example, the entire address space is shared. Shared memory is implemented by mapping the shared segment into a process' image. Typically we let the OS decide where in the image to map the shared memory, and access it through a pointer. System V shared memory, which is supported by Linux, creates the shared segment in main memory. Memory mapping of files is also supported, which is what we will look at in this lab. Memory mapping of files allows processes to directly access the contents of a file, without the normal read/modify/write cycle of file I/O.

To use memory mapping of a file, a handle to the file is first obtained using the `open()` system call. The `mmap()` function performs the mapping of the file; the contents of the file may then be accessed through the returned pointer. The `munmap()` function removes the mapping from the process' address space.

Questions

1. Answer the following questions by reading the `man` page for `mmap()`.

[2] 1a. The first parameter to `mmap()` allows the caller to specify the starting address of the mapped segment, but it is generally not used (i.e. we usually let the OS pick the address). Why do you suppose this is generally not used? How do we ask the OS to pick the address?

[1] 1b. The fourth parameter to `mmap()` specifies the type of the mapped object. If the process wishes to modify the mapped memory so that other processes that map the same file can see the changes, which type should be used?

[3] 1c. Which of the following will cause a mapped file to be unmapped: i) Calling the `munmap()` function, ii) process termination, iii) calling the `close(fd)` system call, where `fd` is the file descriptor of the mapped file?

[1] 1d. If a process that has a mapped memory segment executes a `fork()`, does the child process have access to the segment as well?

2. Download and compile the program `mmap.c`. Prepare a short text file and call it "`mmap.txt`". Run the program, demonstrate your running program for the demonstrator, and answer the following questions:

[2] 2a. The `system()` function executes a shell command within a program. Which command is being executed in `mmap.c`? Briefly describe this command.

[2] 2b. Examine the output of the program and find the listing for your mapped file. What is the size of the mapping? What do you think the fourth character in the "mode" entry indicates?

[3] 2c. The program output should contain 2 listings for the compiled program (`a.out`). What is different about these 2 listings? Can you explain this difference?

[6] 3. Write a second program (beginning with a copy of `mmap.c`) and make the following changes:

i. instead of hard-coding the file name, accept it as a command-line argument (remember you did this for assignment 1).

ii. Immediately after the `system()` function call, print the value of the pointer returned by the `mmap()` function (use `%p` to print a pointer in hexadecimal).

iii. Run your program with the argument `~young/secretmessage.txt`. What is the secret message? Does the value of the pointer agree with the listing displayed by `pmap`?

Demonstrate your programs for the lab demonstrator. The demonstrator will initial and make any comments on the hand–in sheet. If you are not able to demonstrate a working program during lab time then notify your TA and when finished submit your completed program by the end of the day using the dropbox on the D2L – if you do demonstrate, then a dropbox submission is not necessary.