Writing Assignment 1

Alec Merdler
CS 444: Operating Systems II
Spring 2017

1 Introduction

1.1 Processes

The explicit purpose of computers is to execute sets of instructions, called programs. As computers have increased in power and speed, the complexity and number of these programs has also increased. A simple program could count to the number ten, printing out each number along the way, then stop. The resources needed for a computer to execute this program would be minimal (barring any fancy recursion), and has only one explicit process: to count numbers. However, a more complex example would be a program that receives user input, performs extensive calculations, updates a database, and returns a result to the user. This type of program would be more aptly classified as a computer application, in that it contains many different components and processes, versus just counting numbers. Many of these processes can happen at the same time, and there can be duplicate processes in order to divide the work needed to be done. We can therefore think of a computer process as a program that is being executed, from which applications are built.

1.2 Threads

Each computer process requires resources in order to run, including the CPU, RAM, and hard disk storage. The operating system allocates these resources using threads.

1.3 CPU Scheduling

Any operating system that claims to be a multiprocessing operating system must have a way of managing all of the running processes and allocating system resources to them. One of the most critical resources is the CPU's time. Modern CPU's have multiple physical cores, which can execute their own process. However, in order to run even more processes at the same time, the operating system can quickly switch between different processes, giving them time to use the CPU, and creating the appearance of running many programs simultaneously.

2 Linux

2.1 Processes

Each process (also known as a task) in Linux is represented by a task_struct data structure, which is naturally large and complex. Outlined below are the main basic functional areas and fields:

- State - Can be one of following: running, waiting, stopped, zombie
- Scheduling Information - Data regarding process precedence in the system
- Identifiers - Simply a unique number for the process
- Inter-Process Communication - Standard *nix information for processes to share data
- Links - Pointers to parent, sibling, and child processes
- Timers - Data about when the process was created and how much CPU time it has consumed
- File System - Pointers to working directories and other files
- Virtual Memory - How much virtual memory assigned to the process
- Processor Specific Context - Information that is needed to restart the process in the same state it was in

3 FreeBSD

3.1 Processes

4 Windows

4.1 Processes