CS 452 Lab: Week #7
System Resource Utilization

Overview

Resource management is a primary responsibility of the operating system. Mismanagement of resources (including CPU cycles) can cause inefficiencies, and malfunctioning drivers and rogue processes can be responsible for a denial of service to other users. The purpose of this assignment is to become familiar with several methods used to determine the limits placed on system resources. Both static and dynamic mechanisms will be introduced.

Hand-in:

- A word document containing the requested information
- Any requested screenshots (embedded in the word document in the appropriate place).
- Program source code as a separate file(s). No zip file(s).

System Resource Limitations

When working with user or system resources such as processes, files, pipes, shared memory and semaphores, it is prudent to be aware of any system limits placed upon ownership of these resources. This lab introduces several different methods for discovering the values of system limits for specific machine/operating system combinations. Note: these methods are translatable to any operating system.

Static
In this context, static means "not active" (i.e., the limit is determined via a simple lookup). As usual, the first place to start is the man pages and associated include files (*always* a good idea for a developer). General information about system resources can be found in `limits.h` and `sys/param.h` but it may not be machine specific. Be sure to examine `bits/posix1_lim.h` and `bits/posix2_lim.h` in the event your specific system configuration supports the Posix standard for that resource. Note that limits obtained statically may not be reflective of actual runtime values; the source code defaults may have been overridden during

compilation or changed by a systems administrator in the course of a local installation. Please note the location (and existence) of these files may differ from system to system

Dynamic
Many user limits can be obtained dynamically via the **`getrlimit()`** system call and the **`sysconf()`** function. System limits can be determined via the **`sysinfo()`** system call. All of them work by querying the kernel for the actual maximum value of a resource. They pass a flag to the system that specifies the resource of interest; the appropriate value is then either returned directly or written into a structure for subsequent access. Read the man pages for details on how to use these functions at run-time (i.e. from within an executing program).

Empirical
Finally, some limits are not defined in any include file, or available via query. In this case, it is still possible to determine a maximum value experimentally. Recall that all system calls return error codes indicating their success/failure. Obviously, a carefully-designed program could "push the limit" and use the return value to determine the point at which a system call fails, and hence the limit of the resource it is requesting. For example, the **`shmget()`** function takes a parameter specifying the size of the shared memory segment desired. It returns a segment ID if successful, or -1 on failure. The size argument can be manipulated (continuously increased) within your test program to determine the maximum-sized shared memory segment allowed on your system.

Problem Specification

Use the specified method to determine the values for the following system resources. Indicate what machine you are using to perform your tests and provide details regarding your method (i.e. name of include file, system call used, etc.)

1. Maximum number of semaphores per process (static)
2. Maximum value of a counting semaphore (static)
3. Maximum value of a counting semaphore (empirical)
4. Maximum size of a shared memory segment (empirical)
5. Page size in bytes (dynamic)
6. Physical pages in a system (dynamic)
7. Maximum number of processes per user (dynamic)
8. Maximum filesize in bytes (dynamic)
9. Maximum number of open files, hard limit (dynamic)
10. Maximum number of open files, soft limit (dynamic)
11. Clock resolution in milliseconds (dynamic)

Submit the source code you used to determine any of the above methods in a separate file named appropriately (e.g., ‘numberThree.c’, ‘numberFour.c’).

Embed a screenshot of the execution for each program in a word document.