‘Implementation of Semaphores in Linux’
Stage-One Report

Jojumon Kavalan (04305401)
Joy Krishnan Menon (04305402)
Samveen Saif Gulati (04305905)

11 September 2004

Contents

1 Introduction

1.1 Purpose

This is a stage-1 report on the project 'Implementation of Semaphores in Linux', submitted towards the partial fulfillment of requirements of cs701 course as part of the M.Tech programme of the Computer Science and Engineering department of Indian Institute of Technology, Bombay.

1.2 Scope

This stage-1 report for the project broadly addresses the following:

1. The System V IPC Semaphore specification.
2. Shortcomings of the current Semaphore implementation.
3. Approaches towards the implementation of a lighter-weight Semaphore in the Linux kernel.
4. Profiling semaphore usage by applications programs, and
5. Performance evaluation of new implementation.

1.3 Project overview

Semaphores are a mechanism used for interprocess communication (IPC) and mutual exclusion in case of shared resources. An application program can request the operating system for a set of semaphores and use the semaphores to achieve IPC and mutual exclusion. See section 2.1.

The Linux Operating System includes an implementation of the System V IPC Semaphore specification. Due to certain reasons, the current implementation is heavyweight in memory-usage and performance - especially due to heavy context switching.

This project is an exercise into the detailed design, implementation, and profiling of a semaphore that is lightweight in terms of memory-usage and performance.

2 Semaphores

2.1 Semaphore Specification

Semaphores are one of the three inter-process communication mechanisms available in the Linux Kernel, the other two being Message Queuing and Shared Memory. The mentioned specification requires semaphores to be implemented not as single semaphores, but as semaphore sets using the following two data structures:

semid_ds One semid data structure for each set of semaphores in the system.

sem In the semid_ds structure, there is a pointer to the base of the semaphore array. Members of the semaphore array are of the sem structure type.

2.2 Existing Linux Implementation

The Linux kernel maintains a special internal data structure for each semaphore set which exists within its addressing space. This structure is defined in linux/sem.h, code is in linux/sem.c, and other machine specific conditions are handled in one of several files named semaphore.c.

The current implementation is heavyweight in memory-usage and performance. Possible reasons could be the usage of set of semaphores instead single semaphores, or shortcomings in the implementation of algorithms, or uneven design causing heavy context switching.

A conclusive answer to the same will be reached shortly after a detailed study of the relevant sources.

3 Lightweight Semaphore: Approaches

We propose to implement a lightweight semaphore for Linux. The implementation would result into high performance semaphore which can function using less memory and, function better in terms of computation required.

The goals of the project are,

1. Study of existing code and algorithms used for semaphore operations.
2. Design and implementation of lightweight semaphores using any of the identified approaches.
3. Implementation of tools to profile usage of semaphores by systems, and
4. Implementation of tools to test completeness, and demonstrate the improvement achieved.

3.1 Applying existing lightweight implementations

One possible approach would be to study existing lightweight implementations of Semaphores in other systems, and apply these implementations within the Linux framework. A few lightweight implementations of semaphores have been proposed, and even implemented.

One notable implementation is semaphores for embedded systems [2]. This design may be exclusively for smaller systems with limited functional requirements, and may not be directly applicable for Linux.

Another one is the semaphore implementation for the General Parallel File System (GPFS) [1]. A closer study of the same will be needed to know it's applicability to the current problem.

3.2 Modification of the existing algorithms

Alternately, we could modify the existing structures and algorithms to make it lightweight.

In the first stage of this project we would study the existing implementation of semaphores in detail. The study would focus on several aspects which include the representation of the semaphore, efficiency analysis for the existing implementation of procedures for semaphore operations, and analysis of different IPC mechanisms used in the implementation of semaphores.

The study and analysis are aimed at improving the existing implementation wherever possible. This would include:

1. Redesign of semaphore structure.
2. Alternative implementation of semaphore operations using algorithms more efficient in terms of time complexity.
3. Reduce the amount of system/IPC resources used for semaphore creation and operations.

3.3 Single semaphores instead of sets

Adhering to the System V specifications for semaphores, semaphore is implemented as a set of semaphores.

To elaborate, when a program request for a single semaphore the system will provide it with a set of semaphores which contain one semaphore. We observe that most application programs require single semaphores rather than a set of semaphores. So the current implementation leads to certain overhead in managing set of semaphores when single semaphore is required.

We propose to overcome this limitation by designing "single semaphore" which is standalone rather than being associated with a set. Then the programs can request for "single semaphore" when they require a single semaphore, use of which would improve the performance of the system.

4 Other Components

This section provides a functional description of additional components that will be part of the final solution package.

4.1 Profiling Semaphore Usage

The Linux sources include a couple of configurable tools, semtool and semstats, that can provide some minimal usage and monitoring statistics for semaphores. We intend to extend the same, or create new wrapper applications that will:

1. Profile all statistics with respect to the usage of semaphores by candidate applications,

2. Qualify the complexity of overhead that the usage of semaphores adds to these applications.

4.2 Test-suite for Verification

Also part of the deliverables will be an application to test exhaustively the new semaphore implementation. A survey of already available test suites for a similar project may be conducted. The scope of the application will be:

1. Exhaustive testing of all interfaces provided by the new implementation. (functional testing)

2. Testing of the new implementation with respect to stability under extreme loads (stress testing).

5 Project Deliverables

The project deliverables will exclusively include the following:

1. Complete Design of the lightweight semaphore object.
2. Source code (in C) of implementation of the lighter-weight semaphore.
3. Linux kernel integrated with lightweight semaphore implementation.
4. Test-suite and profiling tools to demonstrate the improvement in new implementation.
5. Report on complete research, design and development activity.

6 Development Environment

The Primary development environment for the project in it's implementation will be C language programming with the gcc compiler on GNU/Linux platform. Development will require semaphore implementation sources of Linux, and the complete Linux kernel source for integration and testing purposes. Other Linux kernel development utilities will also be needed.

7 Conclusion

The existing implementation of semaphores in Linux is heavyweight in memory usage and computational performance. The performance of semaphores can be improved by modifying the existing algorithms or through a radical design. This document has proposed research and development to implement efficient lightweight semaphores.

References

[1] Frank Schmuck and Roger Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proc. of the First Conference on File and Storage Technologies (FAST)*, pages 231–244, January 2002.

[2] K. Zuberi and K. Shin. An efficient semaphore implementation scheme for small-memory embedded systems. pages 25–37.