Kernel Synchronization in Linux.

Professor: M.Khalil

Students: K. Bean and W. Jaffal

CSE 8343

1. Abstract

This survey paper introduces a kernel synchronization technique implemented by Linux, a very popular and relatively new Unix-like operating system. The presented work considers uniprocessor and multiprocessor environments supported by Linux.

2. Introduction

As part of his computer science project at the University of Helsinki (1991), Linus Torvalds developed Linux to serve as an operating system for IBM-compatible personal computers (based upon the Intel 80386 microprocessor) [ME02]. This work was presented in 1997 as a Master's thesis entitled "Linux, a Portable Operating System" by L. Torvalds [LinTor97]. Modern versions of Linux are now available on other architectures such as Alpha, SPARC, Motorola MC680x0, PowerPC, and IBM System/390. In contrast with the Windows-based operating systems, Linux is not a commercial operating system. Its source code under GNU General Public License is available to the public and is downloadable via http://www.kernel.org/.

In the "Kernel Control Path" section of this paper, problems that could occur during kernel control path interleaving are investigated. In "Synchronization Techniques," different techniques to avoid race conditions are considered. In "Linux Multiprocessor Architecture," the SMP

multiprocessor architecture is compared with other architectures. At the end of this section, process synchronization on hardware level is considered. In “The Linux/SMP Kernel”, Linux support of multiprocessor environment is described.

3. Kernel Control Path

Kernel processes execute due to the following reasons [BC00]:

- A process executing in User Mode caused some exception (i.e., divide by zero);
- An external device sent a signal to a Programmable Interrupt Controller using an IRQ, with a corresponding interrupt being enabled.

According to [BC00], a *kernel control path* is the sequence of instructions executed in Kernel Mode to handle a kernel request. A kernel control path can handle various situations: system calls, exceptions or interrupts. When one of the following conditions occurs, the CPU does not execute a kernel control path sequentially:

- A context switch occurs.
- An interrupt occurs. CPU starts processing another kernel control path.

A race condition (outcome of some computation depends on how two or more processes are scheduled) can occur not only when a user process is preempted by a higher priority process, but when a CPU interleaves a kernel control path.

4. Synchronization Techniques

4.1 Non-preemptability of a Process in Kernel Mode

According to [BC00], the Linux kernel is not preemptive; a higher-priority process cannot preempt a process running in Kernel Mode. The following assertions always hold true for a running process in Kernel Mode within Linux:

- The process cannot be replaced by another process, except when the former voluntary relinquish control of the CPU.
- Interrupt or exception handling can preempt a running process; however, when an interrupt handle terminates, the process resumes.
- Only a kernel control path can interrupt another kernel control path.

4.2 Atomic Operation

As defined in [BC00], an *atomic operation* is an operation performed by executing a single assembly language instruction; therefore, this instruction cannot be interrupted in the middle.

C language operations such as a = a + 1 or a++ could be implemented by a compiler as single atomic operations or it could be compiled into more complicated code. To guarantee that some operations compiled every time as single atomic operations, Linux kernel provides special functions such as: