Lecture - 8 - Linux (not given as a separate lecture during 2014)

TSEA81

Computer Engineering and Real-time Systems

Linköping University
Sweden

This document is released - 2013-12-16 - first version (new homepage)

Author - Ola Dahl

Linux

- An operating system kernel
- An operating system
- A distribution, containing operating system, file system(s), many applications, tools, graphics, communication
- An embedded operating system, used in communication systems, and in industrial systems
- A Unix-like system, using GNU software and licensing, and open source development model

Linux kernel

- Monolithic
- Loadable kernel modules
- Preemptive multitasking
- Memory management (virtual memory, multiple address spaces)
- Threading

First announcement

Announced August 26, 1991, by Linus Torvalds

- *"Hello everybody out there using minix - I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready"*

Timeline

- 1991 - first announcement
- 1992 - Linux 0.12 - released under GPL
- 1994 - Linux 1.0
- 1994 - first attempt to an ARM port
- 1995 - Linux 1.2 - supporting Alpha, Sparc, and MIPS
- 1999 - Linuxdevices.com founded
- 2001 - Linux 2.4
- 2003 - Linux 2.6
- 2011 - Linux 3.0
- 2012 - Linux 3.6.8

Timeline - others

- 1970 - Unix
- 1969-1973 - C
- 1976 - first version of EMACS (Editor MACroS for the TECO editor)
- 1984 - Apple Macintosh
- 1985 - Intel 80386
- 1985 - GNU Manifesto
- 1985 - Windows 1.0
- 1992 - Windows 3.1 (protected mode but single address space)
- 2002 - Announcement of release of Hurd
- 2012 - "The GNU Hurd is under active development. Because of that, there is no stable version"

Number of lines of C-code, using

```
wc -l `find . -name "*.c"`
```

- Linux 3.1.12.1 - 13107295
- Linux 1.0 - 141361
- FreeRTOS (includes TCP/IP, demos, etc.) - 788751
- MicroC/OS-III - RTOS core, 14310
- Simple_OS - 2822

File system structure

UNIX file system structure, contains e.g.

- */bin* - commands, e.g. *cat*, *ls*
- */sbin* - system-oriented commands, e.g. *insmod*
- */boot* - boot loader files
- */etc* - configuration files
- */home* - user directories
- */edu* - student directories
- */usr/bin* - more commands, e.g. *ssh*, *which*
- */usr/include* - standard include files
- */var* - files that change during run-time

Programming

- C-programming (of course)
- Many other languages
- Process programming, e.g. process communication, sockets, pipes, shared memory
- Example book - http://www.advancedlinuxprogramming.com/

Libraries and system calls

- A *system call* is an activation of a service in the operating system
- A system call changes the processor mode, from *user mode* to a *privileged* mode. This mode is often referred to as *kernel mode*
- A system call can be implemented using a special processor instruction (e.g. *software interrupt*)
- A library is a set of routines used by a program

Libraries and system calls

A program executes in *user mode*. A program may

- Call a library function, e.g. *printf*
- The function printf may issue a system call, e.g. *write()*, which then constitutes the entry point to the operating system
- UNIX/Linux man pages list system calls in section 2 and library functions in section 3 (and general commands in section 1)

A program executing in user mode uses addresses assigned to it. These addresses are referreed to as *user space*. The corresponding addresses when executing in kernel mode are referred to as *kernel space*.

Programs and processes

- Process - an instance of a program in execution
- Parent and child relation
- Threads - execution flows inside a process

Linux schedules *tasks*, that may share resources, e.g. address space.

- Threads are implemented as tasks that share address space
- Processes have separate address spaces
- During task creation, it can be decided which resources to share

Tasks are managed (created, scheduled, and killed) by the Linux kernel.

Device drivers

- Device drivers are used for communication with *devices*
- Device drives also communicate with *programs*
- Device drivers execute in *kernel mode*, using addresses in *kernel space*
- Device drivers can implement system calls, e.g. *read*, *write*, and *ioctl*
- Device drivers can manage interrupt handlers
- Comprehensive book about Linux device drivers: http://lwn.net/Kernel/LDD3/

Process start and termination

- Process 0 - the idle process (a kernel thread)
- Process 1 - the *init* program
- Process creation using *clone* and *fork* system calls
- Process termination using *_exit* system call

Process switch

- Saving and restoring hardware context
- Registers are saved on the *kernel mode stack* of each process, and in its *process descriptor*
- A process descriptor (the *task_struct* struct in the kernel), includes process state, process id (pid), reference to the kernel mode stack, and much more
- Process descriptors can be stored in *linked lists*
- A process switch also involves *memory management*, since address spaces need to be changed - page tables for the new process need to replace page tables for the old process

Process descriptor link:
http://lxr.linux.no/linux+v3.6.6/include/linux/sched.h#L1234

Scheduling in Linux

- Processes are preemptable - a process may be preempted as a consequence of an interrupt, or when its time quantum has expired
- Also the Linux kernel is preemptable
- Processes are scheduled according to a *scheduling policy*
- There are two *real-time scheduling policies* - called *SCHED_FIFO* (similar to priority-based RTOS-scheduling) and *SHED_RR* which is *SCHED_FIFO* with time-slicing
- There is one normal scheduling policy, called *SHED_NORMAL* which is the time sharing CFS method

Scheduling in Linux

- The real-time scheduling policies use static priorities (assigned in a dedicated real-time priority range)
- Static priority and scheduling policy can be modified using system calls, e.g. *nice()* for changing the static priority, and *sched_setscheduler()* for changing scheduling policy

Real-time Linux

Variants of Linux, adapted for better real-time properties

- Linux 2.6 - *CONFIG_PREEMPT* configuration option allows process to be preempted even if they are executing a system call
- Linux 2.6 series - *O(1) scheduler* - can perform scheduling in constant time, and *CFS* scheduler - improved responsiveness for interactive tasks
- CONFIG_PREEMPT_RT patch - https://rt.wiki.kernel.org/ - allows nearly all of the kernel to be preempted
- Thin kernel approach - run Linux as a low priority task in a thin kernel, which runs directly on the hardware (as an RTOS) - examples are RTLinux, RTAI, Xenomai