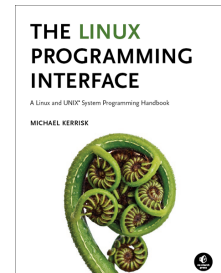


Linux/UNIX System Programming Essentials

Course code: M7D-SPESS01

This one-day course provides an introduction to a range of key system programming concepts: file I/O using system calls, signals, processes, and process lifecycle (*fork()*, *execve()*, *wait()*, *exit()*).



Audience and prerequisites

The audience for this course includes programmers developing and porting system-level applications for Linux and UNIX systems, embedded application developers, security engineers, site reliability engineers, and DevOps engineers.

To get the most out of the course, participants should have:

- Good reading knowledge of the C programming language
- Solid programming experience in a language suitable for completing the course exercises (e.g., C, C++, D, Go, Rust, or Python)
- Knowledge of basic UNIX/Linux shell commands

Previous system programming experience is *not* required.

Related courses

This course provides the background necessary for a number of other courses:

- *Linux Security and Isolation APIs*, M7D-SECISOL02
- *Linux/UNIX Threads and IPC Programming*, M7D-TIPC01
- *Linux/UNIX IPC Programming*, M7D-IPC02

The *Linux/UNIX System Programming Fundamentals* (M7D-SPINTRO01) course covers the same topics, but in greater depth.

Course duration and format

One day, with up to 40% devoted to practical sessions.

Course materials

- Course books (written by the trainer) that include all slides and exercises presented in the course
- An electronic copy of the trainer's book, *The Linux Programming Interface*
- Numerous example programs written by the course trainer

Course inquiries and bookings

For inquiries about courses and consulting, you can contact us in the following ways:

- Email: training@man7.org
- Phone: +49 (89) 2155 2990 (German landline)

Prices and further details

For course prices and further information, please visit the course web page, <http://man7.org/training/spess/>.

About the trainer



Michael Kerrisk has a unique set of qualifications and experience that ensure that course participants receive training of a very high standard:

- He has been programming on UNIX systems since 1987 and began teaching UNIX system programming courses in 1989.
- He is the author of *The Linux Programming Interface*, a 1550-page book acclaimed as the

definitive work on Linux system programming.

- He is actively involved in Linux development, working with kernel developers on testing, review, and design of new Linux kernel-user-space APIs.
- Since 2004, he has been the maintainer of the Linux *man-pages* project, which provides the manual pages documenting Linux system calls and C library APIs.

Linux/UNIX System Programming Essentials: course contents in detail

Topics marked with an asterisk (*) are optional, and will be covered as time permits

1. Course Introduction

2. Fundamental Concepts

- System calls and library functions
- Error handling
- System data types
- Notes on code examples

3. File I/O and Files

- File I/O overview
- *open()*, *read()*, *write()*, and *close()*
- Retrieving file information: *stat()*

4. Processes

- Process IDs
- Process memory layout
- Command-line arguments
- The environment list
- The */proc* filesystem

5. Signals

- Overview of signals
- Signal dispositions
- Useful signal-related functions
- Signal handlers
- Signal sets, the signal mask, and pending signals
- Designing signal handlers

6. Process Lifecycle

- Creating a new process: *fork()*
- Process termination
- Monitoring child processes
- Orphans and zombies
- The *SIGCHLD* signal
- Executing programs: *execve()*

7. System Call Tracing with *strace* (*)

- Getting started
- Tracing child processes
- Filtering *strace* output

The following are a few of the **other courses taught by Michael Kerrisk**. Custom courses are also available upon request. Further details on these and other courses can be found at <http://man7.org/training/>. For course inquiries please email training@man7.org or phone +49 (89) 2155 2990 (German landline).

Linux/UNIX System Programming

Course code: M7D-LUSP01 (5 days)

Intended for a wide audience, including system programmers, embedded developers, devops engineers, and security engineers, this course provides a deep understanding of the operating system architecture and low-level interfaces required to build system-level applications on Linux and UNIX systems ranging from embedded processors to enterprise servers. Detailed presentations coupled with many carefully designed practical exercises provide participants with the knowledge needed to write complex system, network, and multithreaded applications. Topics covered include file I/O; files, directories, and links; signals; processes; process creation and termination; program execution. multithreaded programming with POSIX threads; IPC (pipes, FIFOs, shared memory, semaphores, local and network IPC with sockets); and I/O multiplexing (*poll()*, *select()*, and *epoll()*).

Linux Security and Isolation APIs

Course code: M7D-SECISOL02 (4 days)

Covering topics including control cgroups (cgroups v1 and v2), namespaces (with a deep dive into user namespaces), capabilities, and seccomp (secure computing), this course

provides a deep understanding of the low-level Linux features used to design, build, and troubleshoot container, virtualization, and sandboxing frameworks.

Building and Using Shared Libraries on Linux

Course code: M7D-SHLIB03 (2 days)

This course provides a thorough understanding of the process of designing, building, and using shared libraries on Linux. Topics covered include: fundamentals of library creation and use; shared library versioning; symbol resolution; library search order; executable and linking format (ELF); dynamically loaded libraries; controlling symbol visibility; and symbol versioning.

Linux/UNIX IPC Programming

Course code: M7D-IPC02 (3 days)

This course provides a thorough introduction to the interprocess (IPC) techniques that Linux and UNIX systems provide for use by user-space programs. Using these techniques (pipes, FIFOs, shared memory, semaphores, local and network IPC with sockets, and I/O multiplexing) allows the creation of complex multiprocess applications that coordinate their actions and exchange information with each other.