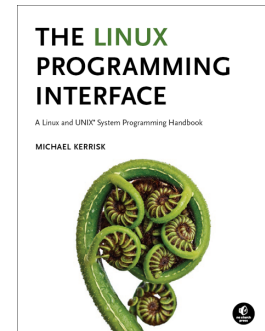


System Programming for Linux Containers

Course code: M7D-SPLC02

This course provides a deep understanding of the Linux technologies (namely, set-UID/set-GID programs, capabilities, namespaces, cgroups, and seccomp) used to implement container, virtualization, and sandboxing systems. (These are the technologies used to build systems such as Docker, LXC, Firejail, and Flatpak.) The course also provides an understanding of the core APIs used to build system-level applications that run on such systems. Detailed explanations and carefully designed practical exercises provide participants with the knowledge needed both to troubleshoot container and sandboxing systems and to write complex applications that run on those systems.



Audience and prerequisites

The audience for this course includes designers, developers, and DevOps who are building, troubleshooting, and administering container and sandboxing systems, as well as designers and developers who are implementing applications to run on such systems.

Participants should have a good reading knowledge of the C programming language and some programming experience in a language suitable for completing the course exercises (e.g., C, C++, Go, Rust). (Note, however, that, except on the first day of the course, most of the course exercises do not require writing programs.)

Previous system programming experience is *not* required.

Related courses

This course is equivalent to the combination of the following two courses:

- *Linux/UNIX System Programming Essentials*, M7D-SPESS01
- *Linux Security and Isolation APIs*, M7D-SECISOL02

Course duration and format

Five days, 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, dates, and further details

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

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 has been actively involved in Linux development, working with kernel developers on testing, review, and design of new Linux kernel-user-space APIs.
- Since 2000, he has been involved in the Linux *man-pages* project, which provides the manual pages documenting Linux system calls and C library APIs, and was the project maintainer from 2004 to 2021.

System Programming for Linux Containers: course contents in detail

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

1. **Course Introduction**
2. **Fundamental Concepts**
 - Error handling
 - System data types
 - Notes on code examples
3. **File I/O**
 - File I/O overview
 - *open()*, *read()*, *write()*, and *close()*
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
 - 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
8. **Security and Isolation APIs Overview (*)**
 - Sandboxing
 - Containers
9. **Classical Privileged Programs**
 - A simple set-user-ID program
 - Saved set-user-ID and saved set-group-ID
 - Changing process credentials
 - A few guidelines for writing privileged programs
10. **Capabilities**
 - Process and file capabilities
 - Permitted and effective capabilities
 - Setting and viewing file capabilities
 - Capabilities-dumb and capabilities-aware applications
 - Text form capabilities
 - Capabilities and *execve()*
 - The capability bounding set
 - Inheritable capabilities
 - Ambient capabilities
 - Capabilities and UID transitions
11. **Capabilities: Further Topics**
 - Capabilities, UID 0, and *execve()*
 - Making a capabilities-only environment: *securebits* (*)
 - Programming with capabilities (*)
12. **Namespaces**
 - An example: UTS namespaces
 - Namespaces commands
 - Namespaces demonstration (UTS namespaces)
 - Namespace types and APIs
 - Namespaces, containers, and virtualization
13. **Mount Namespaces and Shared Subtrees**
 - Mount namespaces
 - Shared subtrees
 - Bind mounts
14. **PID Namespaces**
 - PID namespaces
15. **Other Namespaces**
 - IPC namespaces
 - Time namespaces
 - Cgroup namespaces
 - Network namespaces
16. **Namespaces APIs**
 - API Overview
 - Creating a child process in new namespaces: *clone()*
 - */proc/PID/ns*
 - Entering a namespace: *setns()*
 - Creating a namespace: *unshare()*
 - PID namespaces idiosyncrasies
 - Namespace lifetime (*)
17. **User Namespaces**
 - Overview of user namespaces
 - Creating and joining a user namespace
 - User namespaces: UID and GID mappings
 - User namespaces, *execve()*, and user ID 0
 - Accessing files; file-related capabilities (*)
 - Security issues
 - Use cases
 - Combining user namespaces with other namespaces
18. **User Namespaces and Capabilities**
 - User namespaces and capabilities
 - What does it mean to be superuser in a namespace?
 - Discovering namespace relationships
 - User namespace "set-UID-root" programs (*)
 - Namespaced file capabilities (*)
19. **Mount Namespaces: Further Details (*)**
 - Peer groups
 - Private mounts
 - Slave mounts
 - Unbindable mounts
 - Mounting a container filesystem
20. **Seccomp**
 - Introduction and history
 - Seccomp filtering and BPF
 - The BPF virtual machine and BPF instructions
 - BPF filter return values
 - BPF programs
 - Checking the architecture
 - Productivity aids (*libseccomp* and other tools)
 - Applications and further information
21. **Seccomp: Further Details (*)**
 - Caveats
 - Discovering the system calls made by a program
 - Further details on seccomp filters
 - Extended BPF (eBPF)
 - Further details on BPF programs
 - Recent seccomp features
 - Audit logging of filter actions
22. **Cgroups: Introduction**
 - Preamble
 - What are control groups?
 - An example: the pids controller
 - Creating and destroying cgroups
 - Populating a cgroup
 - Enabling and disabling controllers
23. **Cgroups: A Survey of the Controllers**
 - The cpu, memory, freezer, and pids controllers
 - Other controllers
24. **Cgroups: Advanced Features**
 - Cgroup namespaces
 - Release notification (*cgroup.events* file)
 - Delegation
25. **Cgroups: Thread Mode (*)**
 - Overview of thread mode
 - Creating and using a threaded subtree
26. **Cgroups Version 1 (*)**
 - Cgroups v1: hierarchies and controllers
 - Cgroups v1: populating a cgroup
 - Cgroups v1: release notification
 - Cgroups v1: delegation
 - Problems with cgroups v1; rationale for v2