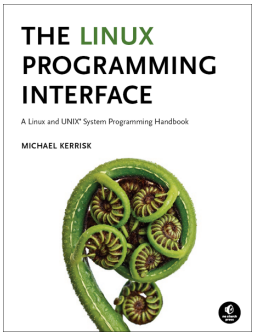
man7.org
Training and Consulting

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.

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.

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
- A copy of the trainer's book, *The Linux Programming Interface*
- A source code tarball containing more than 35,000 lines of example code written by the 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 widely 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 the Linux kernel–user-space and GNU C library APIs.

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**
   - Process vs kernel: different views of the world
   - 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()*
   - The file offset and *lseek()*
   - Relationship between file descriptors and open files
   - Duplicating file descriptors
   - File status flags (and *fcntl()*)
4. **Processes**
   - Process IDs
   - Process memory layout
   - Command-line arguments
   - The environment list
   - Process credentials
   - The `/proc` filesystem
5. **Signals: Introduction**
   - Overview of signals
   - Signal dispositions
   - Signal handlers
   - Useful signal-related functions
   - Signal sets, the signal mask, and pending signals
   - Designing signal handlers
6. **Signals: Signal Handlers (*)**
   - Reentrant and async-signal-safe functions
   - Interruption and restarting of system calls
   - `SA_SIGINFO` signal handlers
   - The signal trampoline
7. **Process Lifecycle**
   - Introduction
   - Creating a new process: *fork()*
   - Process termination
   - Monitoring child processes
   - Orphans and zombies
   - The `SIGCHLD` signal
   - Executing programs: *execve()*
8. **System Call Tracing: *strace* (*)**
   - Tracing child processes
   - Filtering *strace* output
   - Further *strace* options
9. **Security and Isolation APIs Overview (*)**
   - Sandboxing
   - Containers
10. **Privileged Programs**
    - Process credentials
    - Set-user-ID and set-group-ID programs
    - Changing process credentials
    - A few guidelines for writing privileged programs
11. **Capabilities**
    - Process and file capabilities
    - Setting and viewing file capabilities
    - Text form capabilities
    - Capabilities and *execve()*; further capability sets
    - Ambient capabilities
12. **Capabilities: Further Topics**
    - Root, UID transitions, and capabilities
    - Making a capabilities-only environment: securebits (*)
    - Programming with capabilities (*)
13. **Namespaces**
    - Namespace types
    - Mount namespaces
    - UTS, IPC, cgroup, and network namespaces
    - PID namespaces
14. **Namespaces APIs**
    - API Overview
    - Creating a child process in a new namespace: *clone()*
    - `/proc/PID/ns`
    - Entering a namespace: *setns()*
    - Creating a namespace: *unshare()*
    - PID namespaces idiosyncrasies
    - *ioctl()* operations
    - Namespace lifetime
15. **User Namespaces**
    - Overview of user namespaces
    - Creating and joining a user NS
    - User namespaces: UID and GID mappings
    - User namespaces, *execve()*, and user ID 0
    - Security issues
    - Use cases
    - Combining user namespaces with other namespaces
16. **User Namespaces and Capabilities**
    - User namespaces and capabilities
    - What does it mean to be superuser in a namespace?
    - User namespace "set-UID-root" programs (*)
    - Namespaced file capabilities (*)
17. **Mount Namespaces and Shared Subtrees (*)**
    - Mount namespaces
    - Shared subtrees
    - Bind mounts
    - Peer groups
    - Private mounts
    - Slave mounts
    - Unbindable mounts
18. **Seccomp**
    - Introduction and history
    - Seccomp filtering and BPF
    - The BPF virtual machine and BPF instructions
    - Checking the architecture
    - BPF filter return values
    - BPF programs
    - Discovering the system calls made by a program
    - Audit logging of filter actions
    - Caveats
    - Productivity aids (*libseccomp* and other tools)
19. **Cgroups (Control Groups)**
    - Introduction to cgroups v1 and v2
    - Cgroups v1: hierarchies and controllers
    - Cgroups v1: populating a cgroup
    - Cgroups v1: release notification
    - Cgroups v1: a survey of the controllers
    - Cgroups `/proc` files
20. **Cgroups Version 2**
    - Problems with cgroups v1; rationale for v2
    - Cgroups v2 controllers
    - Enabling and disabling controllers
    - Organizing cgroups and processes
    - Release notification (`cgroup.events` file)
    - Delegation
21. **Cgroups v2 Thread Mode (*)**
    - Overview of thread mode
    - Creating and using a threaded subtree