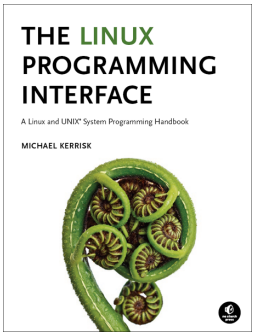
man7.org
Training and Consulting

System Programming for Linux Containers

Course code: M7D-SPLC02

This course provides a deep understanding of the Linux infrastructure used to build container systems such as Docker and LXC, coupled with an understanding of the APIs used to build system-level applications that run inside containers. Detailed explanations and carefully designed practical exercises provide participants with the knowledge needed to troubleshoot and administer container-based systems and to write complex applications that run inside Linux containers.



Audience and prerequisites

The audience for this course includes designers and developers who are building, troubleshooting, and administering system-level applications for Linux-based container 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 50% devoted to practical sessions.

Course materials

- Two 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 around 30,000 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/sys_prog_lxcon/*.

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. **Fundamental Concepts**
   - System calls and library functions
   - Error handling
   - System data types
   - Notes on code examples
2. **File I/O and Files**
   - File I/O overview
   - *open()*, *read()*, *write()*, and *close()*
   - Relationship between file descriptors and open files
   - Duplicating file descriptors
   - File status flags (and *fcntl()*)
   - Retrieving file information: *stat()*
3. **Processes**
   - Process IDs
   - Process memory layout
   - Process credentials
   - The `/proc` filesystem
4. **Signals**
   - Overview of signals
   - Signal dispositions
   - Signal handlers
   - Useful signal-related functions
   - Signal sets, the signal mask and pending signals
   - Designing signal handlers
   - Reentrant and async-signal-safe functions
   - Interruption and restarting of system calls
5. **Process Lifecycle**
   - Introduction
   - Creating a new process: *fork()*
   - Process termination
   - Monitoring child processes
   - Orphans and zombies
   - The SIGCHLD signal
   - Executing programs: *execve()*
6. **System Call Tracing with *strace* (*)**
   - Getting started
   - Tracing child processes
   - Filtering *strace* output
   - Further *strace* options
7. **Cgroups**
   - 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
8. **Cgroups Version 2**
   - Problems with cgroups v1; rationale for v2
   - Cgroups v2 controllers
   - Enabling and disabling controllers
   - Organizing cgroups and processes
   - Release notification
   - Delegation
   - Cgroups Version 2 Thread Mode (*)
9. **Seccomp**
   - Introduction and history
   - Seccomp filtering and BPF
   - The BPF virtual machine and BPF instructions
   - Checking the architecture
   - BPF filter return values
   - BPF programs
   - Recent seccomp features
   - Discovering the system calls made by a program
   - Audit logging of filter actions
   - Further details on seccomp filters
   - Caveats
   - Productivity aids
   - Applications and further information
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**
    - Overview
    - Process and file capabilities
    - Setting and viewing file capabilities
    - Text form capabilities
    - Capabilities and *execve()*
    - Ambient capabilities
    - Summary remarks
    - Root, UID transitions, and capabilities (*)
    - Making a capabilities-only environment: securebits (*)
    - Programming with capabilities (*)
12. **Namespaces**
    - Overview
    - Namespace types
    - Mount namespaces
    - UTS, IPC, cgroup, and network namespaces
    - PID namespaces
    - User namespaces (introduction)
13. **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
14. **User Namespaces**
    - Introduction to user namespaces
    - Creating and joining a user NS
    - User namespaces: UID and GID mappings
    - User namespaces, *execve()*, and user ID 0
    - User namespaces and capabilities
    - Combining user namespaces with other namespaces
    - User namespaces and capabilities revisited
    - Security issues
    - Use cases
15. **Mount Namespaces and Shared Subtrees (*)**
    - Mount namespaces
    - Shared subtrees
    - An aside: bind mounts
    - Peer groups
    - Private mounts
    - Slave mounts
    - Unbindable mounts