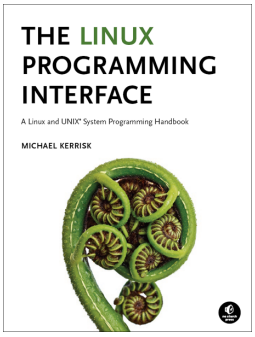
man7.org
Training and Consulting

Linux/UNIX System Programming

Course code: M7D-SP01

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. The course dives into many specifics of the Linux system, but makes careful and frequent reference to the POSIX standard, so that it is also valuable to developers working on other UNIX systems.



Audience and prerequisites

The audience for this course includes programmers developing and porting system-level and network 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.

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/*.

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.

Linux/UNIX System Programming: 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**
   - File I/O overview
   - *open()*, *read()*, *write()*, *close()*
   - The file offset and *lseek()*
   - Atomicity
   - Relationship between file descriptors and open files
   - Duplicating file descriptors
   - File status flags (and *fcntl()*)
4. **File I/O Buffering**
   - Kernel buffering
   - User-space (*stdio*) buffering
   - Controlling kernel buffering
5. **Files**
   - Inodes
   - Retrieving file information: *stat()*
   - File mode
   - Changing file attributes
6. **Directories and Links (*)**
   - Directories and (hard) links
   - Symbolic links
   - Current working directory
   - Operating relative to a directory (*openat()* etc.)
   - Scanning directories
7. **Processes**
   - Process IDs
   - Process memory layout
   - Command-line arguments
   - The environment list
   - Process groups and sessions (*)
   - Nonlocal gotos
8. **Process Credentials**
   - Users and groups
   - Process credentials
   - Retrieving process credentials
9. **Signals: Introduction**
   - Signal dispositions
   - Signal handlers
   - Useful signal-related functions
   - Signal sets, the signal mask, and pending signals
10. **Signals: Signal Handlers**
    - Designing signal handlers
    - Async-signal-safe functions
    - Interrupted system calls
11. **Signals: Further Details (*)**
    - Sending signals from a program
    - `SA_SIGINFO` signal handlers
    - Realtime signals
12. **Process Creation and Termination**
    - Creating a new process: *fork()*
    - File descriptors and *fork()*
    - Process termination
    - Monitoring child processes
    - Orphans and zombies
    - The `SIGCHLD` signal
    - Exit handlers (*)
13. **Executing Programs**
    - Executing programs: *execve()*
    - The *exec()* library functions
    - File descriptors and *exec()*
14. **System Call Tracing with *strace***
    - Getting started
    - Tracing child processes
    - Filtering *strace* output
15. **Privileged Programs**
    - Process credentials
    - Set-user-ID and set-group-ID programs
    - Changing process credentials
16. **Threads: Introduction**
    - Pthreads API basics
    - Thread creation and termination
    - Thread IDs
    - Joining and detaching threads
    - Thread attributes
    - Signals and threads
    - Threads and process control
17. **Threads: Synchronization**
    - Shared resources and critical sections
    - Mutexes
    - Locking and unlocking a mutex
    - Condition variables
    - Signaling and waiting on condition variables
    - Dynamically initialized mutexes
    - Dynamically initialized condition variables
    - Other synchronization primitives
18. **IPC: Introduction and Overview**
    - Categorizing IPC
    - Choosing an IPC mechanism
19. **Pipes and FIFOs**
    - Creating and using pipes
    - Connecting filters with pipes
    - FIFOs
20. **Sockets: Concepts and UNIX Domain**
    - Socket types and domains
    - Creating and binding a socket
    - System calls: stream sockets
    - UNIX domain stream sockets
    - System calls: datagram sockets
    - UNIX domain datagram sockets
    - Further details of UNIX domain sockets
21. **Sockets: Internet Domain**
    - Internet domain sockets
    - Data-representation issues
    - Loopback and wildcard addresses
    - Host addresses and port numbers
    - Host and service conversion
    - Internet domain sockets example
    - Additional sockets system calls
22. **Alternative I/O Models**
    - Nonblocking I/O
    - Signal-driven I/O
    - I/O multiplexing: *poll()*
    - Problems with *poll()* and *select()*
    - The *epoll* API
    - *epoll* events
    - *epoll*: edge-triggered notification
    - *epoll*: API quirks
    - Event-loop programming
23. **POSIX IPC Overview**
24. **POSIX Semaphores**
    - Named semaphores
    - Semaphore operations
    - Synchronizing access to a shared resource
    - Unnamed semaphores
25. **POSIX Shared Memory**
    - Creating & opening SHM objects
    - Using SHM objects
    - Synchronizing access to SHM
26. **POSIX Message Queues (*)**
    - Opening, closing, and unlinking a message queue
    - Message queue attributes
    - Sending and receiving messages
    - The *mqueue* filesystem
    - Message queue limits and defaults
    - Message notification via a signal
    - Message notification via a thread
27. **Daemons (*)**
    - Creating a daemon
    - Reinitializing a daemon
28. **Time (*)**
    - Calendar time
    - Elapsed time
    - Process time
29. **Timers and Sleeping (*)**
    - Historic timer APIs
    - POSIX timers
    - POSIX timers: notification
    - POSIX timers: timer overruns
    - Sleeping