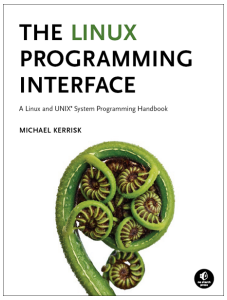
Linux/UNIX System Programming

Course code: M7D-LUSP01

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 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

No previous system programming experience is required.

Related courses

This course is also available as separate smaller pieces:

- *System Programming Fundamentals*, M7D-SPINTRO01
- *Threads and IPC Programming*, M7D-TIPC01

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*
- Numerous example programs written by the course trainer

Course inquiries and bookings

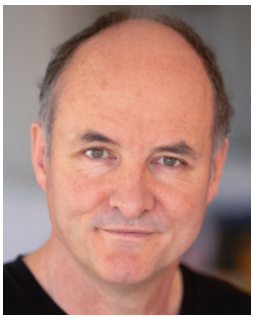
For inquiries about courses and consulting, you can contact us in the following ways:

- Email: training@man7.org
- Phone: +49 (89) 2488 6180 (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/lusp/`.

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.
- He has more than two decades of experience as a teacher and trainer, and first 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 the 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.

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()*, and *close()*
4. **File I/O Buffering**
   - Kernel buffering
   - User-space (*stdio*) buffering
   - Controlling kernel buffering
5. **File I/O: Further Details**
   - The file offset and *lseek()*
   - Atomicity
   - Relationship between file descriptors and open files
   - Duplicating file descriptors
   - File status flags (and *fcntl()*)
   - Other file I/O interfaces (*)
6. **Files**
   - Inodes
   - Retrieving file information: *stat()*
   - File type and mode
   - The process umask
   - Changing file attributes
7. **Directories and Links (*)**
   - Directories and (hard) links
   - Symbolic links
   - Hard links: system calls and library functions
   - Symbolic links: system calls and library functions
   - Current working directory
   - Operating relative to a directory (*openat()* etc.)
   - Scanning directories
8. **Processes**
   - Process IDs
   - Process memory layout
   - Command-line arguments
   - The environment list
   - Process credentials
   - Process groups and sessions (*)
   - Nonlocal gotos
   - The `/proc` filesystem
9. **Signals**
   - Overview of signals
   - Signal dispositions
   - Useful signal-related functions
   - Signal handlers
   - Signal sets, the signal mask, and pending signals
   - Designing signal handlers
10. **Signals: Signal Handlers**
    - Async-signal-safe functions
    - Interrupted system calls
    - `SA_SIGINFO` signal handlers
    - The signal trampoline (*)
11. **Process Creation and Termination**
    - Creating a new process: *fork()*
    - File descriptors and *fork()*
    - Process termination
    - Monitoring child processes
    - Orphans and zombies
    - The `SIGCHLD` signal
    - PID file descriptors
12. **Executing Programs**
    - Executing programs: *execve()*
    - The *exec()* library functions
    - File descriptors and *exec()*
    - Process attributes during *fork()* and *exec()*
13. **System Call Tracing with *strace* (*)**
    - Getting started
    - Tracing child processes
    - Filtering *strace* output
14. **Threads: Introduction**
    - Overview of threads
    - Pthreads API basics
    - Thread creation and termination
    - Thread IDs
    - Joining and detaching threads
    - Thread attributes
    - Signals and threads
    - Threads and process control
15. **Threads: Synchronization**
    - Shared resources and critical sections
    - Mutexes
    - Locking and unlocking a mutex
    - Condition variables
    - Signaling and waiting on condition variables
    - Further details on signaling condition variables
    - Dynamically initialized synchronization primitives
    - Other synchronization primitives
16. **IPC: Introduction and Overview (*)**
    - Categorizing IPC
    - Choosing an IPC mechanism
17. **Pipes and FIFOs**
    - Creating and using pipes
    - FIFOs
    - Connecting filters with pipes
18. **Sockets: Introduction**
    - Socket types and domains
    - Creating and binding a socket (*socket()* and *bind()*)
    - Overview of stream sockets
    - *listen()* and pending connections
    - *accept()* and *connect()*
    - I/O on stream sockets
    - Overview of datagram sockets
    - I/O on datagram sockets
19. **Internet Domain Sockets**
    - Internet domain sockets
    - Data-representation issues
    - Presentation-format addresses
    - Loopback and wildcard addresses
    - Internet domain stream sockets example
20. **Internet Domain Sockets: Address Conversion**
    - Host addresses and port numbers
    - Host and service conversion
    - Internet domain sockets example with *getaddrinfo()*
21. **Sockets: Further Details**
    - Socket shutdown (*shutdown()*)
    - Socket options
    - TCP TIME-WAIT state and `SO_REUSEADDR`
22. **UNIX Domain Sockets**
    - UNIX domain stream sockets
    - UNIX domain datagram sockets
    - Further details of UNIX domain sockets
23. **Alternative I/O Models**
    - Nonblocking I/O
    - Signal-driven I/O
    - I/O multiplexing: *poll()*
    - *poll()* example
    - Event-loop programming
24. **Alternative I/O Models: *epoll***
    - Problems with *poll()* and *select()*
    - The *epoll* API
    - Creating an *epoll* instance: *epoll_create()*
    - Populating the interest list: *epoll_ctl()*
    - *epoll* events
    - Waiting for events: *epoll_wait()*
    - Performance considerations
    - Edge-triggered notification
    - *epoll* API quirks
25. **POSIX Semaphores**
    - Named semaphores
    - Semaphore operations
    - Unnamed semaphores
26. **POSIX Shared Memory**
    - Creating and opening a shared memory object
    - Mapping a shared memory object (*mmap()*)
    - Using shared memory objects
    - Synchronizing access to shared memory