Real Time Group LTD

Real Time & Embedded Linux Solutions

Embedded Linux User-Space \ System Programming

Duration: 90 Hours
Hands-On-Training: 85%

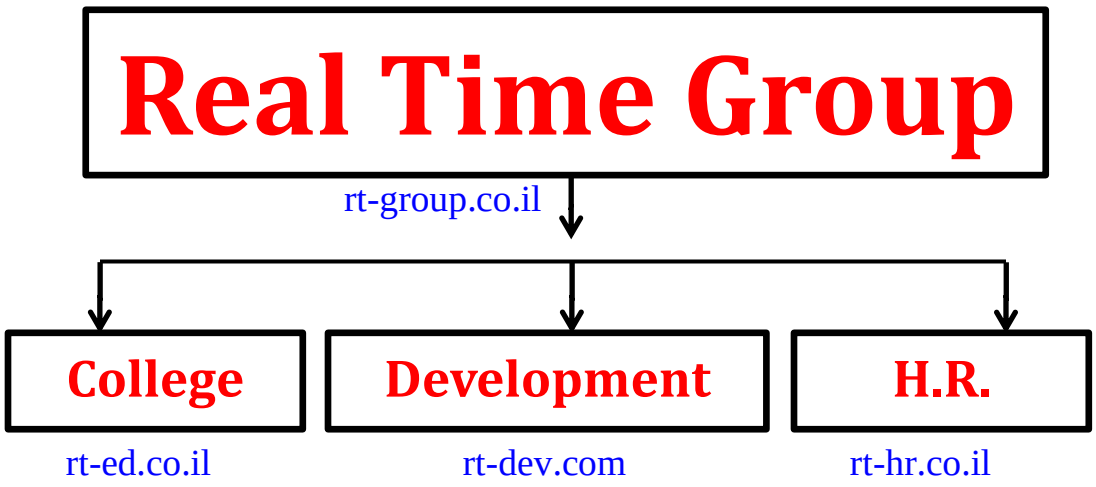
רח' יגאל אלון 123 טל.077-7067057 / 050-3309318 פקס 077-5067058

www.rt-hr.co.il www.rt-ed.co.il www.rt-dev.com

Real Time Group is a multi-disciplinary dynamic and innovative Real-Time O.S. and Embedded Software Solutions Center, established in 2007.

Providing Bare-Metal and Embedded Linux solutions, professional services and consulting, end-to-end flexible system infrastructure, outsourcing, integration and training services for Hardware, Software and RT-OS \ Embedded Systems.

The company is divided into the following three Divisions:



Training Division:

Professional Training Services for Hardware, Software, RT-OS and Embedded systems industries.

We provide the knowledge and experience needed to enable professional engineers to Develop, Integrate and QA Hardware, Software and Networking Projects.

In order to insure experience, all courses are practical – hands-on-training. The latest Development, QA and Automation equipment which are adopted by the industry are used.

All students are supplied with Development-Boards for home-work and course projects.

Course Overview:

The Linux system programming course explains how to implement secure and optimized Linux applications.
It's followed by dozens of class and home hands-on exercises.

The course covers all major areas of programming (development tools, environment processes, Makefiles, building Libraries, using system calls, accessing Hardware, Processes, Threads, IO, networking, synchronization and more).

The course also aims to give an introduction to kernel.

Who should attend:

- C programmers who need to program on Linux operating systems.
- Embedded Linux Developers.
- Linux system programmers \ Engineers.

Prerequisite:

- Knowledge of Linux administration (Linux Fundamentals).
- Knowledge in C programming language.

Linux System programming (Linux User Space)

1. **Introduction to UNIX\Linux Programming**
   a. A Brief History of Linux
   b. Why Linux Programming?
   c. Linux O.S. Architecture

2. **Linux Development tools**
   a. Editing the source files
   b. Compiling with GCC
   c. Coding Defensively

3. **Automating the Process with GNU Make**
   a. What is a Makefile and how does it work
   b. Creating Makefiles

4. **Debugging Linux programs**
   a. Using the gdb debugger
   b. Strace/ltrace
   c. Valgrind
   d. Debuging using the /proc filesystem
   e. Objdump

5. **Writing and Using Libraries**
   a. What are Libraries
   b. Creating and using Static Libraries.
   c. Creating and using Dynamic Libraries.
   d. Making sure the apps find the libraries.

6. **File Descriptors**
   a. What are File Descriptors
   b. The file descriptor table.
   c. Calls That Use File Descriptors
   d. Using System calls (open/read/write/close)
   e. Handling errors.
   f. Using additional calls sucj as: ioct, fctll

7. **User space apps working with Hardware**
   a. How to interact with underlying HW..
   b. Difference between char and block devices in regards to special device files.
   c. Limitations using HW.

8. **Linux Processes**
   a. Linux process tree.
   b. The Idle Process & Init Process.
   c. /sbin/init and startup/shutdown
   d. Process Id's & Viewing Active Processes.
   e. Creating processes.
   f. What gets inherited from parent to child processes?
   g. Waiting for processes to die .
   h. Signals handling - signalfd
   i. Process return value.
   j. Process termination
   k. Zombies and orphans.
   l. Fork performance and vfork.
   m. How is COW used in created processes
   n. epoll vs select
   o. Process groups \ Sessions.

9. **Signals**
    a. What are signals?
    b. How are signals implemented?
        i. Signals as software interrupts
        ii. Hardware interrupts ver software ones
    c. Asynchronous delivery – problems that might occur.
        i. Overcoming the async problem of signals.
    d. Returning back to main context.
    e. Masking signals: sigprocmask
    f. Signal safe functions
    g. Signal interruption.
    h. Various signals and their usage.

10. **Memory associated topics**
    a. What is virtual Memory
    b. What are Page faults
    c. What are Page locks
    d. How does memory protection work?
    e. Memory mapping
    f. Memory Locking
    g. Memory Allocation – what happens behind the curtain
        i. malloc() and its issues..
        ii. mmap()
        iii. memory over commit
        iv. allocating on the stack
        v. Process memory description.

11. **Huge pages in Linux**
    a. what is huge pages in Linux?
    b. How to enable/disable huge pages?
    c. How to determine huge page value

12. **Disk caching & Swapping**
    a. Disk Caching memory:
    b. Swapping Memory
    c. Difference between them

13. **Linux Threads (Pthreads)**
    a. The pthread library.
    b. Creating new threads.
    c. Waiting for threads .
    d. Detaching threads.
    e. pthreads and signals
    f. cleanup handlers.
    g. cancellation points
    h. pthread affinities.
    i. pthreads and signals

14. **Synchronization and Critical Sections**
    a. Using Mutexes
    b. Mutexes ver Semaphores
    c. How are futexes implemented?
    d. Using futexes for process synchronization.
    e. Barriers in user mode
    f. Condition variables
    g. Readers/Writer locks.
    h. Spin locks.
    i. Atomic operations (and their price)

15. **Processes Vs.Threads**
    a. What are the differences between them
    b. Guidelines for When should we use each one

16. **Inter-process Communication (IPC)**
    a. System V Shared Memory
    b. System V Semaphores
    c. Pipes
    d. Sockets

17. **The IO subsystem**
    a. What is the IO Subsystem?
    b. The kernel page and buffer caches.
    c. ionice(2) and it's uses to control IO priority.
    d. Overcoming the copy problem with O_DIRECT.
    e. Overcoming the copy problem with mmap.

18. **Zero copy**
    a. Why do we need zero coping?
    b. mmap()
    c. dpdk

19. **Networking**
    a. Local and domain sockets.
    b. TCP/IP stack overview
    c. The socket API's
    d. Datagram v.s. connection-oriented sockets
        i. TCP- SOCK_STREAM sockts.
        ii. UDP- SOCK_ DGRAM sockts
    e. client/server Local \ UDP \TCP examples
    f. Using raw sockets.

g. Dpdk - Data Plane Development Kit

h. Debugging tools for networking apps.

20. **Introduction to Linux Device Drivers**

a. Working with Hardware Devices

b. Major minor numbers

c. Char and Block devices

<u>הערות :</u>

- ✓ פתיחת המסלול מותנה במספר נרשמים.
- ✓ החברה מביאה לידיעתם של המשתתפים שיתכנו שינוים בתוכן הקורסים ובמועדם.
- ✓ החברה מתחייבת להודיע המשתתפים על כל שינוי.
- ✓ החברה שומרת לעצמה את הזכות לשנות את תכני המסלול בהתאם לשיקול דעתה הבלעדית.