

# Practical Real-Time Programming in User-Space on Linux

foss.in 2007

Lennart Poettering  
lennart@poettering.net



December 2007

# Who Am I?

Software Engineer at Red Hat, Inc.

Developer of PulseAudio, Avahi and a few other Free Software projects

`http://0pointer.de/lennart/`

`lennart@poettering.net`

IRC: mezcadero

What is real-time (RT) programming?

What is real-time (RT) programming?

Whenever an RT process is able to run, it runs.

Where is this used?

Where is this used?

ABS, Medical Systems, Finance, Audio, ...

Our focus: Audio on Linux

## Hard vs. Soft Real-time

## Linux as an RT kernel

# Linux as an RT kernel

## Soft RT

Scheduling latency: HZ

## Making a process real-time

## Real-time priorities

Avoid locks!

Avoid locks!  
Priority inversion

Avoid locks!

Priority inversion

Priority inheritance, PTHREAD\_PRIO\_INHERIT

## Lock-free programming

Lock-free programming

Wait-free programming

Lock-free programming

Wait-free programming

Lock-free algorithms are difficult, wait-free even more so

## Multi-Reader, Multi-Writer lock-free queues

# Multi-Reader, Multi-Writer lock-free queues

## Lock-free reference counting

# Atomic operations

## Atomic operations

APIs and portability: libatomic\_ops vs. glibc vs. glib vs. \_\_sync

## Atomic operations

APIs and portability: libatomic\_ops vs. glibc vs. glib vs. \_\_sync

Emulating atomic ops: In kernel easy, in user-space hard

# Memory Barriers

Lock-free memory allocation: difficult

Lock-free memory allocation: difficult  
Beware of thread local pools like GSlice!

Lock-free memory allocation: difficult  
Beware of thread local pools like GSlice!  
Alternative: free() lists

## Reference-counted memory handling

# Reference-counted memory handling

## Zero-Copy

Reference-counted memory handling

Zero-Copy

Minimize copies, cache pressure

# Memory locking

## Memory locking

To `mlockall()` or not `mlockall()`?

## Memory locking

To `mlockall()` or not `mlockall()`?

`madvise()` instead? Or temporary `mlock()`?

## Timers and sleeping: high-resolution timers

Timers and sleeping: high-resolution timers  
hrtimers on x86 only, for now

Timers and sleeping: high-resolution timers  
hrtimers on x86 only, for now  
nanosleep(), itimers, POSIX Timers are fine

Timers and sleeping: high-resolution timers  
hrtimers on x86 only, for now  
nanosleep(), itimers, POSIX Timers are fine  
But what about poll()?

Timers and sleeping: high-resolution timers

hrtimers on x86 only, for now

nanosleep(), itimers, POSIX Timers are fine

But what about poll()?

Combine itimers + ppoll() with POSIX real-time signals

Timers and sleeping: high-resolution timers

hrtimers on x86 only, for now

nanosleep(), itimers, POSIX Timers are fine

But what about poll()?

Combine itimers + ppoll() with POSIX real-time signals

Signals are evil!

Timers and sleeping: high-resolution timers

hrtimers on x86 only, for now

nanosleep(), itimers, POSIX Timers are fine

But what about poll()?

Combine itimers + ppoll() with POSIX real-time signals

Signals are evil!

Beware of old kernels with ppoll()!

# Mutexes, Semaphores, Conditions

# Mutexes, Semaphores, Conditions Futexes

# Mutexes, Semaphores, Conditions Futexes

Semaphores good

Mutexes, Semaphores, Conditions

Futexes

Semaphores good, Conditions bad

# Mutexes, Semaphores, Conditions Futexes

Semaphores good, Conditions bad, Mutexes with PI good

Using poll() and futexes?

Using poll() and futexes?  
FIFOs and eventfd()

Using poll() and futexes?  
FIFOs and eventfd() ... use locking

Using poll() and futexes?

FIFOs and eventfd() ... use locking

Compromise: Wrap eventfd()/FIFOs in atomic ops

Better solution: kevent

Better solution: kevent (theoretically)

Better solution: kevent (theoretically)

Allows sleeping on timers, futexes, fds

Better solution: kevent (theoretically)

Allows sleeping on timers, futexes, fds and is lock-free

## Lazy binding and relocations

Lazy binding and relocations  
RTLD\_NOW and \$BIND\_NOW

## Robust RT and security

## Robust RT and security

Watchdog thread vs. RLIMIT\_CPU and SIGXCPU

Robust RT and security  
Watchdog thread vs. RLIMIT\_CPU and SIGXCPU  
RLIMIT\_RTPRIO

Robust RT and security

Watchdog thread vs. RLIMIT\_CPU and SIGXCPU

RLIMIT\_RTPRIO RLIMIT\_RTTIME

How to partition your code best between RT threads and non RT threads?

How to partition your code best between RT threads and non RT threads?

It's difficult!

Avoid in RT:

- Disk I/O

Avoid in RT:

- Disk I/O
- Any other kind of blocking I/O

Avoid in RT:

- Disk I/O
- Any other kind of blocking I/O
- Unbounded algorithms, i.e. slower than  $O(n)$

Avoid in RT:

- Disk I/O
- Any other kind of blocking I/O
- Unbounded algorithms, i.e. slower than  $O(n)$
- Code that needs locking – unless there is only one piece of code that locks it

Splitting code up into non-RT and RT threads comes at a cost:

Splitting code up into non-RT and RT threads comes at a cost:  
Context switches

Splitting code up into non-RT and RT threads comes at a cost:  
Context switches; Latency due to buffering

Splitting code up into non-RT and RT threads comes at a cost:  
Context switches; Latency due to buffering; Code becomes a lot  
more difficult to understand.

Thus: in some cases it even makes sense to do non-trivial calculations in the RT thread. As long as it is bounded by  $1/\text{HZ}$  in completion time. Additional benefit: *your* code can decide when it is best to execute the non-trivial calculations.

Thus: in some cases it even makes sense to do non-trivial calculations in the RT thread. As long as it is bounded by  $1/\text{HZ}$  in completion time. Additional benefit: *your* code can decide when it is best to execute the non-trivial calculations.

But this is controversial.

# Debugging RT

## Debugging RT

Make sure to run a watchdog of some kind.

## Debugging RT

Make sure to run a watchdog of some kind.

Possibly a shell with a high real-time priority

Profile your code!

Profile your code!  
Logging in RT programs

Profile your code!  
Logging in RT programs  
Tracing memory allocations

Profile your code!  
Logging in RT programs  
Tracing memory allocations  
Tracing mutexes

Profile your code!

Logging in RT programs

Tracing memory allocations

Tracing mutexes

Idea: A module for valgrind that looks for stuff that should not be done in RT threads

That's all, folks.

That's all, folks.  
Any questions?