



From Fast to Predictably Fast

Dominic Duval
Consultant, Red Hat
July 17, 2009

History

- Realtime used to require considerable modifications to the source code
 - RTLinux
 - RTAI
- Success limited to specific industries
- Largely ignored in others

History (cont'd)

- CONFIG_PREEMPT_RT then introduced
 - Works with standard POSIX interfaces
 - Wrongly interpreted by some as the solution to fix *any* application
 - With improved latencies
 - With no source level modification
 - With no system tuning
 - On any hardware

Agenda

- Scheduling
- Safety
- Memory
- Locking
- Good and bad practices

Disclaimers

- Not a discussion of CONFIG_PREEMPT_RT implementation
- Not comprehensive of all techniques available
- Results depend on hardware
- Acceptable latency in one environment may be unacceptable in another
- Most recommendations also apply to non-RT environments

Before we start

- Appropriate hardware platform
 - HP Proliant with right BIOS version
 - IBM BladeCenter LS21 and HS21 with `ibm-prtm` service
 - Others: test with `smi_detector` or `hwlat_detector`
 - SMIs are the enemy
- Kernel compiled with `CONFIG_PREEMPT_RT=y`
- Accurate, reliable latency measurement mechanism
- Application (with sources!)

Testing the System

- Memory locking
 - Make sure the right capabilities are available
- Possible to set scheduler policies and priorities
- CONFIG_PREEMPT_RT
- Access to high resolution timers
- Clock resolution high enough

`/sys/devices/system/clocksource/clocksource0/current_clocksource`

- `rtcheck`

Scheduler Policies

- Realtime systems enforce process scheduling policies
- Three main policies used:
 - SCHED_OTHER
 - SCHED_FIFO
 - SCHED_RR
- Other newer policies are relatively uncommon:
 - SCHED_BATCH introduced in 2.6.8
 - SCHED_IDLE added in 2.6.22
- Set with `sched_setscheduler()`

Realtime Policies

- SCHED_FIFO or SCHED_RR
- SCHED_FIFO preferred over SCHED_RR
 - No timeslices, predictable
- Preempt regular threads
- Do not expire and yield to lower priority processes
- Can starve lower priority threads
 - Realtime or not
- Give access to realtime priorities

Setting priorities right

- Realtime priorities range from 1 (lowest) to 99 (highest)
- Can be set through `sched_setscheduler()`
- `rtctl` does it persistently with `/etc/rtgroups`:

[group name]:[scheduler policy]:[scheduler priority]:[regexp]

Typical Realtime Priorities

99	Watchdog and migration threads
90-98	High priority kernel threads
81-89	IRQ Threads
80	NFS
70-79	Soft IRQs
2-69	Applications
1	Low priority kernel threads

How to Assign Application Priorities

- Solution may be obvious
 - i.e. one time-critical task only
- Assigning priorities based on timing restrictions:
Rate-monotonic analysis
 - Highest priority goes to the task with tightest requirements (highest frequency)
 - Only valid for tasks with fixed frequencies
- Examining the application's behavior
 - Error prone but often only choice

Interrupt and Process Binding

- 1) Switch irqbalance off
- 2) Determine how many CPU cores the app requires
- 3) Isolate the corresponding number of cores from other processes
 - `tuna --isolate`
- 4) Bind your application threads to those free cores
 - `taskset`
- 5) Bind interrupts to other cores
 - `/proc/irq/* /smp_affinity`

Dont Trust RT Applications

- Things may go wrong: runaway processes
- Keep a shell running with SCHED_FIFO and priority 99
- rt-watchdog
 - User space watchdog running with a high realtime priority
 - Relies on a SCHED_OTHER canary process
 - Prevents lockups by killing runaway realtime processes

Keeping RT Apps in Check

- CONFIG_RT_GROUP_SCHED in 2.6.25
 - Reserve CPU bandwidth for control groups
 - Works for non-root users
 - Not what cgroups were designed for
- 2.6.25 comes with RLIMIT_RTTIME
 - Time slices for realtime processes
 - Malicious code may still starve other processes
 - Realtime Busy loop + fork bomb
- RLIMIT_RTPRIO in 2.6.12
 - Defines priority ceiling for processes

RealtimeKit

- Daemon that hands out SCHED_RR scheduling to threads that ask for it.
- Depends on SCHED_RESET_ON_FORK
 - In Ingo's tip tree
 - `sched_setscheduler(pid, SCHED_RR | SCHED_RESET_ON_FORK, ¶m);`
- Extra dependency: DBUS
- Assumes SCHED_RR

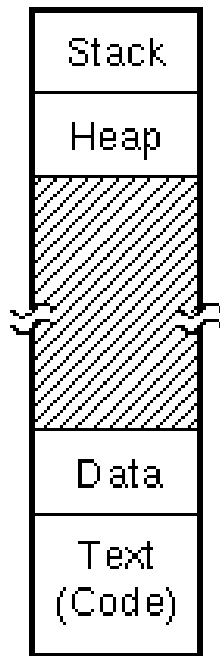
Page Faults

- Main sources of page faults:
 - Pages swapped in
 - Delayed memory allocation
 - Mapped files
- Major page faults
 - Deal with disk access
 - Large source of latency
- Minor page faults
 - Physical memory allocation
 - Relatively negligible

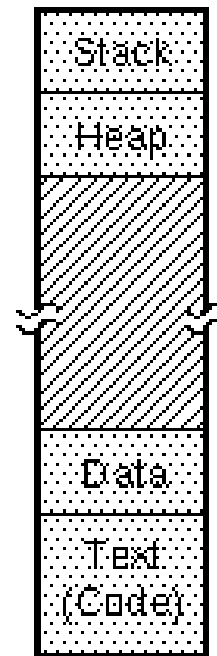
Avoiding Page Faults

- mlockall
 - Easy, heavy-handed method
- mlock
 - More flexible, more complex
- Tips
 - Use `MCL_CURRENT` | `MCL_FUTURE`
 - Memory locking is not perfect
 - Consider possible memory pressure issues
 - Set `CAP_IPC_LOCK` if process is unprivileged
 - `madvise()` with `MADV_WILLNEED`

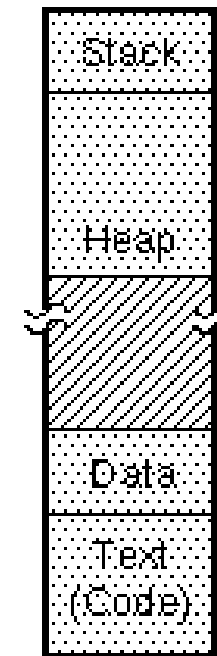
Avoiding Page Faults (cont'd)






Before mlockall
Call



After mlockall
Call



After malloc
Call

-  = Pageable
-  = Locked in physical memory (not pageable)
-  = Unmapped address space

Page Faults and Stacks

- Problem: stack sizes can be unpredictable
- A new page allocated for the stack will cause a minor page fault
- Solution: maximize and pre-fault the stack before locking memory
- How to determine how large?
 - Pre-fault a large stack size to a known value
 - Run the application extensively
 - Then determine how many pages were accessed
- Not possible for all applications

Predictable Memory Allocation

- General rule: don't allocate/free memory in realtime contexts
- Reserve memory at startup time
- Lock it
- Avoid calls that allocate memory in the background
 - `fopen()`
 - `readdir()`
 - etc.

malloc() Enhancements

- Calls to malloc() not always translated to brk()
 - Reusing memory optimizes allocation speed
 - Preallocated sub-heaps reduces fragmentation
 - Hard to predict
- brk() called to reclaim memory
 - Disable with:

```
mallopt (M_TRIM_THRESHOLD, -1)
```
 - Result: memory is never returned to the system

malloc() Enhancements (cont'd)

- `malloc()` relies on `mmap()` for allocations larger than 128k
 - Results in calls to `munmap()` to give back memory
 - Disable with `mallopt(M_MMAP_MAX, 0)`
 - Result: memory not allocated with `mmap()`

malloc() Enhancements (cont'd)

Setup

```
1) mlock/mlockall  
2) mallopt(M_TRIM_THRESHOLD, -1);  
3) mallopt(M_MMAP_MAX, 0);  
4) malloc(MAX_MEM_FOOTPRINT)  
5) Touch allocated memory  
6) free()
```

RT

```
malloc()/free(), new/delete
```

Dynamic Library Preloading

- “Lazy linking” used by default for dynamic libraries
 - Pages loaded only when invoked
 - Reduces physical memory usage
 - Introduces delays on initial library calls
- Alternative: preload all libraries in memory

```
export LD_BIND_NOW=1
```
- Also available at linking time:

```
gcc app.c -o app -Wl,-z,now
```

Monitoring page faults

- Keep an eye on page faults
 - Other scenarios may lead to more memory allocated
 - Stack may be larger than expected
- `getrusage(RUSAGE_SELF, &usage);`
 - Includes current process and threads
 - Minor faults: `usage.ru_minflt`
 - Major faults: `usage.ru_majflt`
- Focus on major faults after application fully initialized

Deterministic Locking

- General rule: rely on the pthread library for locking
- pthread mutexes support PTHREAD_PRIO_INHERIT

```
pthread_mutexattr_init(&rt_safe);
```

```
pthread_mutexattr_getprotocol(&rt_safe,  
    &rt_protocol);
```

```
pthread_mutexattr_setprotocol(&rt_safe,  
    PTHREAD_PRIO_INHERIT);
```

```
pthread_mutex_init(&msgSem, &rt_safe);
```

- Alternative to priority inheritance: priority ceilings

Things to Worry About

- Creating threads and processes
- `sched_yield()`
 - Assumes other processes ready to run at the same priority.
 - Doesn't work well with CFS
 - Replace with `pthread_barrier_*`
- Filesystem-related operations
- Signals
- `ioctl()`

Things to Look At

- Reduce network-level latency with `setsockopt ()` and:
 - `TCP_NODELAY`
 - `TCP_CORK`
- VDSO optimization
 - `echo 2 > /proc/sys/kernel/vsyscall64`
- Disable services you don't need
 - `pcscd`, graphical desktops, mouse services, etc.

Questions?

Dominic Duval
dduval@redhat.com

Slides and whitepaper posted on
<http://people.redhat.com/dduval>