Monitoring Linux Guests and Processes with Linux Tools

Christian Borntraeger (cborntra@de.ibm.com)
Linux on System z Development
IBM Lab Boeblingen, Germany
Session 9266, Wed Feb 14

Agenda

- Linux Time Infrastructure
- Accessing the z/VM Monitor Stream
- Accessing LPAR data
- Outlook

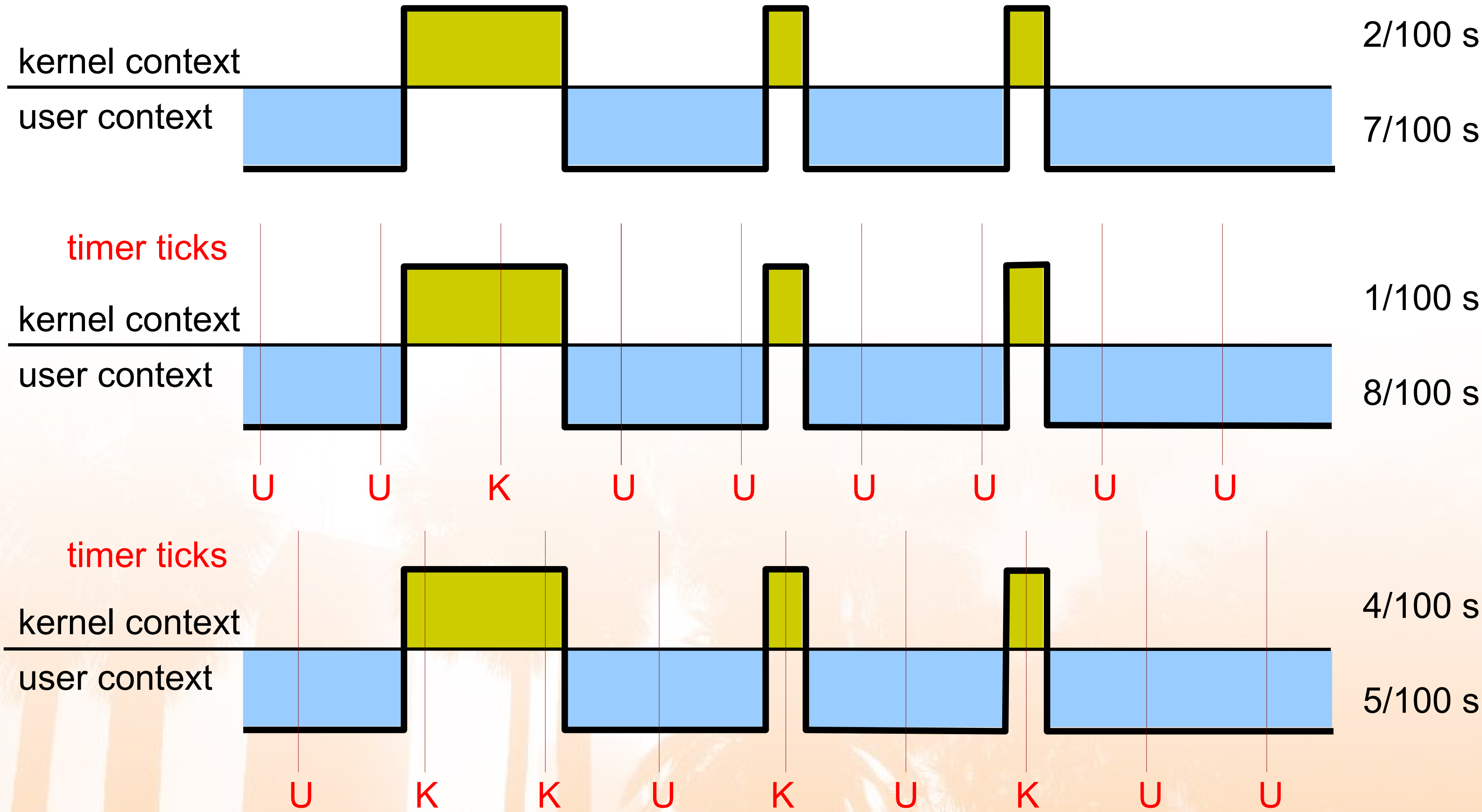
Linux Time Infrastructure

- Linux Time Infrastructure
  - Ressources
  - Tick based time-keeping
  - CPU timer bases time-keeping
  - user interfaces
- Accessing the z/VM Monitor Stream
- Accessing LPAR data
- Outlook

Linux Time Infrastructure

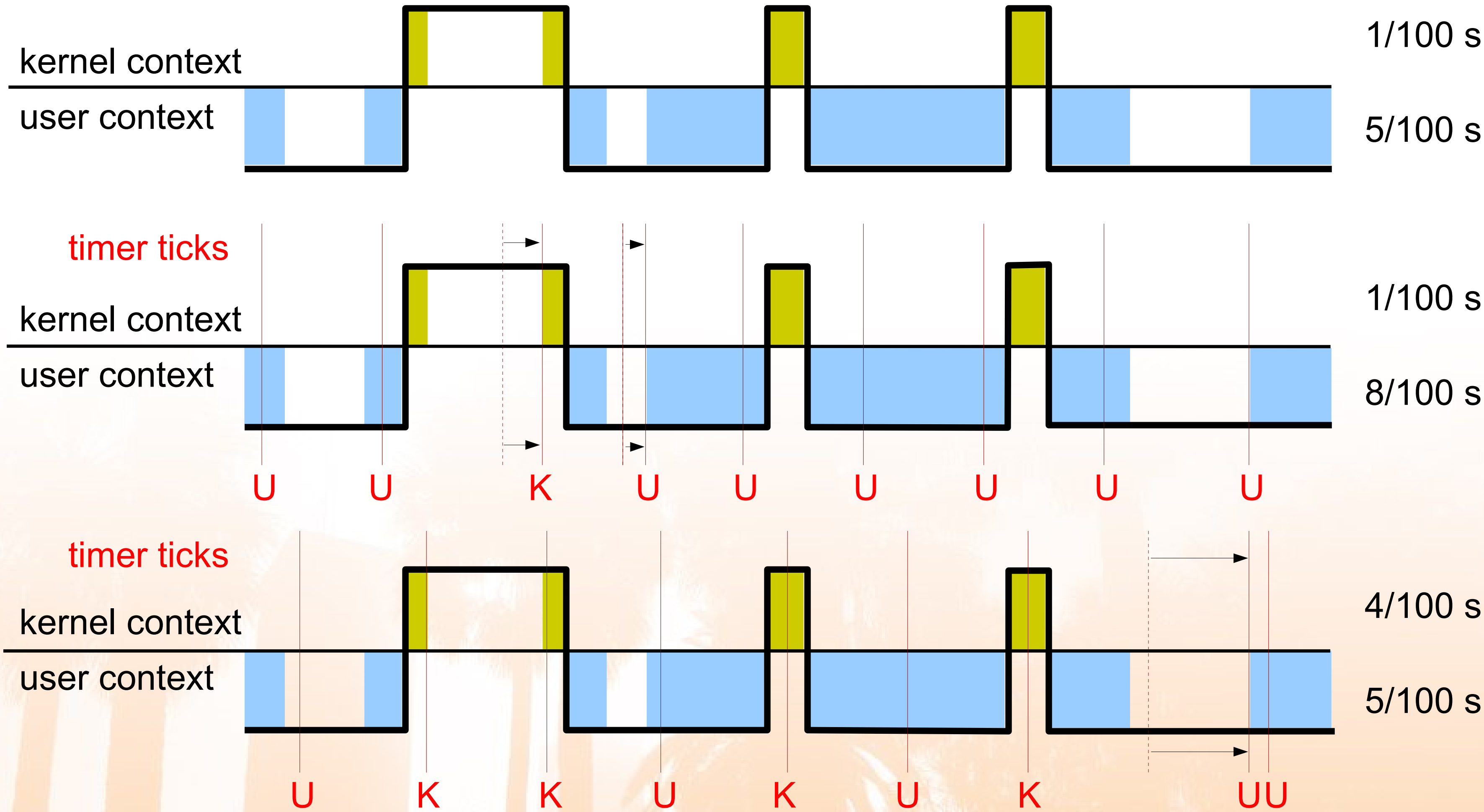
- What ressources can be monitored with Linux tools
  - cpu
  - I/O
  - memory
- Time is an important aspect for measurements
- How is Linux measuring time?
  - timer ticks are used for internal tracking

Tick based (mis-) accounting



Simplified representation: only contains user and kernel context

Tick based cpu accounting & virtual cpus



Simplified representation: only contains user and kernel context

Tick based accounting is wrong

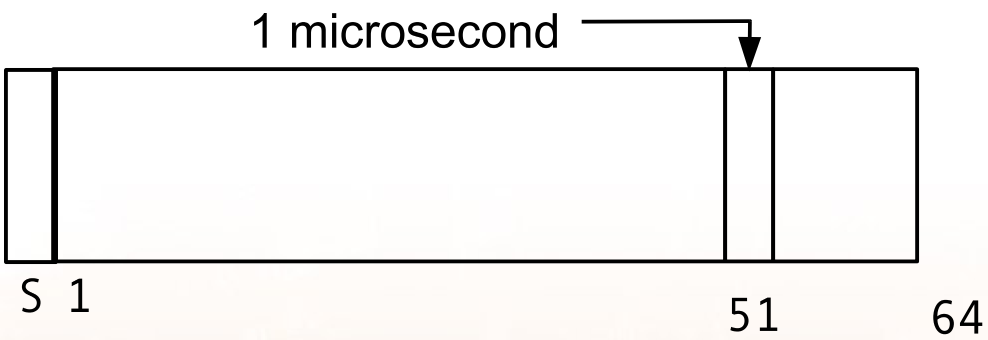
- Tick accounting by design has some inaccuracy
  - On non virtualized system the approach usually is good enough
- Systems with virtual cpus (z/VM, VMware, Xen, etc):
  - The real cpu usually spents part of its time “elsewhere”
  - Process timeslices are based on real time, usually 5-6 ticks.
  - Processes get accounted time they did not use
  - Processes can loose their entire timeslice
- No distinction between real time and virtual cpu time
- No concept of involuntary wait or steal time

How to fix the accounting numbers ?

- Do not use the Linux accounting numbers
  - Use per image accounting numbers generated by the hypervisor
  - Limited scope, only usable to get per image data
- Normalize cpu accounting numbers
  - Read average cpu usage numbers from the hypervisor
  - Multiply Linux cpu accounting numbers with average cpu usage
  - Hard to do right for process accounting numbers
- Do it properly and use a precise accounting mechanism

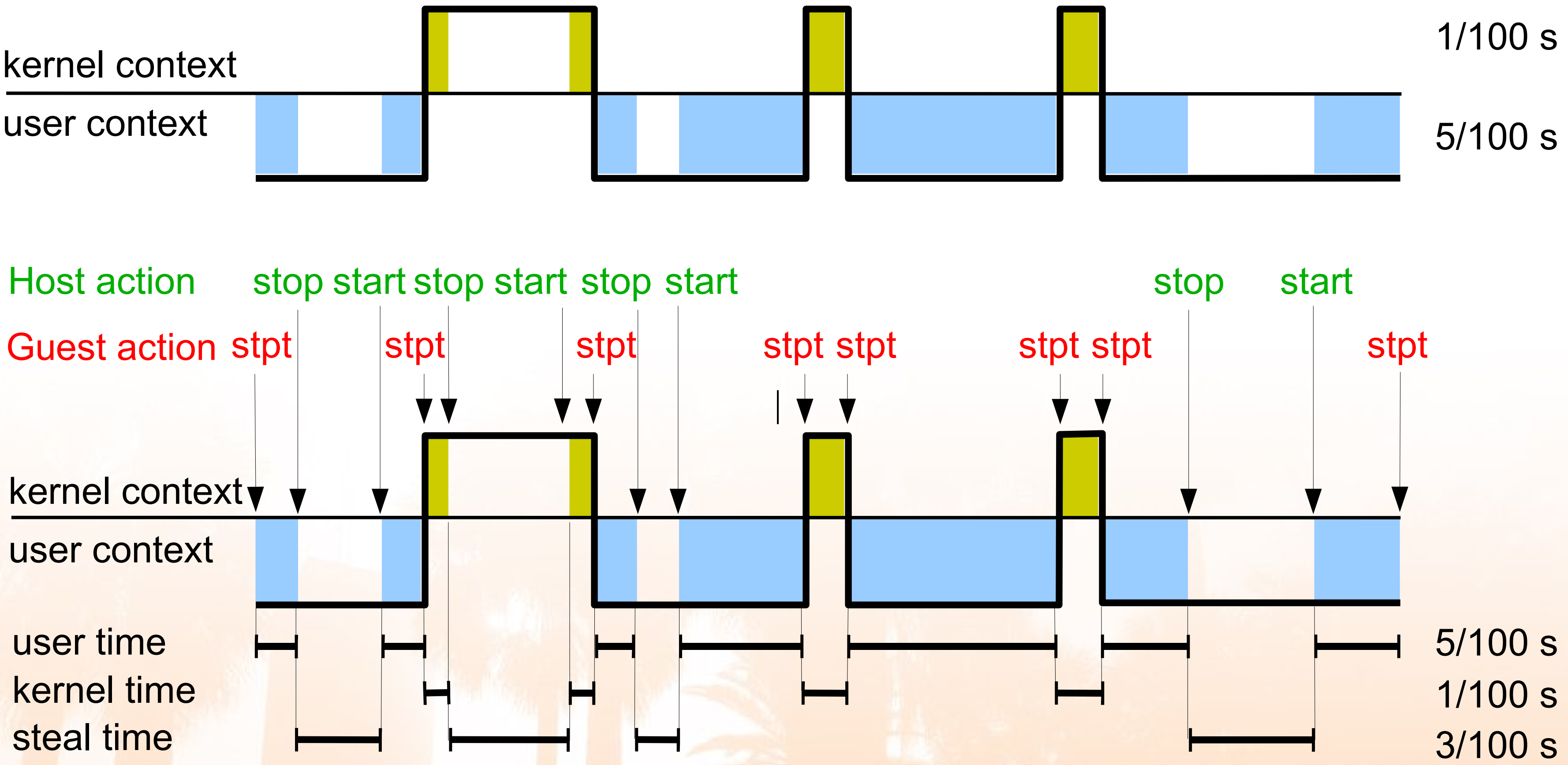
The zSeries cpu timer

- Principles of Operation page chapter 4
- Each cpu has a 64 bit cpu timer register



- Same format as bits 0-63 of the TOD clock except for bit 0 (sign)
- Stepping rate of cpu timer and the TOD clock are synchronized
- and are stepped at the same rate
- ... but only while the virtual cpu is backed by a physical cpu !

Timer based cpu accounting



Simplified representation: only contains user and kernel context

Accounting interfaces revisited (1)

- Times reported by all the Linux accounting interfaces changed
  - Old: percent of time spent in a context by a virtual cpu
  - New: percent of time spent in a context by a real cpu
  - New: additional field in /proc/stat output - steal time
- Precision of the cpu time accounting numbers increased
  - Internal precision is at least 1 microsecond
  - Update is done on each context switch
  - Numbers are converted to ticks (1/100 second) on delivery to user space
- All Linux user space tools suddenly display correct information
  - Except for the “missing” time for cpu steal time, old top adds steal to idle
- Cpu time normalization with average cpu calculation breaks
  - Need to distinguish between “good” and “bad” Linux systems
  - in regard to cpu time accounting numbers

Accounting interfaces revisited (2)

- Overall system information: /proc/stat
  - red – changed semantics, blue – new number

```
# cat /proc/stat
   cpu  212314 0 31246 74377 4152 79 535 1900
   cpu0 107657 0 15727 35701 1967 38 267 955
   cpu1 104656 0 15518 38675 2185 40 267 944
   intr 317360 280140 37220
   ctxt 346461
   btime 1141129302
   processes 69331
   procs_running 1
   procs_blocked 0

   cpu lines: <user> <nice> <system> <idle> <iowait> <irq> <softirq> <steal>
              the unit of these numbers is a tick, 1/100s for zSeries
   intr line: <total number of interrupts> <ext.interrupts> <i/o interrupts>
   ctxt line: number of context switches
   btime line: boot time in seconds since the Unix epoch
   processes line: number of processes created
   procs_running line: number of processes currently running
   procs_blocked line: number of processes currently blocked
```

Accounting interfaces revisited (3)

- red – changed semantics, blue – new number

```
top - 09:50:20 up 11 min,  3 users,  load average: 8.94, 7.17, 3.82
Tasks:  78 total,   8 running,  70 sleeping,   0 stopped,   0 zombie
 Cpu0 : 38.7%us,  4.2%sy,  0.0%ni,  0.0%id,  2.4%wa,  1.8%hi,  0.0%si, 53.0%st
 Cpu1 : 38.5%us,  0.6%sy,  0.0%ni,  5.1%id,  1.3%wa,  1.9%hi,  0.0%si, 52.6%st
 Cpu2 : 54.0%us,  0.6%sy,  0.0%ni,  0.6%id,  4.9%wa,  1.2%hi,  0.0%si, 38.7%st
 Cpu3 : 49.1%us,  0.6%sy,  0.0%ni,  1.2%id,  0.0%wa,  0.0%hi,  0.0%si, 49.1%st
 Cpu4 : 35.9%us,  1.2%sy,  0.0%ni, 15.0%id,  0.6%wa,  1.8%hi,  0.0%si, 45.5%st
 Cpu5 : 43.0%us,  2.1%sy,  0.7%ni,  0.0%id,  4.2%wa,  1.4%hi,  0.0%si, 48.6%st
Mem:    251832k total,   155448k used,    96384k free,     1212k buffers
Swap:   524248k total,    17716k used,   506532k free,    18096k cached

  PID USER       PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
20629 root       25   0 30572  27m 7076 R 55.2 11.1   0:02.14 cc1
20617 root       25   0 40600  37m 7076 R 47.0 15.1   0:03.04 cc1
20635 root       24   0 26356  20m 7076 R 42.3  8.4   0:00.75 cc1
20638 root       25   0 23196  17m 7076 R 27.0  7.2   0:00.46 cc1
20642 root       25   0 15028 9824 7076 R 18.2  3.9   0:00.31 cc1
20644 root       20   0 14852 9648 7076 R 17.0  3.8   0:00.29 cc1
   26 root        5 -10     0    0    0 S  0.6  0.0   0:00.03 kblockd/5
  915 root       16   0  3012  884 2788 R  0.6  0.4   0:02.33 top
    1 root       16   0  2020  284 1844 S  0.0  0.1   0:00.06 init
```

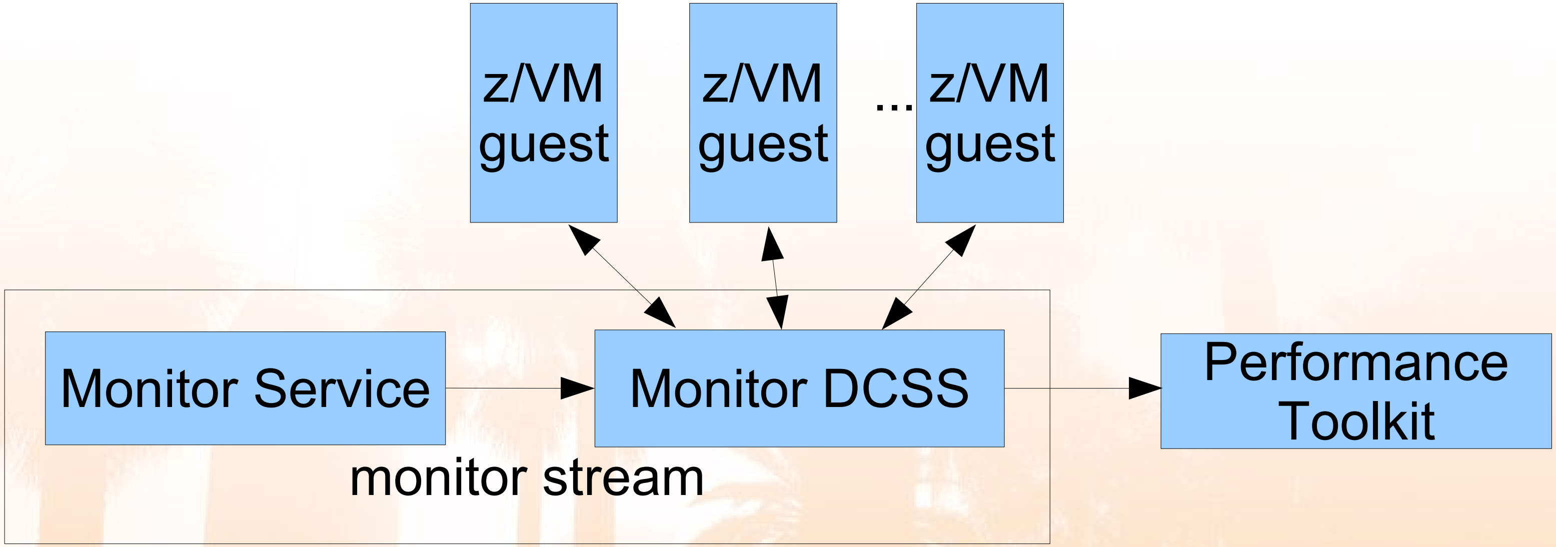
Accessing the z/VM Monitor Stream

Linux Time Infrastructure ✔

- Accessing the z/VM Monitor Stream ✖
  - z/VM monitor infrastructure
  - Linux appldata driver
  - Monitor record reader (monreader)
  - Monitor stream application support (monwriter)
- Accessing LPAR data
- Outlook

z/VM Monitor Service Infrastructure

- Monitor data within the monitor stream
  - data is in a Shared Memory Segment (DCSS)
  - monitor service collects and writes data
  - Performance Toolkit reads data
  - z/VM guest can read and write data



z/VM Monitor Service Infrastructure

- There are different record domains
  - System, Storage, User, Appldata, ...
- There are different record types
  - Event oder Sample Records
- Control via MONITOR CP command
  - setting sampling interval, record domains, types.....
- Performance Toolkit for data evaluation
  - accessible via 3270 or http

Linux appldata driver

- Kernel modules that collects kernel data und puts it into the monitor stream
  - appldata_os
    - cpus and cpu times
    - thread numbers
  - appldata_mem
    - memory
    - paging
    - cache
  - appldata_net_sum
    - packets
    - bytes
    - errors
- Activation via /proc/sys/appldata

```
# modprobe appldata_os
# echo 20000 > /proc/sys/appldata/interval
# echo 1 > /proc/sys/appldata/timer
# echo 1 > /proc/sys/appldata/os
```

Linux appldata driver

- Performance toolkit understands the data format
- Uses virtual CPU time
  - sample interval in virtual cpu time (milliseconds)
  - on idle systems lower sample rate
  - virtual timer is per cpu, accumulated time is used
  - independent from z/VM sampling interval
- Steal time was added in newer versions
  - Kernel 2.6.18, RHEL5
  - current Perfkit ignores the steal time
- Option in user directory is necessary
  - OPTION APPLMON

Monitor record reader

- Device driver monreader for reading monitor data into Linux
  - 2.6.10, SLES9 SP2, RHEL5 (?)
- Device node /dev/monreader
  - char device, read-only
- Applications can read monitor stream in raw format
  - Driver does not transform/format data like Performance Tool Kit
  - similar to monwrite CMS command
- Records are stored in a ring buffer – data may be wrong
  - Use acknowlegment before processing (zero byte read)

Monitor record reader

- Special user directory entries are necessary
  - IUCV *MONITOR
  - NAMESAVE <Monitor DCSS> (z.B. NAMESAVE MONDCSS)
- For loading a DCSS you have to modify the memory settings of Linux
  - DCSS must not overlap with guest memory
  - „mem=“ kernel parameter is necessary
  - alternative: memory hole using „DEFINE STORAGE CONFIG“

```
CP DEF STOR CONFIG 0.144M 180M.512M
STORAGE = 652M
Storage Configuration:
0.144M 180M.512M
Extent Specification          Address Range
------------------------------- ---------------------------------
            0.140M         0000000000000000 - 0000000008BFFFFF
          180M.512M        000000000B400000 - 000000002B3FFFFF
Storage cleared - system reset.
```

Monitor record reader: data layout

- Reading from the device provides a 12-byte monitor control element (MCE), followed by a set of one or more contiguous monitor records (similar to the output of the CMS utility MONWRITE without the 4K control blocks).
  - See “Appendix A: *MONITOR” in z/VM Performance for a layout of a monitor control element (MCE)
- Layout when reading from device driver

```
...
<0 byte read>
<first MCE>                \
<first set of records>    |...
...                            |- data set
<last MCE>                |
<last set of records>   /
<0 byte read>
...
```

Monitor stream application support

- Device driver monwriter
- API for writing APPLDATA monitor records
  - since Linux 2.6.19
- Device node /dev/monwriter
  - allows applications to write appldata monitor records
- User space daemons can write data into the monitor stream
  - e.g. process data like top or file system data like df

Monitor stream application support

- User directory change is necessary
  - OPTION APPLMON
- The monitor must be activated
  - MONITOR SAMPLE ENABLE APPLDATA ALL
  - MONITOR EVENT ENABLE APPLDATA ALL
- Application can open, write and close /dev/monwriter
  - control data structure is defined in monwriter.h (Linux includes)
  - see “Device Driver and Installation” for details

Accessing LPAR data

Linux Time Infrastructure ✔

- Accessing the z/VM Monitor Stream ✔
- Accessing LPAR data ✘
    - hypfs
    - sysinfo
- Outlook

Accessing LPAR data

- Scenario: Linux in an LPAR
  - PR/SM instead of CP as hypervisor
  - scheduling of virtual CPUs on physical CPUs
- How to monitor ressource usage in LPAR?

Hypfs - introduction

- A new filesystem represents the LPAR data
  - uses diagnose 0x204 and 0x224
  - does not work under z/VM (coming soon)
- Filesystems must be mounted
  - console: `mount none -t s390_hypfs /sys/hypervisor/s390/`
  - fstab: `none /sys/hypervisor/s390 s390_hypfs defaults 0 0`

Hypfs - directory structure

```
/sys/hypervisor/s390
|-- update
|-- cpus
|    |-- <cpu-id>
|    |    |-- mgmtime
|    |    `-- type
             [...]
|    `-- <cpu-id>
|         |-- mgmtime
|         `-- type
|-- hyp
|    `-- type
`-- systems
     |-- <lpar-name>
     |    `-- cpus
     |         `-- <cpu-id>
     |         |    |-- cputime
     |         |    |-- mgmtime
     |         |    |-- onlinetime
     |         |    `-- type
     |         `-- <cpu-id>
             [...]
     |
     `-- <lpar-name>
          `-- cpus
             [...]
```

- update: Write only file to trigger the update
- cpus/: Directory for all physical cpus
- cpu-id/: directory for one physical CPU
  - type: e.g. CP, IFL
  - mgmtime: LPAR overhead in microseconds
- hyp/: Directory for hypervisor information
  - type: currently only „LPAR Hypervisor“
- systems/: Directory for all LPARs

Hypfs - usage

- At init time of hypfs
  - the available subcodes are probed
  - the initial values are populated
- Data is only updated if users write into the update file
  - the filesystem rebuilds all files – already open
- When an update of hypfs is triggered, DIAG 204 is issued to gather the new Hypervisor data.
- If an application wants to ensure to get consistent data, the following should be done:
  - 1.Read modification time via stat(2) from the update attribute
  - 2.If data is too old, write to update attribute and goto 1
  - 3.Read data from filesystem
  - 4.Read modification time of the update attribute again and compare it with first time stamp. If the timestamps do not match then goto 2

sysinfo

- System z offers the StoreSystemInformation (STSI) instruction returns information about the system
- /proc/sysinfo returns most of the relevant data

```
linux07:~ # cat /proc/sysinfo
Manufacturer:         IBM
Type:                 2094
Model:                708
Sequence Code:        00000000000xxxxx
Plant:                02

CPUs Total:           10
CPUs Configured:      8
CPUs Standby:         0
CPUs Reserved:        2
Capability:           1456
Adjustment 02-way:    245
Adjustment 03-way:    238
Adjustment 04-way:    232
Adjustment 05-way:    226
...
```

```
...
Adjustment 06-way:    221
Adjustment 07-way:    216
Adjustment 08-way:    211

LPAR Number:          7
LPAR Characteristics: Shared
LPAR Name:            LINUX07
LPAR Adjustment:      421
LPAR CPUs Total:      2
LPAR CPUs Configured: 2
LPAR CPUs Standby:    0
LPAR CPUs Reserved:   0
LPAR CPUs Dedicated:  0
LPAR CPUs Shared:     2
```

Outlook

- Hypfs available under z/VM
- User space deamons for the monwriter

Related

- z/VM Monitor Domains und Record Formate
  - http://www.vm.ibm.com/pubs/ctlblk.html
- Linux appldata/monreader Dokumentation
  - http://www.ibm.com/developerworks/linux/linux390/ (Device Drivers, Features and Commands)
- z/VM 5.2 Dokumentation
  - http://www.ibm.com/servers/eserver/zseries/zos/bkserv/zvmpdf/zvm52.html
  - „z/VM: CP Commands and Utilities Reference“ (MONITOR, QUERY MONITOR)
  - „z/VM: Performance“ und „z/VM: Performance Toolkit“

Thank you

- cborntra@de.ibm.com

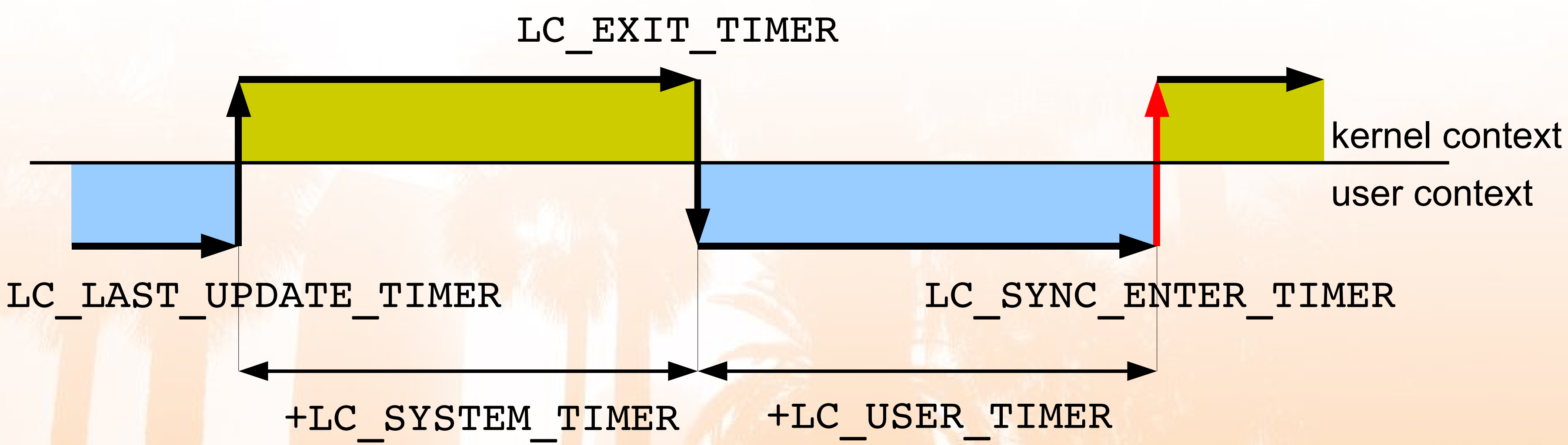
Backup Slides

Cpu time accounting implementation

- Add new architecture dependent data type cputime_t
  - Find all common code spots where to replace ticks with cputime_t
  - Use architecture dependent macros: cputime_add, cputime_sub, etc.
- Update cpu time counters on each context switch
  - System call, program check, i/o interrupt and ext. interrupt from userspace
  - Switches between processes, hard irq context and softirq context
- Timer ticks still serve a purpose
  - Update the TOD clock
  - Execute timer events
  - Transfer accumulated cpu time numbers to process

High precision user / system times (1)

- syscall
  - 1. stpt LC_SYNC_ENTER_TIMER
  - 2. LC_SYSTEM_TIMER += LC_LAST_UPDATE_TIMER - LC_EXIT_TIMER
  - 3. LC_USER_TIMER += LC_EXIT_TIMER - LC_SYNC_ENTER_TIMER
  - 4. LC_LAST_UPDATE_TIMER = LC_SYNC_ENTER_TIMER
- sysreturn: stpt LC_EXIT_TIMER



High precision user / system times (2)

```
        .globl  system_call
system_call:
        stpt    __LC_SYNC_ENTER_TIMER
        stmg    %r12,%r15,__LC_SAVE_AREA
        larl    %r13,system_call

        ...

        tm      SP_PSW+1(%r15),0x01        ; interrupting from user ?
        jz      svc_do_svc
        UPDATE_VTIME __LC_EXIT_TIMER,__LC_SYNC_ENTER_TIMER,__LC_USER_TIMER
        UPDATE_VTIME __LC_LAST_UPDATE_TIMER,__LC_EXIT_TIMER,__LC_SYSTEM_TIMER
        mvc     __LC_LAST_UPDATE_TIMER(8),__LC_SYNC_ENTER_TIMER
sysc_do_svc:

        ...

sysc_leave:
        mvc    __LC_RETURN_PSW(16),SP_PSW(%r15)
        lmg    %r0,%r15,SP_R0(%r15)
        stpt   __LC_EXIT_TIMER
        lpswe  __LC_RETURN_PSW
```

Overhead of CPU time accounting (1)

- Some instructions are added to the first level interrupt handler:
  - Two store cpu timer “stpt” instructions
  - A test and a branch on condition
  - Two 64 bit calculations of the form A = A + (B – C)
- Empty getpid() system call on z990:
  - with VIRT_CPU_ACCOUNTING=n: ~175 cycles
  - with VIRT_CPU_ACCOUNTING=y: ~210 cycles
- ~35 cycles added to the critical system call path (+20 %)
- Micro-Benchmark
  - Empty system call is the absolute worst case
  - Other performance tests show almost no decrease

Overhead of CPU time accounting (2)

- LMBench results:
  - Simple Syscall -18%
  - Pipe Latency -11%
  - Pipe Bandwidth -9%
  - Context Switch -3%
- DBench – no noticeable change
- iozone – no noticeable change
- specjbb2000 – no noticeable change
- Overall: normal workload should not slow down noticeably