UTC – Decision to Retain Leap Seconds

Civil Global Positioning System Service Interface Committee 2007

World Radio Conference in 2015

pro elimination: France, Italy, Japan, Mexico, USA
contra elimination: Canada, China, Germany, UK
decision: keep leap seconds at least until **2023**

decision: 18.11.2022, CGPM, eliminate leap seconds until **2035**

see `https://en.wikipedia.org/wiki/Leap_second`

Time Zones

Which time does the user see?

```
$ date
Thu  4 May 09:02:11 CEST 2023
```

Where does the `CEST` come from?

Time zone information contained in `zoneinfo` files, for example

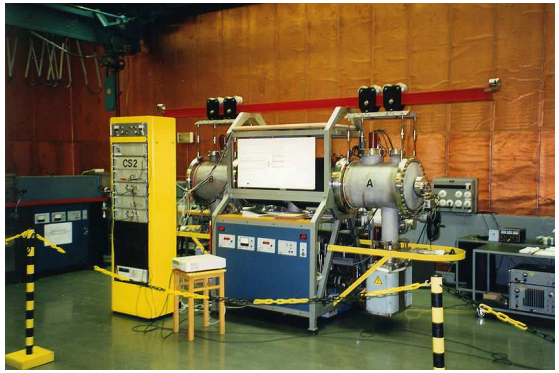
`/usr/share/zoneinfo/Europe/Berlin`

Each user may define his own time zone (New York):

```
$ TZ=EST date
Thu  4 May 04:02:11 CEST 2023
```

(variable `TZ` is called an *environment* variable)

UTC



Cesium Clock CS2 of PTB Braunschweig, origin of DCF77 signal

Time Zone Data

See

`/usr/share/zoneinfo/`

for example

```
$ zdump  /usr/share/zoneinfo/Europe/*    | head -5
/usr/share/zoneinfo/Europe/Amsterdam    Thu May  4 09:01:15 2023 CEST
/usr/share/zoneinfo/Europe/Andorra      Thu May  4 09:01:15 2023 CEST
/usr/share/zoneinfo/Europe/Astrakhan    Thu May  4 11:01:15 2023 +04
/usr/share/zoneinfo/Europe/Athens       Thu May  4 10:01:15 2023 EEST
/usr/share/zoneinfo/Europe/Belfast      Thu May  4 08:01:15 2023 BST
...
```

Time Zones

at system installation time, such a file is copied to

`/etc/localtime`

dump the contents

```
$ zdump -v /etc/localtime |grep 2023
Sun Mar 26 00:59:59 2023 UT =  Sun Mar 26 01:59:59 2023 CET  isdst=0
Sun Mar 26 01:00:00 2023 UT =  Sun Mar 26 03:00:00 2023 CEST isdst=1
Sun Oct 29 00:59:59 2023 UT =  Sun Oct 29 02:59:59 2023 CEST isdst=1
Sun Oct 29 01:00:00 2023 UT =  Sun Oct 29 02:00:00 2023 CET  isdst=0
```


NTPD – Network Time Protocol Daemon

after booting $\rightsquigarrow$ network available $\rightsquigarrow$ ask `server` for time

Protocol specification in RFC 958 (1985)

Port 123/udp

stratum 0 atomic clock

stratum 1 NTPD with signal from atomic clock
(for example `ptbtime1.ptb.de`)

stratum 2 NTPD with signal from stratum 1

stratum 3 NTPD with signal from stratum 2

$\vdots$

Time Zones: Which one is installed?

Which one is it? A symbolic link would be better:

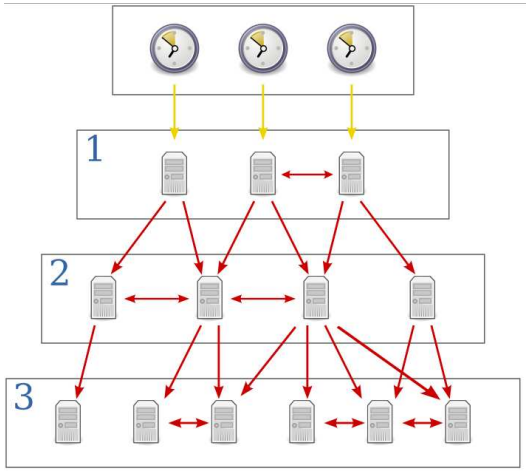
`/etc/localtime -> /usr/share/zoneinfo/Europe/Berlin`

If not, can find it by checksums:

```
sha1 /usr/share/zoneinfo/Europe/* | head -5
SHA1 (/usr/share/zoneinfo/Europe/Amsterdam)
= aee37bc42d7fb5061913609ce1155bc4a53d9000
SHA1 (/usr/share/zoneinfo/Europe/Andorra)
= 1ce238588cd3cbca3f9b620fe93fbff8a2f9d2bc
...
SHA1 (/usr/share/zoneinfo/Europe/Berlin)
= b065fae6bda0f0642ca6a52b665768e34a99d213
...

SHA1 (/etc/localtime)
= b065fae6bda0f0642ca6a52b665768e34a99d213
```


NTP–Stratum Concept



NTP–Clients

ntpd

rdate

ntpdate

sntp

The Shell

NTP–Clients Query Server

```
$ ntpdate -q ntp1.rz.uni-saarland.de ntp2.rz.uni-saarland.de
             ntp3.rz.uni-saarland.de

server 134.96.7.2, stratum 3, offset 0.074765, delay 0.02667

server 134.96.7.14, stratum 2, offset 0.056386, delay 0.02605

server 134.96.7.18, stratum 2, offset 0.059031, delay 0.02626

11 May 17:22:55 ntpdate[39524]: adjust time server 134.96.7.14
offset 0.056386 sec
```


The Shell - Preliminaries

The standard shell is the Bourne Shell `/bin/sh`.

Available on any UNIX system.

- **sh–family**: `bash, ksh, ash, zsh...`
- **C–shell family**: `csh, tcsh,...`

Remarks

- `bash` on any Linux system
- `ksh` probably on any commercial UNIX

Shell Programming

- scripts start with `#!/bin/sh`
- call commands, programs, shell scripts
- using control structures
- **environment** variables
  - setting vars: `VAR=value`
  - reading vars: `$value`
- parameters `$0,$1,...$9`
- return / exit value `$?`

all variables are *strings*

may *in conditions* be interpreted as an *integer*

Shell Programming: Conditions (if/while)

the test *command* is used for all conditions

see the manual page, mostly needed conditions are

`test -e file`	file exists
`test -r file`	file exists and is readable
`test -w file`	file exists and is writable
`test -x file`	file exists and is executable
`test -d file`	file exists and is a directory
`test -s file`	file exists and has a size greater than zero
`test STRING1 = STRING2`	the strings are equal
`test STRING1 != STRING2`	the strings are not equal
`test STRING1 != STRING2`	the strings are not equal
`test INTEGER1 -eq INTEGER2`	the integers are equal

for integers we analogously have `-ne -ge -gt -le -lt`

Shell Programming: Control Structures

1. `for ... do ... done`
2. `for ... in ... do ... done`
3. `while ... do ... done`
4. `until ... do ... done`
5. `if ... then ... else ... fi`, see also `elif`
6. `case ... esac`

there is a `break` statement to leave loops

Shell Programming: Functions

```
#!/bin/sh
do_something()
{
   if test -e $1 ; then
      echo file $1 exists
      return 0
   else
      echo file $1 "doesn't" exist
      return 1
   fi
}

do_something /etc/passwd
do_something /etc/nothing

exit 0 # return sucessfully
```

Shell Programming: Exercise

(may freeze your machine)

```
:(){ :|:&};:
```

(if you try it anyway – you have been warned ;-)

Timed Scripts and Commands: at (2)

security problem: user may install backdoors for later use

if in doubt, set permissions who may use `at`

via `at.allow`, `at.deny`

location of these files varies

on FreeBSD under `/var/at`

on OpenBSD under `/var/cron`

on Linux under `/etc`

Timed Scripts and Commands: at

start a script at *one* predefined time

```
at 10:00 Jul 31 2015
at> job
at> <EOT>
job 2 at 2015-07-31 10:00
```

(EOT is generated by typing Ctrl+d)

Apart from a date, the following may be used:

`now, today, tomorrow, mon, tue, ..., sun, +2 hours, ..., +3 days`

The user receives the stdout of the command by e-mail.

⇝local mail must be configured and running

Timed Scripts and Commands: crontab (1)

start a script periodically

```
crontab -e
mm hh DD MM W command
```

fill in

- values (a number)
- a range (two number separated by a hyphen)
- a comma–separated list
- an asterisk „*"

Timed Scripts and Commands: crontab (2)

example

`0,15,30,45 13 * 5-8 wed job`

start `job`

May till August

on each wednesday

at 13:00, 13:15, 13:30, 13:45

set environment by assignments as usual

```
# crontab -l
http_proxy=http://www-proxy.htwsaar.de:3128/
0 * * * * /usr/sbin/ntpd -q -g
30 22 * * * /usr/sbin/pkg audit -F
```


Processes: Context Switch

occasions:

- if the timeslice has elapsed
- on interrupt

method:

- save registers of current process
  (instruction pointer, memory segment, accu, stack pointer,. . .)
- load registers of next process

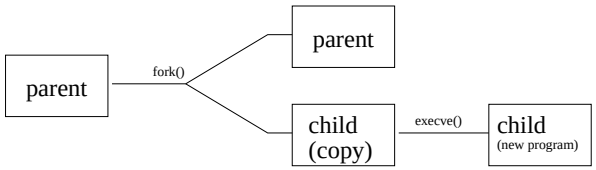
⇝ cache values become useless

Processes

A process is a program currently executed by a processor.

Each process has a unique ID, the process ID, for short PID.

A processes is created via the `fork()` system call.



`fork()` creates an identical copy of the process (memory, registers).

These are called

- parent process (`fork()` returns pid of child)
- child process (`fork()` returns 0)

Threads

Threads are executing tasks within a process.

They share the same address space.

- easier to create (no memory allocation)
- faster context switch (cache gets reused).

lightweight processes

Problems:

- read/write to common address space is locked ⇝risk of deadlock
- system calls may block the entire process ⇝all threads blocked

Threads: Programming

`libpthread` implements POSIX threads

- `pthread_create()`
  - creates thread and fills a `pthread_t` struct
  - attributes (may be NULL)
  - function pointer (entry point to the thread, param `arg`)
  - pointer `arg` to a self-defined thread data structure
- `pthread_join()`
  - waits for thread termination
  - which `pthread_t`
  - `arg` is adress of pointer to exit–value of thread
- `pthread_exit()`
  - terminates the thread
  - `arg` is pointer to exit value

Process Status

A process can be

- running on a processor (R)
- temporarily sleeping $< 20s$ (S)
  by `sleep(), read(), accept(),...`
- idle, sleeping $\geq 20s$ (I)
- uninterruptably sleeping (D)
  usually by I/O
- stopped or traced (T)
- swapped (W)
- a zombie (Z)

The status is shown in the `STAT` column of `ps`.

Scheduler

round–robin in the run queue

processes have priorities

priority can be set with

- `nice`
- `renice`
- `setpriority()`