CERN School of Computing

Data Technologies
Exercises

1.1.1

HINT 1 (1)

1. **bash has an intrinsic time command with %% time resolution**
2. **the GNU program /usr/bin/time has only % time resolution**

Run in the bash shell 'time sleep 1' – not '/usr/bin/time sleep 1'

1.1.2

HINT 1(1)

Most programs you will measure with time will have longer startup times than a sleep command. The implementation of a 'sleep 1' is just one systemcall

```
nanosleep({1, 0}, NULL)      = 0
```

You can consider the overhead you measure as a lower boundary for a systematic error in the measurement!

1.1.3

HINT 1(1)

The `gettimeofday` function in Linux is able to return you the current time with a microsecond resolution.

sleep is also a function in the standard C library!

1.2.1/2

HINT 1(3)

Read in the man page 'SORTING of taks window'

by memory

and

by cpu consumption!

In any case you leave top with 'q' and you can enforce an update
by using SPACE

1.2.1/2

HINT 2(3)

2 Hz means twice per second

Search in the manual page for

'COMMAND-LINE Options'

'Delay time'

1.2.1/2

HINT 3(3)

You can sort by physical memory usage pressing 'M' in the interactive top window.

You an switch to cpu usage sorting pressing 'P'.

You can set the update frequency pressing 's' or 'd' and then the interval time e.g. 's1' for 1 Hz.

1.2.3

HINT 1(1)

You can run vmstat with 1Hz updates like:

```
> vmstat 1
```

The first argument is time between two outputs.

You can calculate the total used cpu time as (us + sy).
[user + system time]

1.3.1

HINT 1(2)

```
> yes
```

This command writes in an infinite loop ***'y\n'*** to STDOUT.

You can redirect STDOUT to a file like:

```
> yes > /tmp/yes.out
```

You can interrupt the running command using Control-C

1.3.1

HINT 2(2)

IO rate = <Mb written> / <time to write>

1.3.2

HINT 1(1)

To trace system calls you just prepend your command line with the 'strace' command:

```
> strace yes
```

Compare the blocksize used for yes on the terminal and in case of /dev/null or file redirection!

1.3.3

HINT 1(1)

You can write arbitrary strings to STDOUT with yes in an infinite loop using the syntax:

```
> yes <string>
```

1.3.4

HINT 1(2)

Try to identify your hard disk in the *iostat* window and read the kb/s read value. Certainly you can trust these values only if you are sure no other applications use your device at the same time.

If you use *vmstat* with a 1Hz output it is easier to see, at which time data is written out to disk.

1.3.4

HINT 2(2)

'no cache' means every write call is going directly to write on disk without caching.

'write-through' means every write call is writing through the cache on disk

'write-back' means write calls are first only written into the cache and in regular intervals or under memory pressure pages get written on disk.

1.4.1

HINT 1(1)

Inspect the system calls for reads on the source and writes on the target!

1.4.2

HINT 1(1)

Measure the *realtime* with the *time* command first and then repeat the same inserting '*strace*' on the commandline before the executable name.

1.5

HINT 1(2)

Try to add the '-f' flag to the *strace* command to follow detaching programs!

1.5

HINT 2(2)

Check the CPU consumption and process PIDs using *top*!

Redirect the output of 'strace -f <cmd>' to a file and try to understand the syscall pattern. Lookout for fork & execv statements!.

Just count how many times certain events occure!