Course TÖL401G: Stýrikerfi / Operating Systems Assignments 1 & 2: Solution

Note

- The solutions shown in the following cover all the possible different answers (and provide some additional background information).
- As such, they are much more detailed than what a student is expected to submit.

Assignment 1: Question

- Discuss whether operating systems are needed at all! Consider for example the situation where only one single fixed application program is executed on a hardware (like in an embedded system such as the software running on the electronic control unit hardware of a car, e.g. for motor control).

- In case you come to the conclusion that operating systems are not necessarily needed: why is it nevertheless reasonable to have an operating system in-between an application program and the hardware?

Assignment 1: Answer (1)

- If you have only one single fixed application, an OS is not necessarily needed:
  - Every service (listed in section 2.1 of ch. 2) that an OS typically provides can be implemented by the application itself.
    - In the example of a an embedded system, anyway often only support for a few I/O devices is needed (but none of the other services).
    - Everything else is not needed or very basic, e.g. only the single fixed application needs to be "loaded" (in fact it is typically provided by ROM and thus already loaded), typically no additional memory is requested at run-time, user-interface is non-existent or minimal via the I/O devices, no storage/file system needed, etc.

Sections 1.6-1.9 of ch. 1 cover these such services (and later, ch. 2 covers this in section 2.1).

Assignment 1: Answer (2)

- However, as soon as a second application may be running, a separate OS is reasonable:
  - In principle, each application could implement it's own OS-like services (as described before).
  - However, as soon as resources are shared between these processes, the access to the resources needs to be coordinated: if application 1 does not know about application 2 and vice versa, such a coordination is not possible. Instead, a separate instance (=the OS) is needed that coordinates access to shared resources.
  - Example of a shared resource: hard disk.
    - Imagine application 1 stores data on some location of the hard disk and application 2 accidentally decides to store data on the same location (because application 2 does not know about the fact application 1 is using the hard disk as well): data would get overwritten. A separate OS that is used by both application can avoid this, because it has a global view.

Assignment 1: Answer (3)

- Well, you could write application 1 and 2 in a way that they are aware of each other and coordinate their access to shared resources.
- However, then you have to implement the same OS services in two different applications. This would be unnecessary work and redundant. $\Rightarrow$ Better to have it at one location: the OS!

- Even if you do not need an OS (because only 1 application is running), it may pay off to have an OS, e.g. because:
  - It will be easier to port that application to a different hardware by just exchanging the OS (without needing to modify the application).
  - If another single application needs to be written it is likely that this other application will need similar OS-like services: instead of rewriting them again from scratch, having them in a separate OS allows to re-use them.
  - Separating application and OS-like services introduces some layering (see advantages of layering in ch. 2).

Excursion: Example system without OS: Arduino

- Cheap computer ($10-$20):
  - 16 MHz 8 bit CPU, 2 KB RAM (for variables),
  - 32 KB Flash Memory (for program code),
  - USB + Pins for digital and analogue signals.
- Used by do-it-yourself makers to automate all kind of things.
- Programmed using a PC to compile code and to send code to Arduino.
- Has no OS, only a firmware that runs after power on:
  - If PC sends code: receive code from USB and store in flash memory.
  - Otherwise: execute code in flash memory.
- Runs only the one single program loaded to flash:
  - Program is in full control of hardware: can use complete RAM on it's own.
  - The only needed OS-like functionality would be:
    - Access to USB port (typically only used by firmware for receiving new program).
    - Access to I/O pins (i.e. simple).
    - Source code libraries exist: re-use these functions for, e.g., I/O.

Assignment 2: Question

- Explain the POSIX standard!
  `http://pubs.opengroup.org/onlinepubs/9699919799/`
  Take care to cover briefly:
  - 1. which standardisation bodies standardised POSIX, what is the number and year of the latest standard, and what POSIX is about in general,
  - 2. what of each of the volumes “Base Definitions”, “System Interfaces”, “Shell & Utilities” is about
    (1 or 2 sentences per volume is sufficient),
  - Names of an example operating systems that is
    - Fully POSIX compliant,
    - Mostly POSIX compliant.

“Standardisation body” is just a fancy name for a standardisation organisation such as “ISO”.

Assignment 2: Answer 1(a)-(b)

- 1. (a) POSIX is standardised by IEEE (Institute of Electrical and Electronics Engineers) and The Open Group (an industry consortium of vendors of Unix systems).
  - They work together in a group called "The Austin Group".

- 1. (b) The latest POSIX standard has number "POSIX.1-2017" and is simultaneously
  "IEEE Std 1003.1-2017" and "The Open Group Technical Standard Base Specifications, Issue 7, 2018 Edition".
  - Notes:
    - Each standardisation body has it's own naming/numbering scheme.
    - (For this assignment, sufficient to name 1 out the above 3 names/numbers.)
    - Older versions of these standards exist: important to know the year!

Assignment 2: Answer 1(c)

- 1. (c) POSIX defines
  - a standard operating system interface
    - (=system calls and their parameters, i.e. functions that an application programmer can call to make use of OS services)
  - and environment,
    - including a command interpreter ("shell"),
    - and common utility programs

  to support applications portability at the source code level.
  - => A shell script that works on one POSIX OS runs an any other POSIX OS.
  - => Source code of an application program that works on one POSIX OS runs an any other POSIX OS
    - (re-compilation may be necessary, i.e. source code compatible, not binary compatible).
  - Assumption: only functionality defined in POSIX is used.

  - POSIX refers to what people typically expect to be a "Unix"-like operating system.

Assignment 2: Answer 2. (a)-(c)

- The POSIX standard consists of several "volumes", e.g.:
  - 2 (a) Volume "Base Definitions": Defines a general environment that a user experiences and can expect, e.g.:
    - Allowed characters in a file name,
    - Standard directories such as **`/tmp`** for temporary files,
    - File access permissions based on access rights for owner of file, group to which file belongs, and all others,
    - Environment variables such as PATH that contains the directories in which executable files are searched for.
  - 2 (b) Volume "System Interfaces": Defines the system calls to be provided by a POSIX system, i.e. an API using C headers and a description of input and output parameters as well as the behaviour of the system call, e.g. **`write(…)`** operation to write data to a file.
  - 2 (c) Volume "Shell & Utilities": Defines the commands of the command-line interface (the shell), the syntax of shell scripts and some standard executable programs such as "**`cp`**" to copy files or "**`awk`**" to process files.
  - There is a new volume "Rationale" that gives some background info.

Assignment 2: Answer 3. (a)-(b)

- 3 (a) POSIX-compliant OSes (according to http://en.wikipedia.org/wiki/POSIX from 2019), e.g.:
  - Apple Mac OS X, Oracle/Sun Solaris, Hewlett-Packard HP-UX, IBM AIX
    - (I.e. most commercial Unix-like OSes are fully POSIX-compliant: it is a selling point to be POSIX-compliant.)
- 3 (b) Many open-source Unix-like OSes do almost fulfill the POSIX standard, but typically in some (often minor) parts they behave slightly different even though they offer mostly the full POSIX API, e.g.:
  - Linux (for example, thread handling is slightly different from POSIX),
  - FreeBSD, NetBSD, OpenBSD.
- Microsoft Windows is not at all POSIX-compliant: Completely different API!
  - Cygwin provides a substantial part of POSIX on top of MS Windows by mapping POSIX API on Windows API.
    - Can be used to run POSIX shell scripts & to compile and run POSIX-compliant source code.
  - Windows Subsystem for Linux (since Windows 10, 2016): allows to run even Linux binaries.

That’s all!

- Any questions, e.g.
  - Alternative solutions?

Outlook: Assignment 3 and 4

- Both assignments are based on chapter 2.

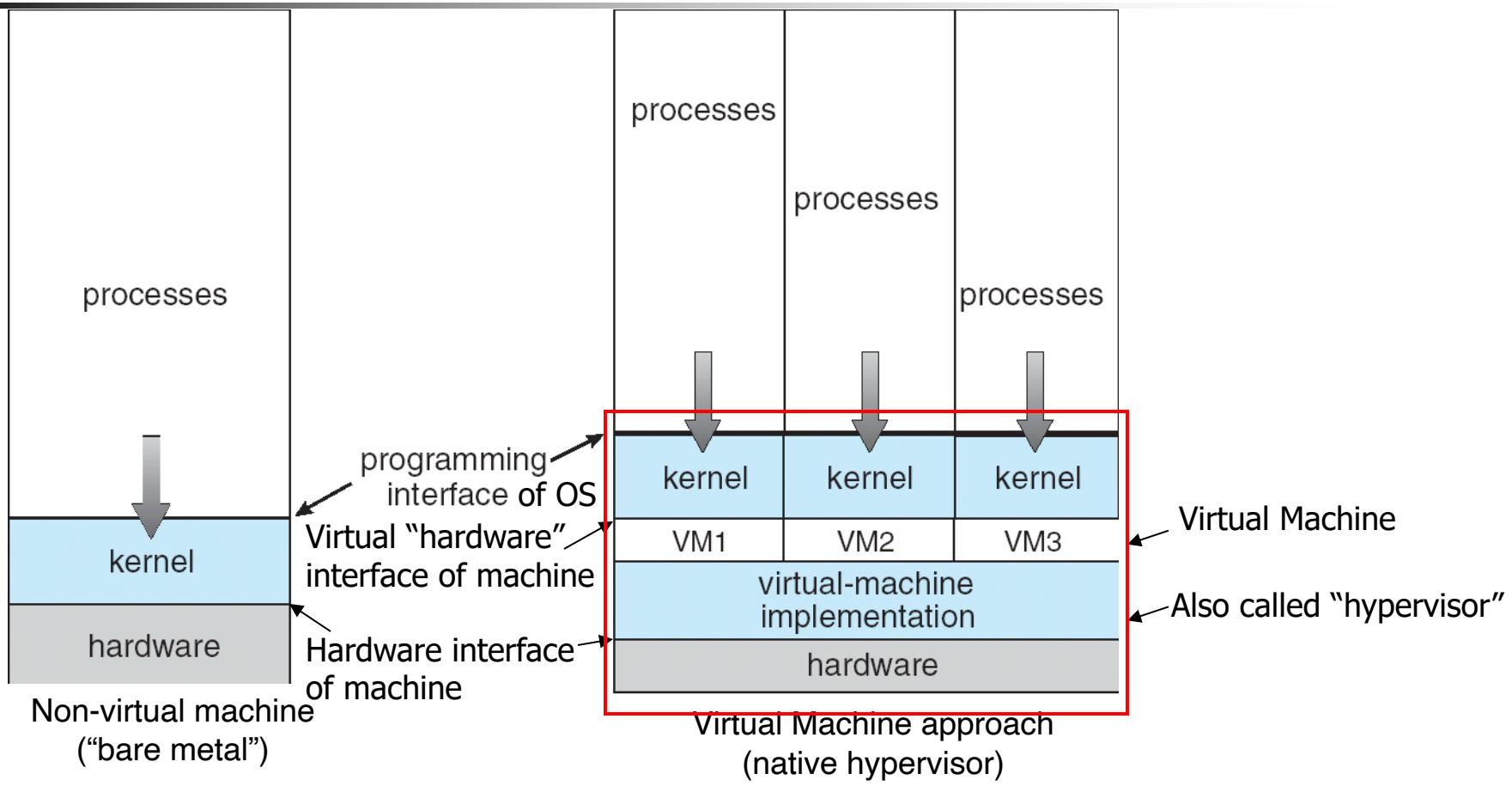
Assignment 3: Virtual machines

- Choose either Assignment 3a or 3b:
  - Assignment 3a: if you prefer to do some paper work (and web search) on virtual machines;
    - Experience shows that students have problems with this Assignment 3a, hence you might score a better grade with Assignment 3b.
  - Assignment 3b: if you prefer to experiment and have a fast Internet connection and 3GB space left on your disk in order to download, install a virtual machine running Linux as guest OS.
    - Download at home via optical fibre: approx. 11 minutes, alspo available from teacher during Thursday class on USB key-VM image.
      - Do not change filename of VM image!
    - Not intended to have further assignments based on this VM!
      - So no problem to do Assignment 3a in instead of 3b.

Assignment 3a Questions

- Consider a virtual machine for running multiple operating systems (OS1 and OS2) on top of it:
  - 1. The scheduler of the VM hypervisor has just granted OS1 CPU time for 20ms. While OS1 is running in the granted time slot, the timer of OS2 for triggering the CPU scheduler of OS2 expires. Describe two different options how the VM hypervisor could deal with this! (Assume single processor/single core system!)
  - 2. In a virtual machine scenario, the different virtual machines are really completely separated from each other: OS1 running on VM1 can neither access the main memory nor the les of OS2 running on VM2. Describe one possible solution if you nevertheless want to exchange data between different virtual machines.

From ch.2: Non-Virtual Machine vs. Virtual Machines



- Native hypervisor: Runs on top of bare hardware – all OSes run on top of it.
- Hosted hypervisor: Runs on top of an ordinary OS that runs on top of bare hardware.

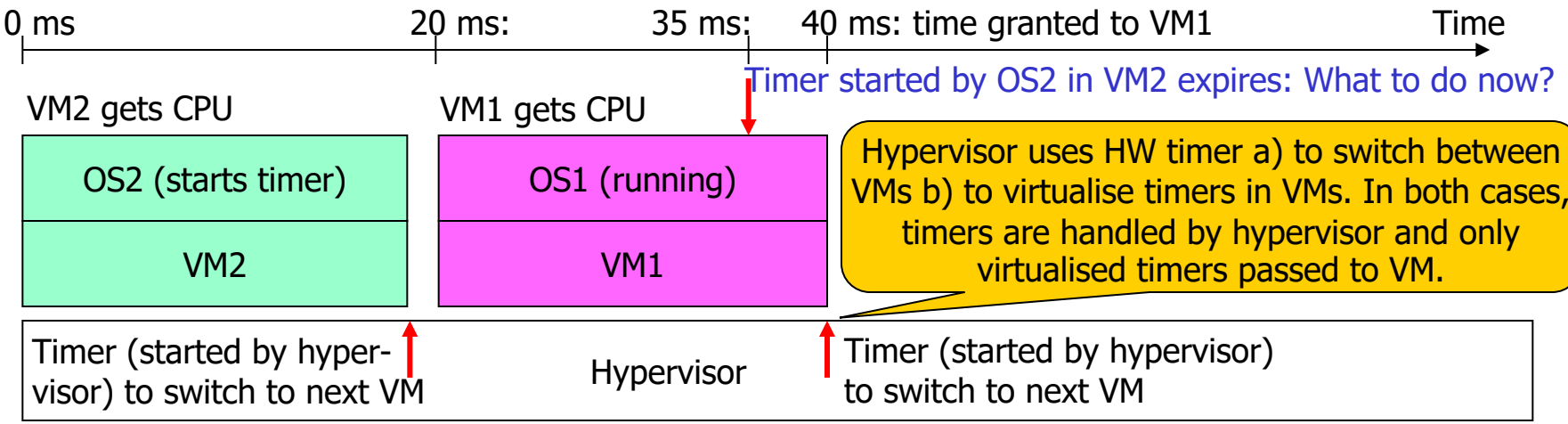
From ch.2: Implementation of VMs: CPU virtualisation

- Hypervisor must virtualise all hardware resources:
  - CPU:
    - Hypervisor has CPU scheduler for sharing CPU time between guest operating systems.
      - Similar to CPU scheduler of timesharing/multi-tasking OS.
    - User mode/kernel mode: Guest OS must not be able to access resources directly using kernel mode. (Nevertheless, guest OS assumes that all of its interrupt handlers are running in kernel mode and thus tries to access physical devices instead of virtualised ones.)
      - VM must provide a virtual kernel mode and virtual user mode. However, even in virtual kernel mode, the guest OS is actually in physical user mode.
      - If guest OS performs an action that requires physical kernel mode, this will lead to an interrupt → interrupt handler of hypervisor is called: hypervisor (running in physical kernel mode) will analyse action and perform a corresponding action on behalf of the guest OS and finally resumes execution of guest OS instructions (in virtual kernel mode).

Modern CPUs may provide additional features to support virtualisation, e.g. additional CPU modes ("real kernel mode" for hypervisor, virtualised kernel mode for OS, user mode.)

Assignment 3a Question 1 Graphical view

- Consider two virtual machines for running operating systems OS1 and OS2 inside them:
  - 1. The scheduler of the VM hypervisor has just granted OS1 CPU time for 20ms. While OS1 is running in the granted time slot, the timer of OS2 for triggering the CPU scheduler of OS2 expires. Describe two different options how the VM hypervisor could deal with this! (Single processor/single core assumed!)



Assignment 3a Question 2

- Consider a virtual machine for running multiple operating systems (OS1 and OS2) on top of it:
  - 2. In a virtual machine scenario, the different virtual machines are really completely separated from each other: OS1 running on VM1 can neither access the main memory nor the les of OS2 running on VM2. Describe one possible solution if you nevertheless want to exchange data between different virtual machines.

- Hint: Google is your friend!
  - As always: state your sources.
- Or install a VM hypervisor and find out how it is done there.
  - But then you should better do Assignment 3b.

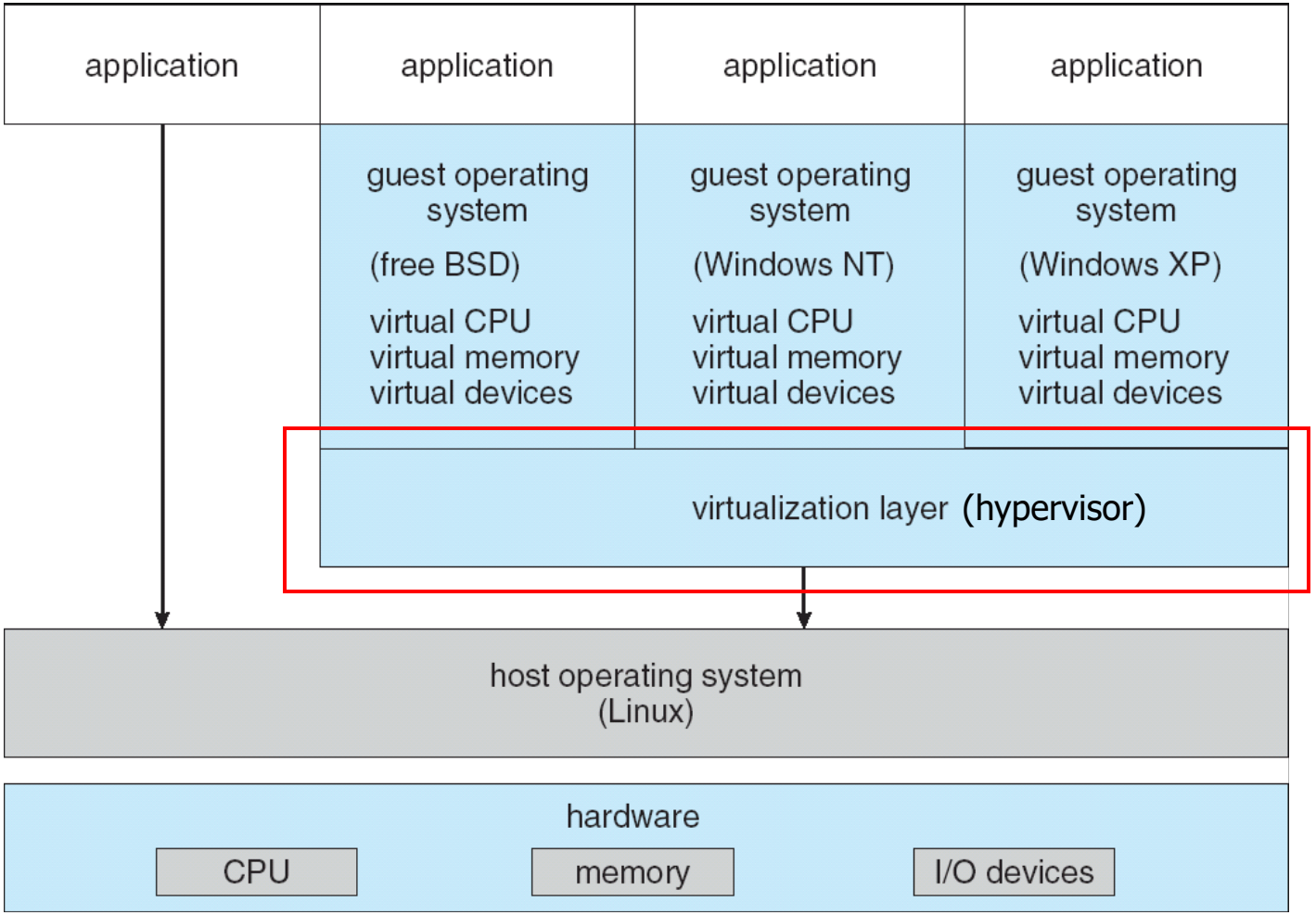
Assignment 3b Questions 1.-4.

- Install the hosted hypervisor VirtualBox (runs on Windows, Mac, Linux) **`https://www.virtualbox.org`**
  to answer the questions below:
  - 1. While you should not download and run unknown software from untrusted sources, why is it not a problem to download and run the VM image for solving this assignment?
  - 2. Create CPU load inside the guest OS: describe briefly how this CPU load affects your host system in terms of CPU load?
  - 3. Now, do the opposite by creating load on your host system: describe briefly how this CPU load affects your guest system in terms of CPU load.
  - 4. Are the effects reported by the system monitors the same in the two above cases? If not, how can the different monitoring results be explained?

Assignment 3b Hints

- You need 3 GB free disk space.
- If you have another VM hypervisor than VirtualBox or another VM image (even not containing Linux), this is also OK.
- Check Helmut's Panopto video on using Virtual Box.
- **`http://people.westminstercollege.edu/faculty/ggagne/osc/vm/index.html`** contains usage infos, includes VM image.
  - Faster download: **`http://notendur.hi.is/helmut/OSC-2016.ova`**
  - If you have an MD5 checksum tool (e.g. **`md5sum OSC-2016.ova`**) to check integrity of file: 93f6b9a6570f70af4f717af2af585d84.
    - Do not change filename of VM image.
  - If VirtualBox gives error: Have virtualisation support enabled in BIOS/UEFI.
- Default keyboard layout assumed by the Linux instance is US, e.g.: "/" is next to your right shift key, where þ. is on an Icelandic keyboard.
  - But you can change keyboard layout inside Linux guest system.

From ch.2: Example: Hosted hypervisor (e.g. VMware Workstation Player)



Some device drivers of guest OS need to be exchanged by device drivers that make calls to the underlying hypervisor instead of directly accessing hardware (para-virtualisation).

Support for hypervisor needs to be provided by host OS.

Host OS runs directly on bare hardware.

Assignment 3b Questions 2.-3.

- 2. Create CPU load inside the guest OS: describe briefly how this CPU load affects your host system in terms of CPU load? (E.g. "CPU Load increases", "Load stays same", etc.)
  - Open terminal and type: `yes >/dev/null` (CTRL-C to stop)
- 3. Now, do the opposite by creating load on your host system: describe briefly how this CPU load affects your guest system in terms of CPU load.
  - If your host is Linux or Mac OS: `yes >/dev/null`
  - If your host is Windows: create a file that contains the following lines, save it into, e.g., your home directory, in a file of type `.BAT`, and execute that file (CTRL-C to stop):

```
@echo off
:loop
goto loop
```

Assignment 4 Question

- Describe the steps of the system boot process until the login prompt of the operating system appears.
- While slides 2-51 to 53 focus mainly on the first half of system boot, you should focus on what happens after the bootstrap program/boot code has loaded the kernel and has just handed over control to the kernel.
  - Slide 2-53 describes this only in three sub bullet points.
  - Assume that the loaded kernel has a design based on modules! ($\rightarrow$hints on later slide)
    - Hint: Kernel modules explained in section 2.7 and video.

From ch.2: Module-based Operating System

- Most modern operating systems (e.g. Linux, Solaris, Mac OS X, even Windows) implement kernel modules:
  - Kernel consists only of core functionality.
  - Functionality can be added by loading kernel modules at run-time.
    - Similar to object-oriented approach:
      - Each class of kernel module (file system, device driver etc.) has a well defined interface.
      - Each kernel module implements this interface.
  - Mixture of Microkernel and Layers, but
    - Faster than microkernel: no message passing required for communication, instead: kernel and kernel modules run in kernel space and may use ordinary function calls to call each other.
    - More flexible than layered approach: each class of kernel module may call functions from another class of kernel module (no strict hierarchical layers), additional kernel modules may be added at run-time.

Assignment 4: Hints

- As the Linux kernel is module-based, loading just the kernel into memory is not sufficient to access all devices:
  - To access a device, a driver is needed that is provided by a kernel module.
    - In particular, the device drivers for accessing the storage device and a file system module to access the files that are stored on the storage device are needed.
  - These (and other) kernel modules are stored in files that need to be loaded from a file system that is stored on some storage device.
  - However, the kernel modules to load the modules (=for accessing storage device and files system) have not been loaded, yet (chicken-egg problem)!
    - **Take care to cover the solution to this problem!**
- You can stop your description at the point when the operating system has decided which program (=which filename) to start as very first process (that will take care of that the login prompt gets displayed).
  - **Enough to stop with describing how the decision for this program is made.**