OS Overview *

Jens Lechtenbörger

IT Systems, Summer Term 2025

Contents

1 Introduction

1.1 Recall: Big Picture of IT Systems

- Explore **abstractions bottom-up**
  - **Computer Architecture**: Build computer from logic gates

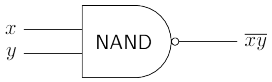

Figure 1: "NAND" under CC0 1.0; from GitLab

    * Von Neumann architecture
    * CPU (ALU), RAM, I/O

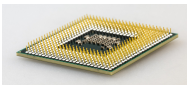

Figure 2: "CPU" under CC0 1.0; cropped and converted from Pixabay

  - **Experiment with OS concepts**
    * Explain core OS **management** concepts, e.g., processes, threads, virtual memory
    * Use GNU/Linux command line and explore system

*This PDF document is an inferior version of an OER in HTML format; free/libre Org mode source repository.

Figure 3: "Tux, the Linux mascot" under CC0 1.0; from Wikimedia Commons

  - **Experiment with containerization** for cloud infrastructures
    * Explain core concepts
    * Build images, run Docker **containers** and Kubernetes cluster

Figure 4: "Kubernetes logo" under Kubernetes Branding Guidelines; from GitHub

Figure 5: "Docker logo" under Docker Brand Guidelines; from Docker

1.1.1 OS Responsibilities

Warning! External figure **not** included: "What does your OS even do?" © 2016 Julia Evans, all rights reserved from julia's drawings. Displayed here with personal permission.
(See HTML presentation instead.)

1.1.2 Definition of Operating System

- Definition from (Hailperin 2019): **Software**
  - that **uses hardware** resources of a computer system
  - to provide support for the **execution of other software**.

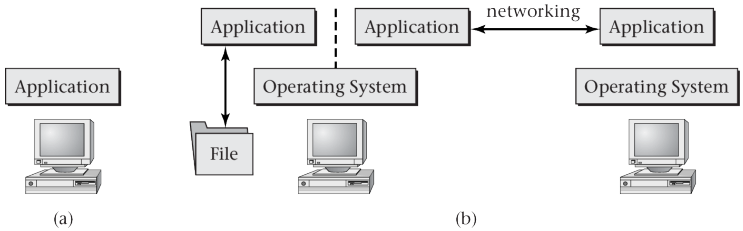

Figure 6: "Figure 1.1 of (Hailperin 2019)" by Max Hailperin under CC BY-SA 3.0; converted from GitHub

1.2 Prerequisite Knowledge

- Be able to write, compile, and execute small *Java* programs
  - What is an object? What is the meaning of `this` in Java?
  - How do you execute a program that requires a command line argument?
- Be able to explain basic *data structures* (stacks, queues, trees) and *algorithms* (in particular, hashing)
- Being able to explain the database *transaction concept* and *update anomalies*

2 OS Plan

2.1 Big Picture of OS Part

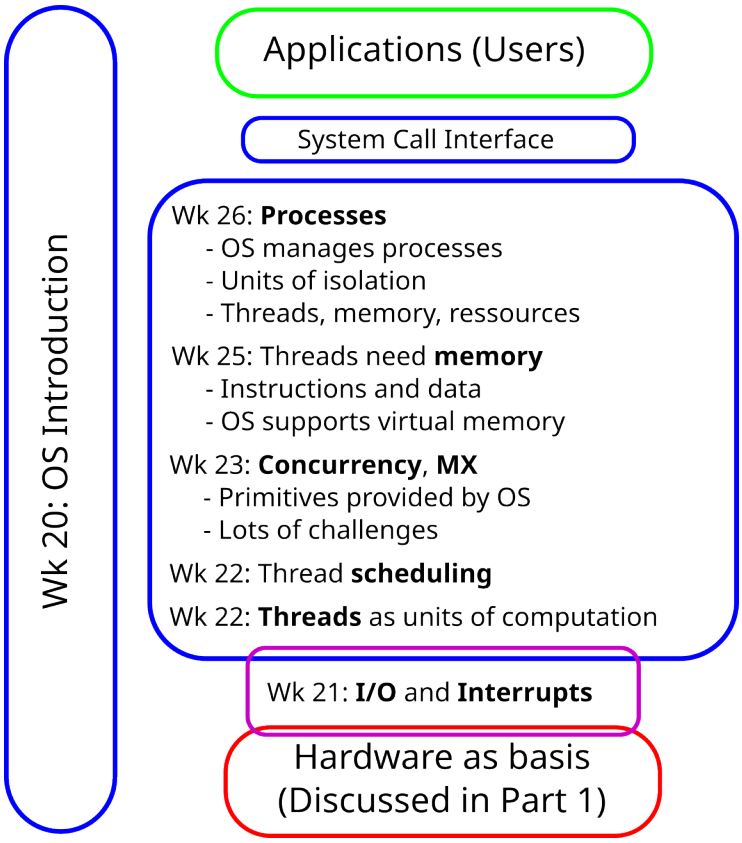

Figure 7: OS course plan, summer 2025

Although the Hack computer does not have an OS, it will help to recall how you interact with that machine. On Hack, you are able to run a single program, where you access hardware directly. E.g., reading from the keyboard requires access to a special memory location that represents the state of the underlying hardware device.

With OSs, applications no longer have direct access to hardware. Instead, OSs manage applications and their use of hardware. You will learn that the core part of OSs is called kernel, and each vendor's kernel comes with a specific interface to provide functionality to applications, usually in the form of so-called system calls. E.g., when you access `System.in` in Java to read keyboard input, the Java runtime executes a system call to ask the OS for keyboard input.

Starting from system calls, we will look into techniques for input/output processing (I/O for short) such as access to the keyboard. Recall that in Hack you programmed a loop to wait for keyboard input. Clearly, such a loop keeps the CPU busy even if no key is pressed, wasting the resource CPU if other applications could perform useful computations. Hence, I/O is usually paired with interrupt processing, which does not exist in Hack. Briefly, interrupts indicate the occurrence of external events, and OSs come with interrupt handlers. Then, for example, keyboard processing code only runs in response to key presses.

In contrast to Hack, OSs manage the execution of several applications, and each application might contain several so-called threads of execution. It is up to application programmers to decide how many threads to use for a single application (and you will create threads in

Java).

The OS manages all those threads and their scheduling for execution on the CPU (or their parallel execution on multiple CPU cores). Usually, scheduling mechanisms involve time slicing, which means that each thread runs for a brief amount of time before the OS schedules a different thread for execution. Such scheduling happens in intervals of about 10 to 50 milliseconds, creating the illusion of parallelism even if just a single CPU core exists. Such time-sliced or parallel executions are also called concurrent executions.

If shared resources are accessed in concurrent executions, subtle bugs may arise, a special case of which are update anomalies in database systems. The notion of mutual exclusion, MX for short, generalizes several synchronization mechanisms to overcome concurrency challenges, and we will look at typical related OS mechanisms and their use in Java.

Just as in Hack, instructions and code of applications need to be stored in memory. Differently from Hack with its Harvard architecture, code and instructions are stored uniformly in RAM in our von Neumann machines, and the OS manages the allocation of RAM. Importantly, mainstream OSs provide support for virtual memory, which does not exist in Hack. Briefly, with virtual memory the OS manages the allocation of memory in the form of RAM and disk space to all running applications.

Bringing all these concepts together, mainstream OSs manage processes as abstraction for resource management of applications, where the OS isolates different processes from each other. Importantly, threads of a single process cooperate and share resources (such as virtual memory or files).

To sum up, this figure visualizes what OS topics will be discussed when, and how topics build upon each other.

Note that some parts of this figure are hyperlinked to other presentations, which the mouse pointer should indicate.

2.2 A Quiz

Bibliography

Hailperin, Max. 2019. *Operating Systems and Middleware – Supporting Controlled Interaction*. revised edition 1.3.1. `https://github.com/Max-Hailperin/Operating-Systems-and-Middleware--Supporting-Controlled-Interaction`.

License Information