Computer Systems

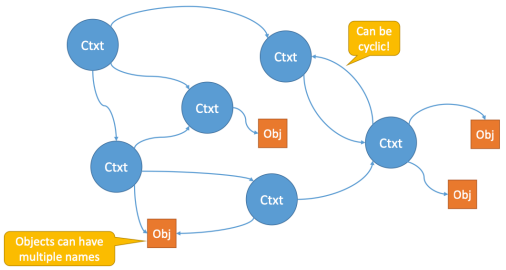
by dcamenisch

1 Introduction

This document is a summary of the 2022 edition of the lecture *Computer Systems* at ETH Zurich. I do not guarantee correctness or completeness, nor is this document endorsed by the lecturers. If you spot any mistakes or find other improvements, feel free to open a pull request at TODO: INSERT LINK. This work is published as CC BY-NC-SA.

2 Naming

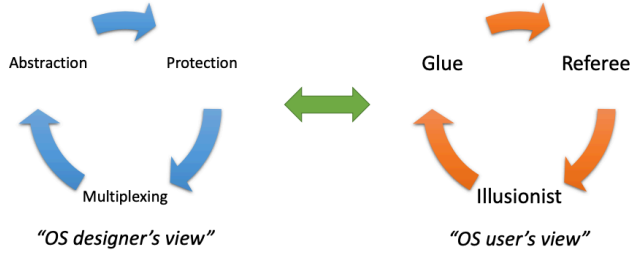
Naming is a fundamental concept, it allows resources to be bound at different times. Names are bound to objects, this is always relative to a context. One example of this would be path names, e.g. */usr/bin/emacs*. Name resolution can be seen as a function from context and name to some object. The resolved object can be a context in itself. This gives us a naming network.



Both synonyms (two names bound to the same object) and homonyms (the same name bound to two different object) can occure.

3 The Kernel

These are the main functions of any operating system. Commonly they are refereed to from a designer's view, but the user's view can be much more helpful to actually understand how it works.



3.1 Bootstraping

The term bootstraping referees to pulling himself up from his own boots. In computer systems it is why we call the process of starting up a computer booting. This boot process looks like this:

1. CPU starts executing code at a fixed address (Boot ROM)
2. Boot ROM code loads 2nd stage boot loader into RAM
3. Boot loader loads kernel and optionally initials file system into RAM
4. Jumps to kernel entry point

The first few lines are always written in assembly, but generally we want to switch to C as soon as possible.

3.2 Mode Switch

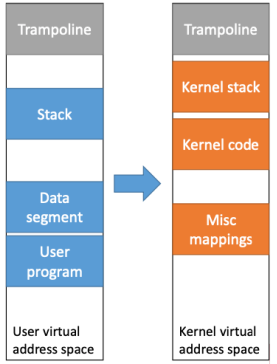
One of our main goals is to protect the OS from applications that could harm it (intentionally or not). For this purpose we introduce two different modes:

- Kernel mode - execution with full privileges, read/write to any memory, access and I/O, etc. code here must be carefully written
- User mode - limited privileges, only those granted by the OS kernel

These two (or more) modes are already implemented in hardware. The main reason for a mode switch is when we encounter a processor exception (mode switch from user to kernel mode). If this is the case, we want the following to happen:

1. Finish executing current instruction
2. Switch mode from user to kernel
3. Look up exception cause in exception vector table
4. Jump to this address

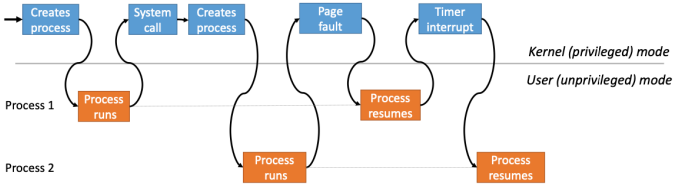
Further we may also want to save the registers and switch page tables. When switching between the modes we also have to change our address space, but we might want to access some informations from the user mode address space. One way of doing this is to use a so called **trampoline**, this is a part of the address space that gets mapped to the same location in user and kernel mode.



Mode switches can also occur the other way around (from kernel mode to user mode). The mains reasons for this are:

- New process / thread start
- Return from exception
- Process / thread context switch
- User-level upcall (UNIX signal)

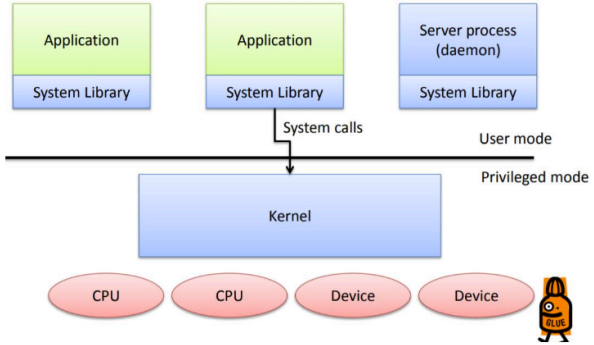
This leads us to the following perspective:



The mode switch is fundamental to modern computers:

- It enables virtualization of the processor
- It creates the illusion of multiple computers
- It referees access to the CPU

3.3 General Model of OS Structure



- Kernel - code that runs in kernel mode
- System Library - interface to kernel, enough for most programs
- Daemon - user space process which provides OS services, varying levels of privileges

We can differentiate between monolithic kernels and microkernels, depending on the amount of code in kernel mode.

3.4 System Calls