#uni/semester3/Betriebssysteme/chapter4/Processes

A process is a running program.

HOW TO PROVIDE THE ILLUSION OF MANY CPUs?

OS creates the illusion that many virtual CPUs exist when in fact there is only one physical CPU. Basic technique is **time sharing** of the CPU, allowing the user to run as many concurrently processes as he wants. Potential cost is **performance**, as each will run more slowly if the CPU must be shared.

Low level machinery (**mechanisms**) and some high level intelligence are needed to implement virtualisation of the CPU. Mechanisms are low level methods or protocols that implement a needed piece of functionality.

Context switch gives the OS the ability to stop running one process and start running another on a CPU.

Scheduling policy in the OS will make the decision, which program of many should be executed.

Time sharing makes an entity to run on CPU for a little and another for a little, thus the illusion. **Space sharing** is for assigning a resource (disk space) to entities. But only one entity has access.

Program ➜ static code and static data
Process ➜ dynamic instance of code and data

The **process machine state** of a process is what a program can read or update when it is running.

- ➜ Everything that the running code can affect or be affected by registers
- ➜ Address space is the memory that a process can address (heap, stack and code)
- ➜ Open files

A process comprises:

- ➜ Memory (instructions, data section)
- ➜ Registers (program counter, stack pointer)

The separation of policy (which decision?) and mechanism (how to implement it?) brings

modularity, so you can easily change policies without having to rethink the mechanism

Process API

- ➜ Create, a new process
- ➜ Destroy, halt a runaway process
- ➜ Wait, for a process to stop running
- ➜ Miscellaneous Control, suspend a process then resume it
- ➜ Status, get status info about a process

How are programs transformed into processes?

➜ OS must first load its code and static data into memory, into the address space of the process

➜ Early OS would do it **eagerly** (everything at once before running the program)

➜ Modern OS do it **lazily** by loading pieces of code or data as they are needed during execution

➜ Then the OS needs to allocate some memory for the programs **run-time stack**. It will specifically fill the stack with arguments for the main method. The stack is used for local variables, return addresses, function parameters. The OS may also allocate some memory for the programs **heap**. The heap (malloc and free) is used for dynamic data structures such as linked lists, hash tables...

➜ Then the OS does some initialisation, e.g. each process has three **file descriptors** (input, output and error)

➜ Then the OS jumps to the main() routine and the execution starts

What states does a process have? See **Figure 4.2**

- ➜ Running
- ➜ Ready

➜ Blocked, a process has performed some kind of operation that makes it no ready to be executed until some other event takes place

➜ Schedule and descheduled is about changing the state of a process

Process list is a key data structure to track ready process and their information. **Register context** will hold for a stopped process, the contents of its registers. The individual structure that stores information about a process is referred as a **Process Control Block PCB** (aka **process descriptor**)

See the summary, its pretty good!

Use Clock get time for next homework! Immer die man pages lesen!