Operating Systems

Tutorial 2

Fabian Klopfer

16. November 2020

Intro

- Was the exercise sheet okay?
- How long did you take?
- Please tell me if something is odd with the screen/audio!
- Something unclear?

Section 1

Exercise Sheet 1

Exercise 1 I

1. **What is the difference between kernel mode and user mode?**
   Kernel mode has full access to hardware, user mode is restricted (memory, hardware).

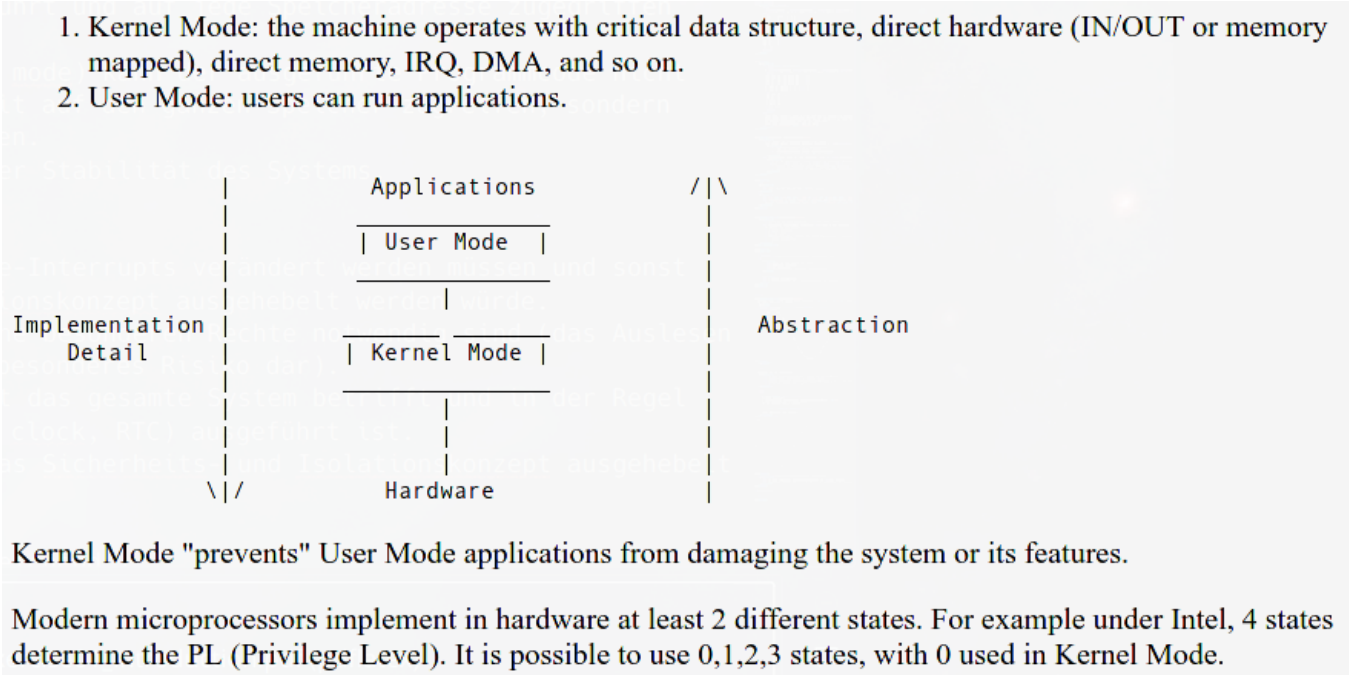

Abbildung: The Linux Kernel Documentation Project: Fundamentals [1]

Exercise 1 II

2. **Which of the following instructions should be allowed only in kernel mode, and why?**

   - Disabling all interrupts
     $\Rightarrow$ Kernel, disables interaction HW$\rightarrow$ Kernel
   - Reading the system clock
     $\Rightarrow$ User, does not interfere with Kernel or other user programs
   - Setting the system clock
     $\Rightarrow$ Kernel, does interfere with Kernel and other user programs that use the system clock
   - Changing the memory allocation tables
     $\Rightarrow$ Kernel, interferes with the memory consistency of Kernel and all other programs

Exercise 1 III

3. **Modern operating systems virtualize memory address space, that is, they use logical memory addresses instead of physical (hardware) memory addresses. Describe two advantages of this approach.**
   - Can use more memory than physically available (swapping)
   - Possible to have only necessary parts of program in-memory
   - Memory isolation: Every program has a separate logical memory space and no access to other addresses
   - Program memory doesn't need to be physically continuous (ease allocation)

Exercise 1 IV

4. When a user program reads from a disk file, it becomes blocked until the device driver has finished performing the read operation. Suppose the user program was not blocked when writing to a file, that is, execution of the program would continue while the device driver still performs the write operation.What problem could arise? Can the device driver solve it?

Exercise 1 V

Problems:

- What if another thread reads the file before the write?
  Does it get the correct version of the contents? Need a signal for the driver to schedule access
- What if the write failed?
- What if the same program alters the buffer to be written during the write process?

Exercise 2 I

What is characteristic for the use of the following types of operating systems?
Operating systems for

- personal computers
  Responsive, interactive, one user at a time
- embedded systems
  Few resources, fixed set of programs, reliability, control over device
- sensor nodes
  Few resources, often only one program to transfer/broadcast data
- real-time applications
  Hard or soft deadlines for routines, preemptive scheduling, latency of program is fixed within an interval

Exercise 2 II

What is the difference between timesharing systems and multiprogramming systems?

- Timesharing systems: several users run programs on one computing system at the same time.
- Multiprogramming systems: users run several programs simultaneously.

$\Rightarrow$ Timesharing $\subset$ Multiprogramming

Exercise 3 I

1. **Complete the table.** C.f. next slide
2. **Which quantities are typically specified with binary prefixes?**
   Data rates: Transimission speed, Read and Write Speeds.
   In general: Digital information.
3. **Why are there no binary prefixes in the lower part of the table?**
   The bit is the smallest possible chunk of information. There are no „half“ bits.
4. **How large is 1 TB in TiB?**
   $\frac{10^{12}}{2^{40}} \sim 0.9094947017729282$

Exercise 3 II

	SI		binär			
$b = 10$	Name	Symbol	Name	Symbol	$b = 2$	$b = 1024$
24	yotta	Y	yobi	Yi	80	8
21	zetta	Z	zebi	Zi	70	7
18	exa	E	exbi	Ei	60	6
15	peta	P	pebi	Pi	50	5
12	tera	T	tebi	Ti	40	4
9	giga	G	gibi	Gi	30	3
6	mega	M	mebi	Mi	20	2
3	kilo	k	kibi	Ki	10	1
-3	milli	m				
-6	micro	μ				
-9	nano	n				
	...					

Tabelle: Completed table of prefixes from ex. 3

Fetch Decode Execute I

From Patterson & Hennessy [2]:

A Fetch current instruction from memory by querying the memory with the program counter/ instruction pointer (RIP/EIP), advance it by the length of an instruction (32/64 bit)

B Decode fetched instruction into micro-code and fetch stored registers from memory if necessary. Compute possible branch targets

C Execute the micro-code instruction.
One of: Memory reference - add base and offset, register-register ALU instr. - compute what is spec'ed by op with values from registers, register-immediate ALU instr. - instr. defined second operand like increment or determine whether conditional branch condition is true.

D Memory access: Write result to register or main memory

Fetch Decode Execute II

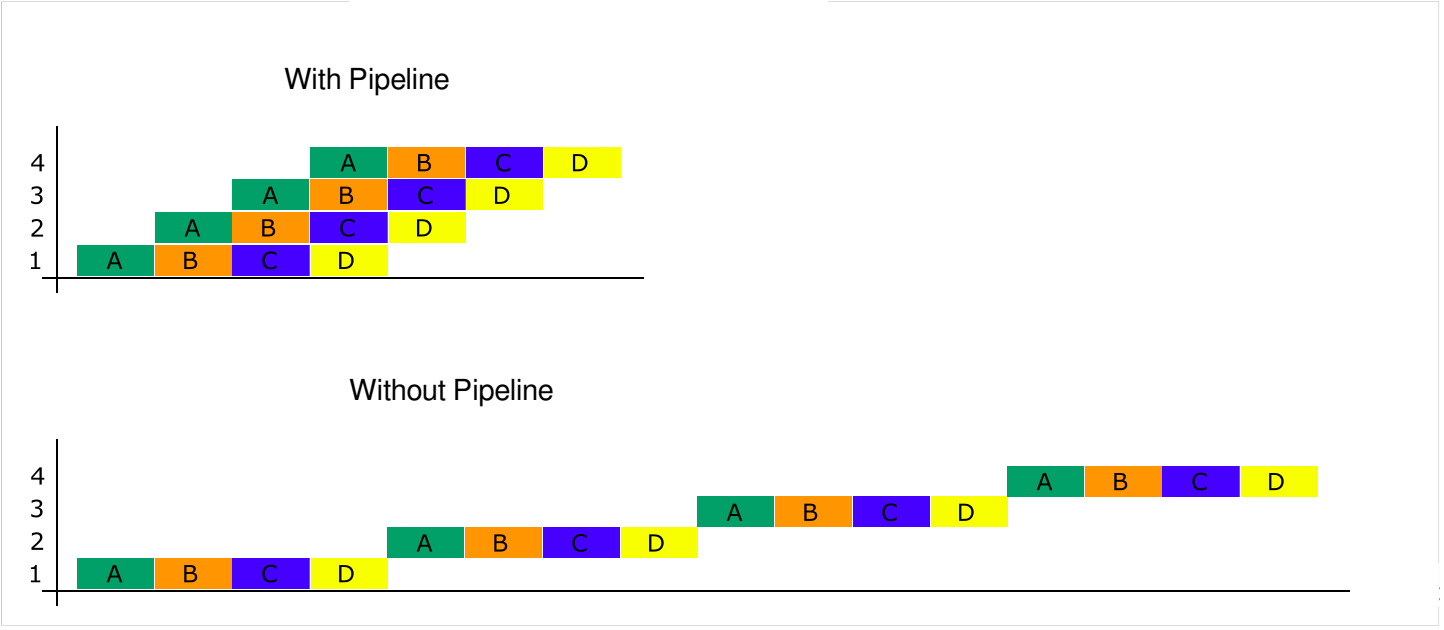

Abbildung: Illustration of instruction pipelining [3]

Fetch Decode Execute III

For unpipelined execution engines we have, with n instructions, k steps (here $k = 3$ for fetch decode and execute) and τ the time one instruction takes:

$$n \cdot k \cdot \tau$$

For pipelined execution engines we get, with i the number of steps before execution (here $i = 2$ for fetch and decode):

$$n \cdot \tau - i$$

Exercise 4 I

1. A CPU has a three-level fetch-decode-execute pipeline. Each stage of the pipeline takes the same time of 1 ns to process.

   1.1 **How many instructions can the CPU process per second?**
   $\frac{1 \text{ instr.}}{1 \text{ ns}} = \frac{1 \text{ instr.}}{1 \text{ s} \cdot 10^{-9}} = 10^9 \cdot 1 \frac{\text{instr.}}{\text{s}}$
   1 billion instructions per second.

   1.2 **How many instructions can the CPU process if the pipelines has five levels?**
   Pipelining let's the elements of the FDE cycle operate asynchronous. The same amount, as in pipelines there is only one ALU/element that can execute the operations.

Exercise 4 II

2. A computer system has a cache (access time 2 ns), RAM (access time 10 ns) and an external storage device (access time 10 ms).

   2.1 **If the cache hit rate is 95 % and the main memory hit rate is 99 %, how long is the average access time?**
   $0.95 \cdot 2\,\text{ns} + 0.05 \cdot (0.99 \cdot 10\,\text{ns} + 0.01 \cdot 10\,\text{ms}) = 5002.39\,\text{ns} \sim 5\,\mu\text{s}$

   2.2 **Which of the three access times dominates the average access time?**
   External memory as it is slower by orders of magnitude.

   2.3 **Another external memory device is connected. The measured average access time is now 2002.395 ns. What is the access time of the new external storage device?**
   $2002.395\,\text{ns} = 0.95 \cdot 2\,\text{ns} + 0.05 \cdot (0.99 \cdot 10\,\text{ns} + 0.01 \cdot x\text{ms})$
   Solving for x yields $x = 3\,996\,000\,\text{ns} \sim 4\,\text{ms}$

Exercise 5 I

1. **Write a `C` program that outputs “`Hello world!`”.**

```c
#include <stdio.h>

int main()
{
        printf("Hello world!\n");
        return 0;
}
```

Exercise 5 II

2. **Compile your program using the flags `-std=c11 -g -Wall -Wpedantic`. What does each of these flags mean?**
   - -std=c11 the 2011 ISO C standard
   - -g debugging information
   - -Wall turn on lots of error checking
   - -Wpedantic strict ISO C conformity

Exercise 5 III

3. **Which additional files are generated when you use the `-save-temps` flag and what do they contain?**
   - .i generated by the pre-processor, source with expanded pre-processor directives
   - .s generated assembly code by the compiler
   - .o generated machine code by the assembler
   - .out generated executable by the linker

Exercise 6 I

Write a C program that outputs the sine function $\sin(x)$ for $x \in [0, 2\pi]$ on the console.

Exercise 6 II

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
        // screen resolution
        const int WIDTH = 80;
        const int HEIGHT = 25;

        // scan resolution and rendering
        const double WIDTH_STEP = 2.0 * M_PI / ((double) WIDTH);
        const double HEIGHT_STEP = 2.0 / ((double) HEIGHT);
        const double THICKNESS = 1.2;

        // loop line by line, from left to right
        for (double y = 1.0; y >= -1.0; y -= HEIGHT_STEP)
        {
                for (double x = 0.0; x <= 2.0 * M_PI; x += WIDTH_STEP)
                {
                        double result = sin(x);
                        if (fabs(y -  result) < THICKNESS * HEIGHT_STEP) {
                                printf("*");
                        } else {
                                printf(" ");
                        }
                }
                 printf("\n"); // new line
        }

        // success
        return 0;
}
```

Exercise 6 III

```c
#include <stdio.h>
#include <math.h>
#include <string.h>

#define HEIGHT 16
#define WIDTH 64

int main () {
    const float horizontal_step = 2 * M_PI / (WIDTH);
    char grid[HEIGHT][WIDTH];

    float x = 0;
    int y;

    memset(&grid, 0x20, sizeof(grid));

    for (unsigned int i = 0; i < WIDTH; i++) {
        y = (int) HEIGHT/2 * sin(x) + HEIGHT/2;
        grid[y][i] = '*';
        x += horizontal_step;
    }

    for (int i = HEIGHT - 1; i > -1; i--) {
        printf("%.*s\n", (int) WIDTH, grid[i]);
    }
    return 0;
}
```

Section 2

Exercise sheet 2

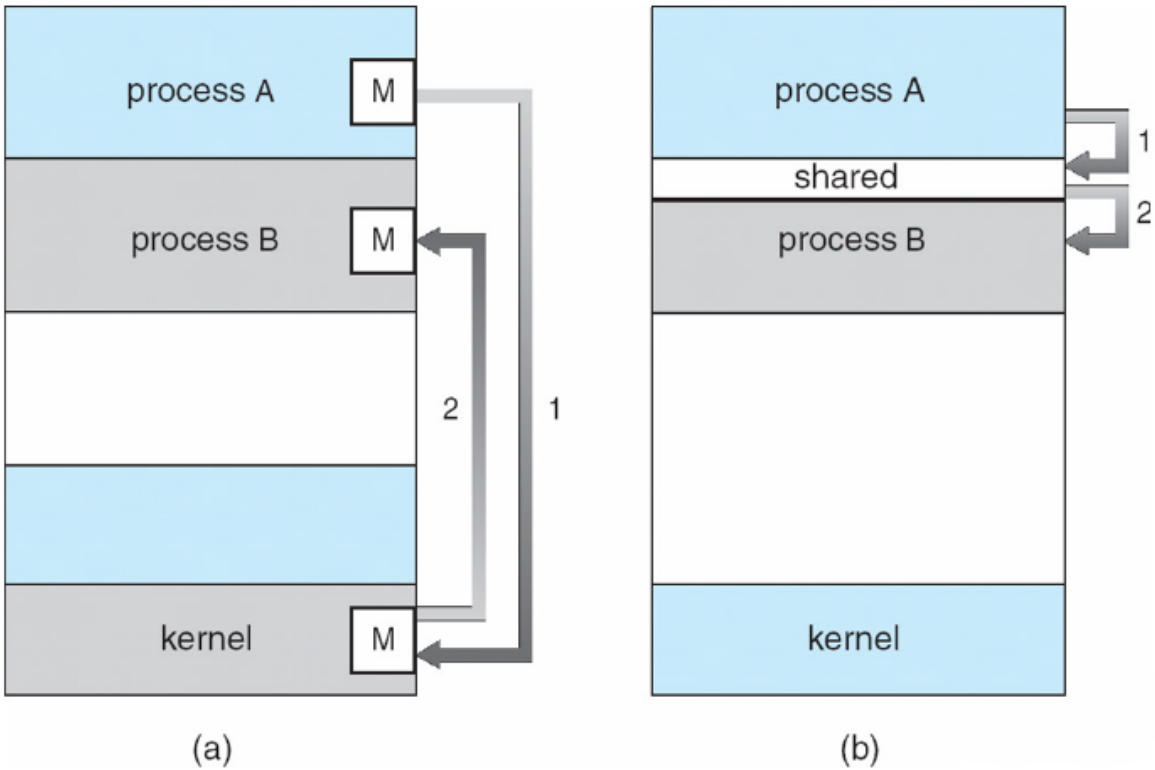

Abbildung: Visualization of how the kernel logically maps the main memory [4]

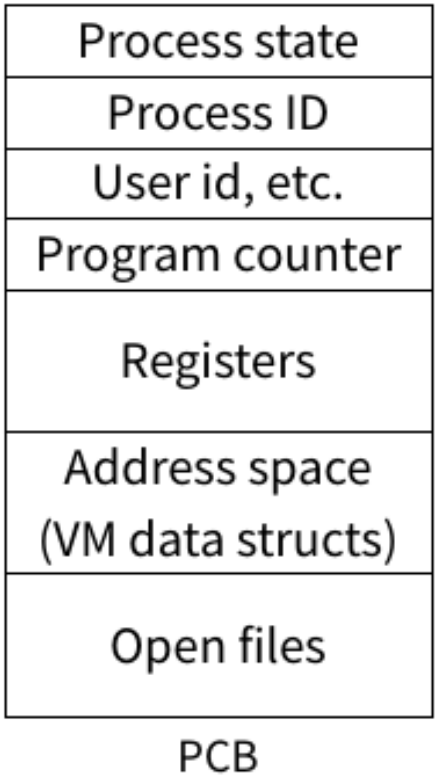

Abbildung: Elements contained in a process control block [5]

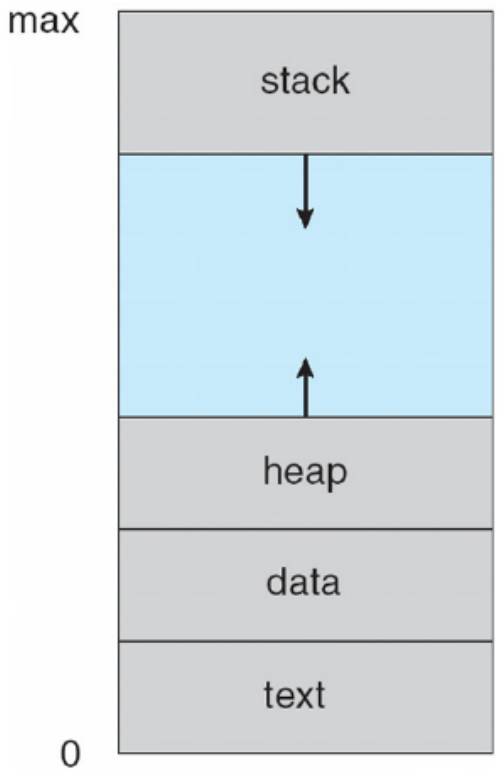

Abbildung: Elements contained in a process control block [4]

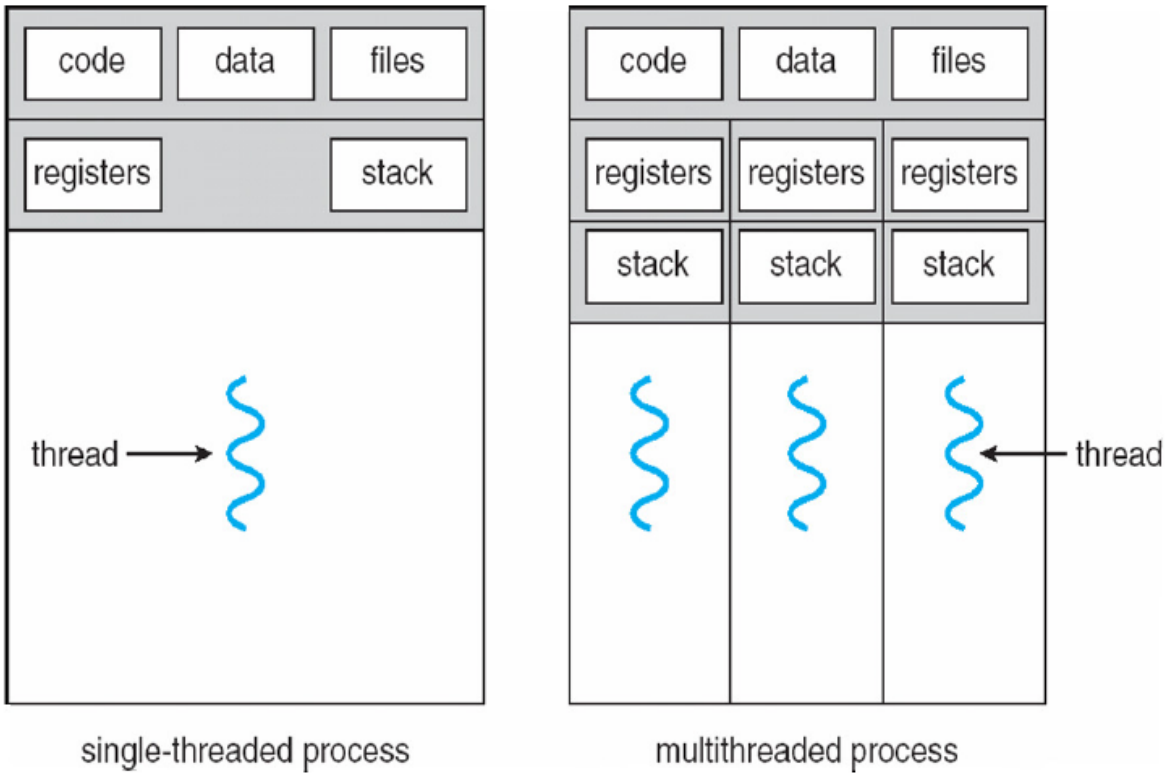

Abbildung: Elements contained in a process control block [4]

Section 3

C

Memory is uninitialized!

- https://wiki.sei.cmu.edu/confluence/display/c/EXP33-C.+Do+not+read+uninitialized+memory
- C Arrays

References I


(27. März 2003). „KernelAnalysis-HOWTO: Fundamentals“, Adresse: https://tldp.org/HOWTO/KernelAnalysis-HOWTO-3.html (besucht am 15. 11. 2020).


J. L. Hennessy und D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.


(1. Nov. 2020). „Pipeline (Prozessor) Wikipedia“, Adresse: https://de.wikipedia.org/wiki/Pipeline_(Prozessor) (besucht am 15. 11. 2020).


A. Silberschatz, P. B. Galvin und G. Gagne, *Operating system principles*. John Wiley & Sons, 2006.


A. S. Tanenbaum und H. Bos, *Modern operating systems*. Pearson, 2015.


W. Stallings, *Operating systems: internals and design principles*. Upper Saddle River, NJ: Pearson/Prentice Hall, 2009.


(15. Nov. 2020). „Microsoft's Software Is Malware - GNU Project - Free Software Foundation“, Adresse: https://www.gnu.org/proprietary/malware-microsoft.en.html (besucht am 15. 11. 2020).


(15. Nov. 2020). „Apple's Operating Systems are Malware - GNU Project - Free Software Foundation“, Adresse: https://www.gnu.org/proprietary/malware-apple.html#content (besucht am 15. 11. 2020).