Operating Systems

Tutorial 2

Fabian Klopfer

23. November 2020

Intro

- Pingo Polls
- Has everybody managed to see the annotations in the correction?

Exercise sheet 1 I

- **If you take initialization into account**, the exact answer for exercise 4.1 is $1 \cdot 10^9 - 2$ Op
  - Why -2? Before the first instruction can be executed you need to fetch one (1ns) and decode one (1ns).
  - Why not -5? Theoretically it suffices if one pipeline level is ready. One does not need to wait for all pipelines to fetch and decode.
  - Practically this is of course not the case: For example if branches are predicted incorrectly the whole pipeline is flushed. Also on arrival of an interrupt and after it's handling the pipeline is flushed. Faults and traps make the CPU push the pipeline to the stack $\rightarrow$ you also lose some cycles for pushing & reloading the pipeline.
- actually asked for was the **throughput**! This is $1 \cdot 10^9$ Op/s no matter the initial latency!

Exercise sheet 1 II

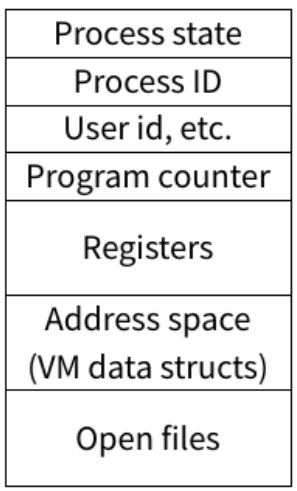
- **ALWAYS** check your code with following flags. Notice that thread can not be used with address and undefined. More information is available here.
  - Always do one pass with `-fsanitize=address -fsanitize=undefined` to check e.g. array bounds, memory leaks and other issues connected to memory.
  - For code that shall be thread safe use `-fsanitize=thread`.
  - There are even more utilities that check things at runtime for you, check the doumentation.

Section 1

Exercise Sheet 1

Exercise 1 I

1. What is the difference between program code and process?
   Process: Running program; code executing dynamically in a context.
2. What is a process control block and which information does it contain?
   Data structure containing context of a process Used by OS for maintenance and scheduling



PCB

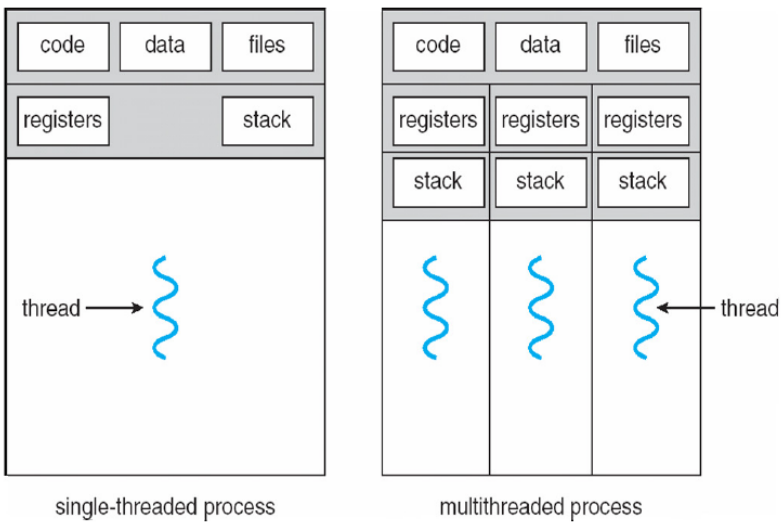
Exercise 1 II

3. What is the difference between a mode switch and a process switch?
   A process switch (context switch) stores the state of a process and loads the state of another process.
   A mode switch changes switches between user mode and kernel mode, due to an interrupt, trap or exception.

4. What is the key difference between processes and threads?
   Processes have a full context.
   Threads share the context of their process.

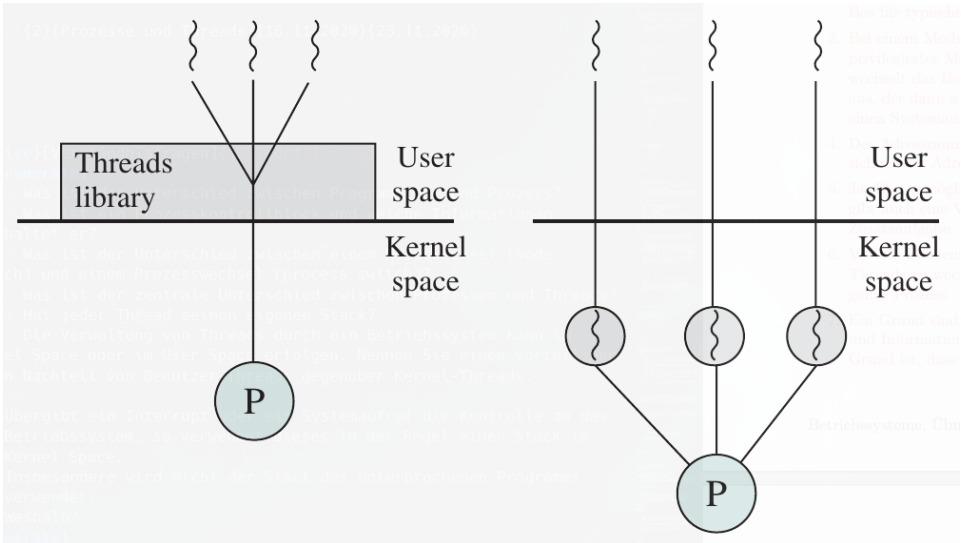
Exercise 1 III

5. Does each thread have its own stack?
   Each thread has an own stack and register values.



Exercise 1 IV

6. The management of threads by an operating system can happen in kernel space or in user space. Name an advantage and a disadvantage of user threads compared to kernel threads.
   + No mode switch for a thread switch, platform independent
   - Error prone, if a thread blocks the whole process does



Exercise 1 V

7. If an interrupt or a system call passes control to the operating system, the latter usually uses a stack in kernel space. In particular, the stack of the interrupted program is not used. Why?

   - Stack might be too small when user program uses a lot
   - Stack might be read out after returning to the user program
     sub esp X

Exercise 2 I

1. Five programs are running on a computer system. At any given time, each of the programs waits for input/output independently of the others with a probability of 1/2. What is the average fraction of CPU time that is wasted?
   Tanenbaum [1] p. 96: $Pr(\text{All Processes do IO}) = p^n$ with p the average probability to do IO and n the number of processes.

$$\Rightarrow \frac{1}{2}^5 = \frac{1}{32}$$

Exercise 2 II

2. A computer system has a memory of 16 GiB, of which the operating system occupies 768 MiB. For simplicity, we assume that all processes use 256 MiB of memory and behave the same way. CPU utilization should be 99 %. What is the maximum amount of time (in percent) each process can use for input/output?

$\frac{16\text{GiB}}{256\text{MiB}} = 64$ blocks.
OS uses 3 blocks $\Rightarrow$ 61 blocks used for 61 processes.
Tanenbaum [1] p. 96: CPU utilization $= 1 - p^n$

$$0.99 = 1 - p^{61} \Leftrightarrow 0.01 = p^{61} \Leftrightarrow p = 0.01^{\frac{1}{61}} = 0.92728 \sim 93\%$$

Exercise 3 I

Consider the following C program:

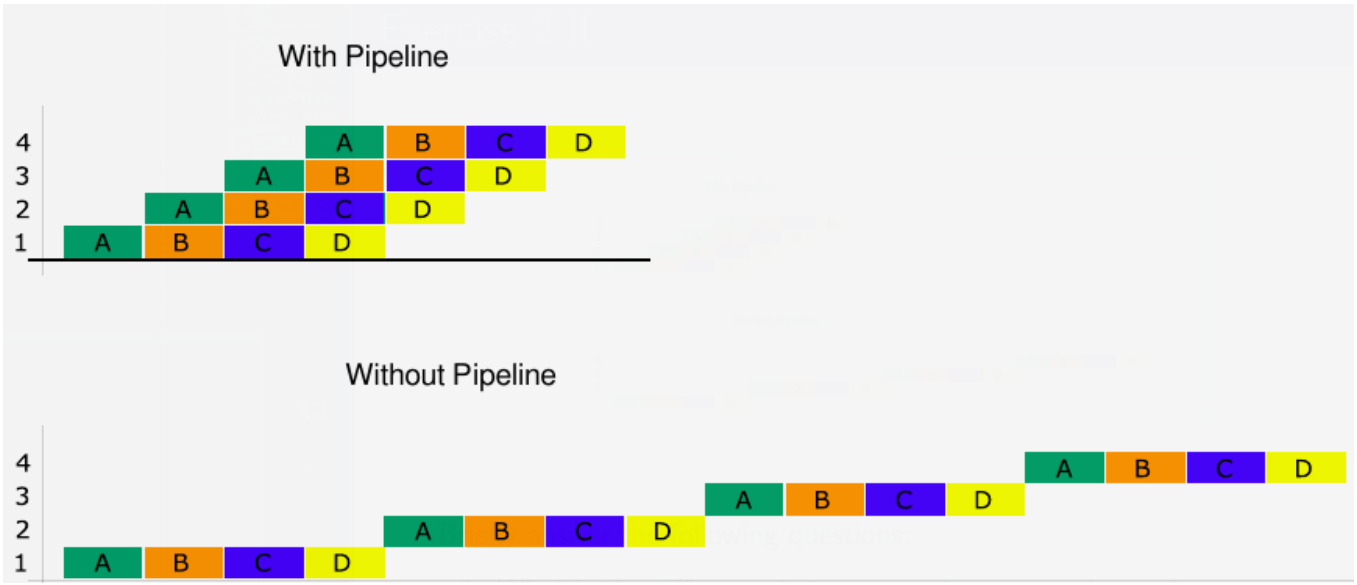
```c
#include <sys/types.h>
#include <unistd.h>

int main()
{
    fork();
    fork();

    return 0;
}
```

How many **child** processes are created when the program is executed?
Describe briefly what happens.
Pingo poll & execute it

Exercise 4 I

Define the terms “synchronous” and “asynchronous”

- Synchronous: Coordinated with others (e.g. threads, processes, devices, nodes), e.g. actively wait for completion of previous step
- Asynchronous: The timing is detached from others and signalled instead



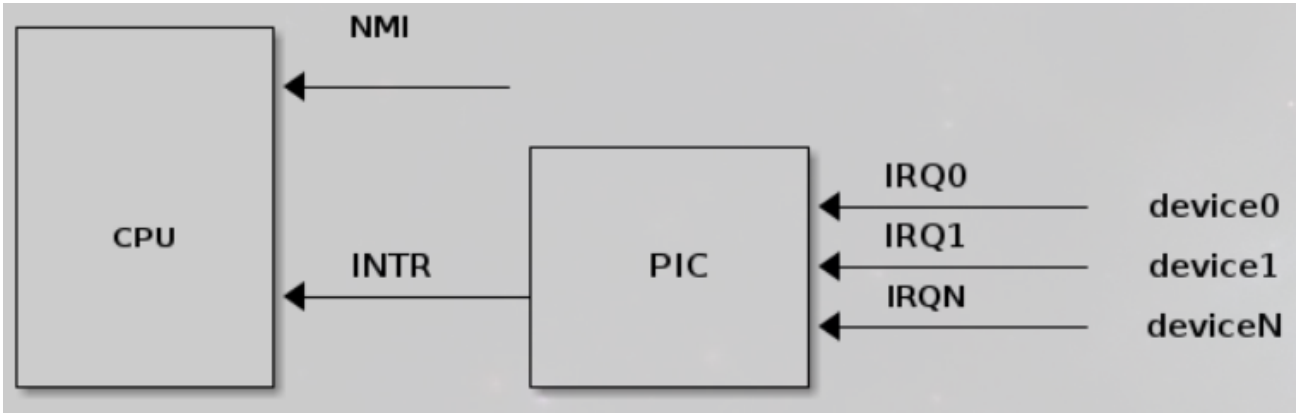
Exercise 4 II

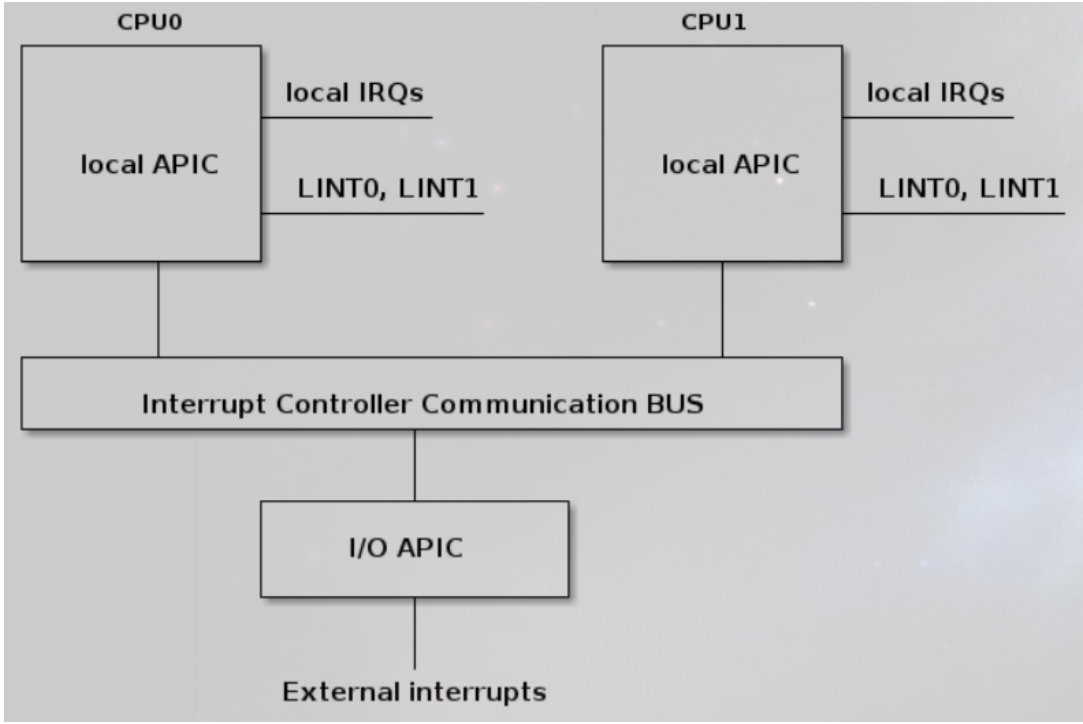
Briefly answer the following questions:

1. What is the difference between synchronous and asynchronous interrupts?
   - Synchronous Interrupts: Issued by the CPU itself; exception, traps, faults e.g. page fault, syscalls
   - Asynchronous Interrupts: Issued by other hardware components; e.g. when network card receives a package, when DMA finished
2. What is an Advanced Programmable Interrupt Controller (APIC)?

   Controller in between CPU and other HW
   Receives interrupts, converts to vector number, writes this to register & raise on INTR or NMI pin, masks interrupts, distributes to local APICs

Exercise 4 III





Exercise 4 IV

3. What is the Interrupt Descriptor Table?
   Maps each interrupt to an interrupt handler('s address)
4. What is an Interrupt Handler?
   A program that is called if a specific interrupt arrives.
   E.g. device driver, entry point for syscalls, page fault handling, ...
5. Which information is contained in the Extended Flags (EFFLAGS), Code Segment (CS) and EIP (Extended Instruction Pointer) registers?
   - EFLAGS: Status & system flags, like carry, parity, trap, interrupts en-/disabled, privileges, overflow, ...
   - CS: Code Segment pointer; points to start of current process' code
   - EIP: offset from CS to current instruction

Exercise 4 V

6. How does one register an interrupt handler?

```
request_irq(
    1,          /* The number of the keyboard IRQ on PCs */
    irq_handler, /* our handler */
    SA_SHIRQ,    /* We share the IRQ */
    "test_keyboard_irq_handler", /* name of our handler */
    (void *)(irq_handler)); /* id of the handler */
/* (used for removing specific handler in case of shared IRQs) */
```

Exercise 4 VI

7. What is meant by “upper half” and “bottom half”? What is the reason for the “bottom half”?

   - Upper half: Interrupts disabled. Acknowledge the IRQ, execute the handler, call end of interrupt
   - Bottom half: Interrupts enabled. Handler usually queues work. Execution of this work is bottom half.

Exercise 5 I

Consider the following two-dimensional array:

```
#define N 5
        #define M 6

        const int array[N][M] =
        {
            { 1,  2,  3,  4,  5,  6},
            { 7,  8,  9, 10, 11, 12},
            {13, 14, 15, 16, 17, 18},
            {19, 20, 21, 22, 23, 24},
            {25, 26, 27, 28, 29, 30}
        };
```

Write a C program with the functions described below. The output of the program should consist of three lines.

Exercise 5 II

1. Write a C function that receives the array as parameter and outputs it line by line:
   `1 2 3 4 5 6 7 8 9 $\ldots $ 24 25 26 27 28 29 30`
   Use a double nested `for` loop for this.

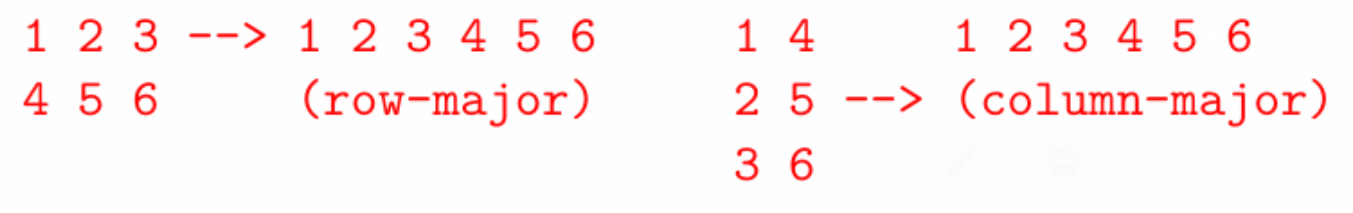
2. Write a C function that gets the array as parameter and outputs it column by column:
   `1 7 13 19 25 2 8 $\ldots $ 29 6 12 18 24 30`
   Use a double nested `for` loop for this.

3. Write a C function that receives the array as parameter and outputs it line by line.
   Use a single, non-nested `for` loop for this.

Show code

Exercise 5 III

4. Let $A \in \mathbb{R}^{N \times M}$ be a matrix and let A^T denote the transposed matrix. Is the row-major memory layout of A the same as the column-major memory layout of A^T? Rationalize your answer, e.g. with a diagram.



Exercise 6 I

Inform yourself about stackless threads. A possible starting point is

1. What advantage do stackless threads offer over conventional threads?
   Need less memory.
2. What is the disadvantage of stackless threads compared to conventional threads?
   local variables are not preserved between thread switches $\Rightarrow$ need to use static or global vars
3. Do stackless threads use pre-emptive or cooperative scheduling?
   Cooperative

Section 2

Exercise sheet 3

Exercise 1 (Comprehension questions) $(4 \cdot 1 = 4$ Points)

1. What is the difference between cooperative and preemptive scheduling?
2. Name one disadvantage of cooperative scheduling.
3. Which information is required for Shortest-Job-First-Scheduling?
4. If the costs for a context switch are high, should one choose a small or a large time quantum (e.g., for round-robin scheduling)?

- `yield` vs. forced switch
- What if not yielded?
- How to know what's the shortest?
- low quantum means many switches

Exercise 2 (Round-Robin-Scheduling) $(2 + 2 = 4$ Points)

1. Round-Robin schedulers often employ a list of all running processes. Each process appears exactly once in such a list. What could be the use of having a process appear multiple times in the list? Which problem could occur?

2. *Priority inversion:* There are two processes H (high priority) and L (low priority) running on a computer. With priority scheduling, H is always run when H is ready to compute. Now the following happens: L reserves a resource (e.g., exclusive access to a file), H becomes ready for calculation (e.g., after completion of an I/O operation) and tries to access the same resource as L. Since this resource is already exclusively occupied by L, H (actively) waits for L to release the resource. However, L never receives computing time while H is running and cannot release the resource. The result is an endless loop.

   Can this problem also occur if Round-Robin scheduling is used instead of priority scheduling?

- Multiple occurrences means multiple time quanta per round. What if it blocks?
- Round Robin is preemptive no matter the priority

Exercise 3 (CPU efficiency) $(4 \cdot 1 = 4$ Points)

Assume that processes run (on average) for a time T before they block because of I/O, and that a context switch needs time S. For Round-Robin scheduling with time quantum Q, specify a formula for CPU efficiency in the following cases. CPU efficiency is process computing time as a fraction of total computing time.

1. $Q > T$
2. $S < Q < T$
3. $Q = S$
4. $Q \to 0$

- Give a per process formula!
- Runtime = CPU time + IO time
- Consider how many context switches are necessary to finish the process
- For $Q = S$ use the previous result & substitute
- What does the OS do often if the quantum is very close to zero?

Exercise 4 (Process turnaround times) $(4 \cdot 2 = 8$ Points)

The five batch processing jobs $A, \ldots, E$ arrive together at a data center. Their estimated run times and priorities are known (table). Determine the average turnaround time for each of the following scheduling algorithms. Neglect the time for process switches and assume that no process is waiting for input/output.

Process	Runtime / time units	Priority (high=important)
A	10	3
B	6	5
C	2	2
D	4	1
E	8	4

1. Round-Robin Scheduling
2. Priority Scheduling
3. First-Come-First-Serve Scheduling. The arrival order of the processes is A, B, C, D, E.
4. Shortest-Job-First scheduling.

Draw a diagram similar to the one on the left side.
Popular exam exercise!

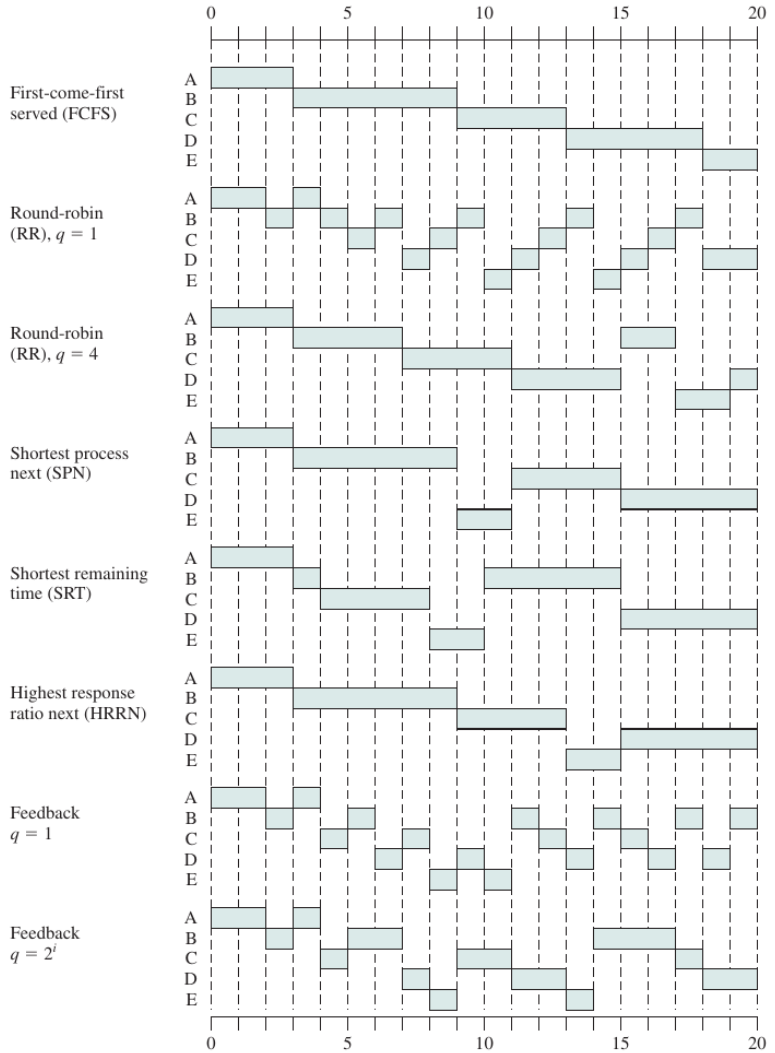


Figure 9.5 **A Comparison of Scheduling Policies**

Programming exercise 5 (Variable-length arrays) (1 + 1 = 2 Points)

Consider the following C program:

```
#define SIZE 1024*1024*1024

int main(void)
{
    int array [SIZE];

    return 0;
}
```

Assume that no special command line arguments were used during compilation. What happens when the program is executed? Why?

Note: The command `int * array = calloc(SIZE, sizeof(int));` reserves heap memory. This is not what happens in the above program.

Use the sanitizers! Both address and undefined!

Programming exercise 6 (Self study: command line arguments) (6 Points)

Learn how a `C` program accepts command line arguments, for example at cppreference.com. Write a program that accepts exactly one string as input and outputs the number of vowels (a, e, i, o, u, A, E, I, O, U) in it. If no or more than one command line argument is given, an error message should be displayed.

Example for an error message:

```
./a.out
Please specify a single string as the only command line argument.
```

Example of a successful call:

```
> ./a.out "Hello world!"
> 3
```

Check the C programming slides or the C reference.

Bonus exercise 7 (Shortest-Job-First Scheduling) $(4 + 2 = 6$ Extra points)

1. Prove that Shortest-Job-First Scheduling always leads to the shortest average turnaround time, and is therefore optimal in that sense.
2. This does not apply for different arrival times. Give a counter example.

Check the Tanenbaum book p. 208/209.
Adapt the proof in the book to be more general.

References I


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.


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


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


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