Operating Systems

Tutorial 2

Fabian Klopfer

30. November 2020

Intro

- Again: Sheet 1 Ex. 4 (FDE Pipeline):
  If in the exam, will be asked for the throughput specifically.
- Pingo Polls

Exercise Sheet 3

Exercise 1

Exercise 1 I

1. What is the difference between cooperative and preemptive scheduling?
   Cooperative: Scheduler doesn't actively interrupt process; process **yields**.
   Preemptive: Scheduler queues timer interrupt to pause execution and schedule sth. else.
2. Name one disadvantage of cooperative scheduling.
   If CPU not released by programmer $\Rightarrow$ System deadlocked
3. Which information is required for Shortest-Job-First-Scheduling?
   (Expected) runtime of jobs to schedule.
4. If the costs for a context switch are high, should one choose a small or a large time quantum?
   Large to minimize overhead.

Exercise 2

Exercise 2 I

1. Round Robin:
   What could be the use of having a process appear multiple times in the list?
   Which problem could occur?
   Prioritization
   If prioritized process blocks, all entries need removal, else blocks CPU.

Exercise 2 II

2. *H* (high priority), *L* (low priority) processes.
   L reserves a resource, *H* becomes ready & tries to access same resource.
   H (actively) waits for *L* to release the resource, *L* never receives computing time while *H* is running and cannot release resource.
   $\Rightarrow$ Deadlock.
   Can this problem also occur if Round-Robin scheduling is used instead of priority scheduling?
   No. *L* will still get CPU time & release resource s.t. *H* can execute.

Exercise 3

Exercise 3 I

Avg. process CPU time before IO T, context switch takes S, Round-Robin scheduling with time quantum Q.
Specify formula for CPU efficiency in the following cases.

1. $Q > T$

$$\frac{T}{(S+T)}$$

Process runs until it's doing IO, context switches only happen on IO.

2. $S < Q < T$

$$\frac{T}{T + S \cdot T/Q}$$

Requires $\frac{T}{Q} - 1$ context switches due to scheduling $+1$ for IO

Exercise 3 II

3. $Q = S$

$$\frac{T}{T + S \cdot T/Q} \Rightarrow \frac{T}{T + Q \cdot T/Q} = \frac{1}{2}$$

4. $Q \to 0$

$$\lim_{Q \to 0} \text{Context switches share} \to 100\%$$

$$\Rightarrow \lim_{Q \to 0} \text{CPU efficiency} \to 0\%$$

Exercise 4

Exercise 4 I

Jobs $A, \dots, E$.
Determine the average turnaround time for:

1. Round-Robin
2. Priority
3. First-Come-First-Serve.
   Order: A, B, C, D, E.
4. Shortest-Job-First.

Process	Runtime	Priority
A	10	3
B	6	5
C	2	2
D	4	1
E	8	4

Exercise 4 II

1. RR: Official Solution (Tanenbaum Ch. 2 Ex. 45 [1])
   Based on rounds $\Rightarrow$ no order assumption

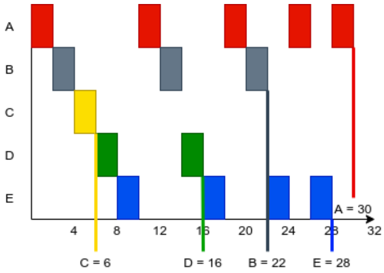
Process	Turnaround
A	30
B	24
C	10
D	18
E	28

$$\frac{1}{5} \cdot (30 + 24 + 10 + 18 + 28)$$

$\Rightarrow$ 22 min

Exercise 4 III

Using execution: Assuming the same order as FCFS.



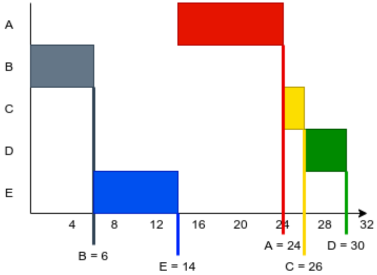
Process	Turnaround
A	30
B	22
C	6
D	16
E	28

$$\frac{102}{5} = 20.4$$

20 min 24 s

Exercise 4 IV

2. Priority: $B = 5, E = 4, A = 3, C = 2, D = 1$

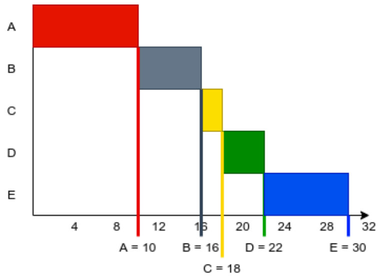


Process	Turnaround
A	24
B	6
C	26
D	30
E	14

20 min

Exercise 4 V

3. FCFS

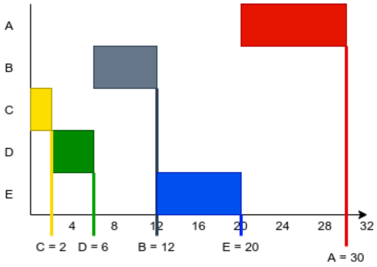


Process	Turnaround
A	10
B	16
C	18
D	22
E	30

$19.2 = 19$ min 12 s

Exercise 4 VI

4. SJF



Process	Turnaround
A	30
B	12
C	2
D	6
E	20

14 min

Exercise 5

Exercise 5 I

Consider the following C program:

```c
#define SIZE 1024*1024*1024 // 1GiB

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

    return 0;
}
```

What happens when the program is executed? Why?

Segmentation Fault: Stack Overflow.
Default stack size in Linux: `ulimit -s` = 8192 KB.[1]

[1]Interesting read: SO: Pages thread stacks

Exercise 6

Exercise 6 I

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.

```c
#include <stdio.h>
#include <string.h>
#define MAX_SIZE 100

int main(int argc, char* argv[]) {
    if (argc != 2) {
        printf("Please specify only one string to count the vowels of.");
        return -1;
    }

    unsigned int vowel = 0;
    size_t len = strlen(argv[1]);

    for(size_t i = 0; i < len; i++) {
        switch(argv[1][i]) {
            case 'a':
            case 'e':
            case 'i':
            case 'o':
            case 'u':
            case 'A':
            case 'E':
            case 'I':
            case 'O':
            case 'U':
                vowel++;
                break;
            default:
                break;
        }
    }

    printf("Total number of vowels = %i\n", vowel);

    return 0;
}
```

Exercise 7

Exercise 7 I

1. Prove that Shortest-Job-First Scheduling always leads to the shortest average turnaround time, and is therefore optimal in that sense.

Exercise 7 II

Lemma 1 [2]: Given a set of jobs, moving a short job (S) before a long job (L) decreases the waiting time of the short job more than it increases that waiting time of the long job. Consequently the average turnaround time — as well as the average waiting time – decrease.

Exercise 7 III

Proof: Let L, S as above with run times l and s respectively. $S1$ denotes the schedule before switching, while $S2$ denotes the schedule afterwards. The waiting time of S with respect to schedule $S1$ is $WTS(S1)$. Define the others similarly. The total waiting time of $S,\ L$ with respect to a schedule is $WT_{Total}(S1) = WTS(S1) + WTL(S1)$, similarly for $S2$.
Let $WTS(S2) = WTS(S1) - l$ the waiting time of S without having to wait for L, i.e. when switching, while $WTL(S2) = WTL(S1) + s$ is the waiting time for L in this case.

Exercise 7 IV

Using the above we get:

$$WT_{Total}(S1) > WT_{Total}(S2)$$

$$WTS(S1) + WTL(S1) > WTS(S2) + WTL(S2)$$

$$WTS(S1) + WTL(S1) > WTS(S1) - l + WTL(S1) + s$$

$$0 > -l + s$$

Since $l > s$, indeed $0 > -l + s$, thus $WT_{Total}(S1) > WT_{Total}(S2)$

□

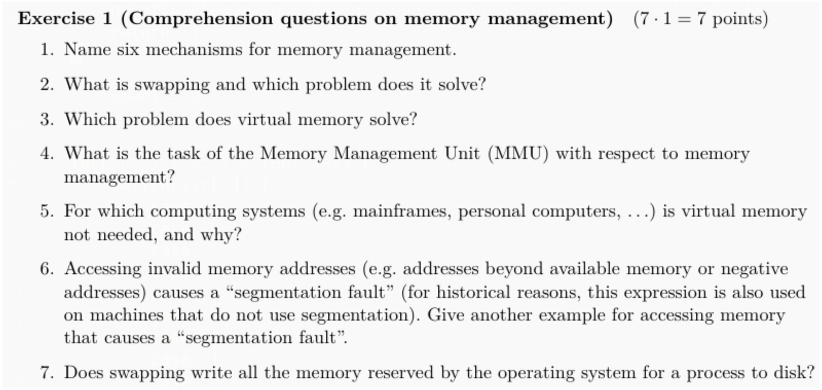
Exercise 7 V

Applying the above Lemma repeatedly results in an ordering by runtime. With finitely many jobs and thus finitely many exchanges, this ordering is unique up to permutation of jobs with the same runtime and gives a shortest job first schedule. This schedule is minimal as there is no change that reduces the runtime further.

Exercise 7 VI

2. This does not apply for different arrival times. Give a counter example.
   Five processes A, B, C, D, E with runtimes $2, 4, 1, 1, 1$ and arrival times $0, 0, 3, 3, 3$.
   Initially only A or B can be selected.
   SJF gives $A, B, C/D/E$ with an average turnaround time of
   $$\frac{1}{5}(2+6+4+5+6) = 4.6$$
   $C/D/E, A, B$ has the smaller average cycle time
   $$\frac{1}{5}(1+2+3+5+9) = 4$$

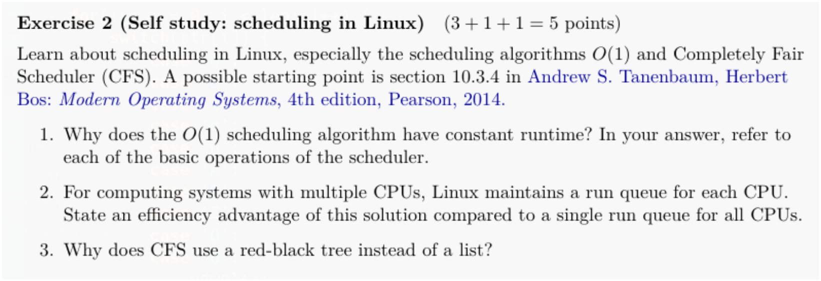
Exercise sheet 4



Exercise 1 (Comprehension questions on memory management) (7 · 1 = 7 points)

1. Name six mechanisms for memory management.
2. What is swapping and which problem does it solve?
3. Which problem does virtual memory solve?
4. What is the task of the Memory Management Unit (MMU) with respect to memory management?
5. For which computing systems (e.g. mainframes, personal computers, ...) is virtual memory not needed, and why?
6. Accessing invalid memory addresses (e.g. addresses beyond available memory or negative addresses) causes a "segmentation fault" (for historical reasons, this expression is also used on machines that do not use segmentation). Give another example for accessing memory that causes a "segmentation fault".
7. Does swapping write all the memory reserved by the operating system for a process to disk?

- Use the bold-printed keywords from chapter 3 [1], P. 187, P. 194, P. 196, P. 185 (also think of washing machines), P. 205, what is paging good for

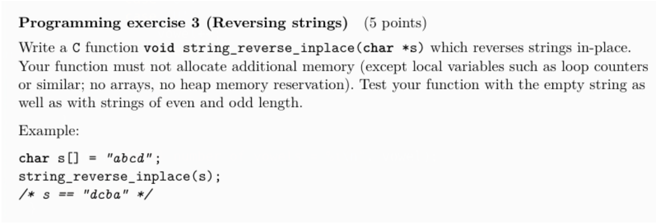


Exercise 2 (Self study: scheduling in Linux) $(3 + 1 + 1 = 5$ points)

Learn about scheduling in Linux, especially the scheduling algorithms $O(1)$ and Completely Fair Scheduler (CFS). A possible starting point is section 10.3.4 in Andrew S. Tanenbaum, Herbert Bos: *Modern Operating Systems*, 4th edition, Pearson, 2014.

1. Why does the $O(1)$ scheduling algorithm have constant runtime? In your answer, refer to each of the basic operations of the scheduler.
2. For computing systems with multiple CPUs, Linux maintains a run queue for each CPU. State an efficiency advantage of this solution compared to a single run queue for all CPUs.
3. Why does CFS use a red-black tree instead of a list?

- Figure 10.10 in [1]
- 2.6 and ongoing[2]
- TLDP - CFS Scheduler[3]



Programming exercise 3 (Reversing strings) (5 points)

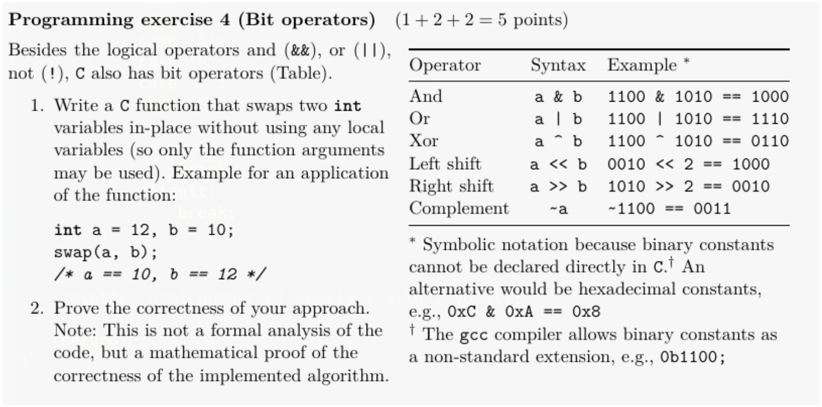
Write a C function `void string_reverse_inplace(char *s)` which reverses strings in-place. Your function must not allocate additional memory (except local variables such as loop counters or similar; no arrays, no heap memory reservation). Test your function with the empty string as well as with strings of even and odd length.

Example:

```
char s[] = "abcd";
string_reverse_inplace(s);
/* s == "dcba" */
```

- Always swap two chars
- start at beginning and at end with exchange (1. Iter: change chars 0 & $n-1$, ...)
- mind the `'\0'` null terminator of strings!
- Fix all warnings!

```
-g -fsanitize=undefined -fsanitize=address -Wall -Werror -Wpedantic -Wextra
```



Programming exercise 4 (Bit operators) $(1 + 2 + 2 = 5$ points)

Besides the logical operators and (`&&`), or (`||`), not (`!`), `C` also has bit operators (Table).

1. Write a `C` function that swaps two `int` variables in-place without using any local variables (so only the function arguments may be used). Example for an application of the function:

```
int a = 12, b = 10;
swap(a, b);
/* a == 10, b == 12 */
```

2. Prove the correctness of your approach. Note: This is not a formal analysis of the code, but a mathematical proof of the correctness of the implemented algorithm.

Operator	Syntax	Example *
And	`a & b`	`1100 & 1010 == 1000`
Or	`a | b`	`1100 | 1010 == 1110`
Xor	`a ^ b`	`1100 ^ 1010 == 0110`
Left shift	`a << b`	`0010 << 2 == 1000`
Right shift	`a >> b`	`1010 >> 2 == 0010`
Complement	`~a`	`~1100 == 0011`

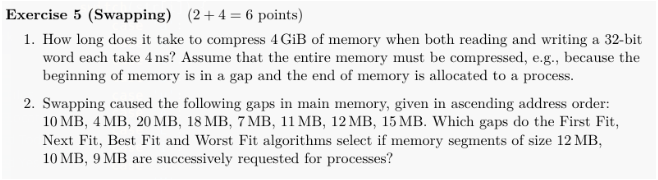
* Symbolic notation because binary constants cannot be declared directly in `C`.[†] An alternative would be hexadecimal constants, e.g., `0xC & 0xA == 0x8`

† The `gcc` compiler allows binary constants as a non-standard extension, e.g., `0b1100;`

- xor is self-inverse:

$$A \text{ XOR } B \text{ XOR } A = B$$

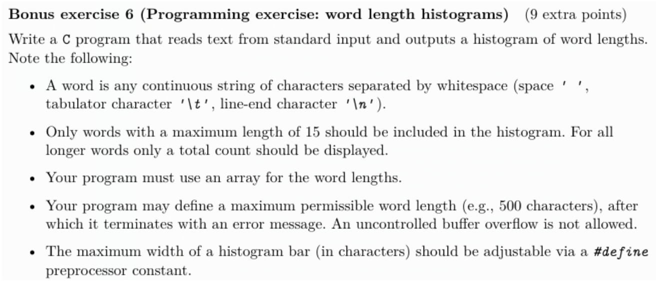
- You could start with 4.2



Exercise 5 (Swapping) (2 + 4 = 6 points)

1. How long does it take to compress 4 GiB of memory when both reading and writing a 32-bit word each take 4 ns? Assume that the entire memory must be compressed, e.g., because the beginning of memory is in a gap and the end of memory is allocated to a process.
2. Swapping caused the following gaps in main memory, given in ascending address order: 10 MB, 4 MB, 20 MB, 18 MB, 7 MB, 11 MB, 12 MB, 15 MB. Which gaps do the First Fit, Next Fit, Best Fit and Worst Fit algorithms select if memory segments of size 12 MB, 10 MB, 9 MB are successively requested for processes?

- 32-bit = 4 bytes. Every memory cell needs to be read and written
- Important, common exam exercise.
  Understand the algorithms & draw the memory per algorithm execution!



Bonus exercise 6 (Programming exercise: word length histograms) (9 extra points)

Write a `C` program that reads text from standard input and outputs a histogram of word lengths. Note the following:

- A word is any continuous string of characters separated by whitespace (space `' '`, tabulator character `'\t'`, line-end character `'\n'`).
- Only words with a maximum length of 15 should be included in the histogram. For all longer words only a total count should be displayed.
- Your program must use an array for the word lengths.
- Your program may define a maximum permissible word length (e.g., 500 characters), after which it terminates with an error message. An uncontrolled buffer overflow is not allowed.
- The maximum width of a histogram bar (in characters) should be adjustable via a `#define` preprocessor constant.

- Fix all warnings!

  ```
  -g -fsanitize=undefined -fsanitize=address -Wall -Werror -Wpedantic -Wextra
  ```

- scanf with `"%MAX_LENGTHs"`.
- scale the histogram according to the most often encountered word length and the defined max length.

[2]https://www.cs.montana.edu/ chandri-
ma.sarkar/AdvancedOS/CSCI560_Proj_main/index.html
[3]https://www.kernel.org/doc/html/latest/scheduler/sched-design-CFS.html

Syscall vs. Upcall vs. Hypercall
Monotlithic vs. Micro vs. Hypervisor

References I

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

E. T. Lee, „On average turnaround time“, *Kybernetes*, 1990.

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.