Operating Systems

Tutorial 5

Fabian Klopfer

7. Dezember 2020

Intro

- Pingo Polls

Exercise Sheet 4

Exercise 1

Exercise 1 I

1. Name six mechanisms for memory management.
   For example: Address spaces, swapping, virtual memory, paging, segmentation, compression.
2. What is swapping and which problem does it solve?
   - Using disk space to store process memory to RAM of currently not used processes.
   - Solves the problem of having not enough RAM for all processes.
3. Which problem does virtual memory solve?
   The problem of requiring more RAM than available.
4. What is the task of the Memory Management Unit (MMU) with respect to memory management?
   Translates virtual to physical addresses.

Exercise 1 II

5. For which computing systems (e.g. mainframes, personal computers, . . .) is virtual memory not needed, and why?
   Where hardware and software is known and fixed.
   Small scale systems e.g. sensor nodes, special processors (e.g. DSPs).
   Pros: Simpler MM, smaller OS.
6. Give another example for accessing memory that causes a "segmentation fault".
   Accessing another processes memory, writes to read-only regions.
7. Does swapping write all the memory reserved by the operating system for a process to disk?
   Only the actually used regions.

Exercise 2

Exercise 2 I

1. Why does the $O(1)$ scheduling algorithm have constant runtime?
   1.1 Get next process to run:
   Identify non-empty highest priority queue, i.e. find first set bit in *active* array. $\Rightarrow \mathcal{O}(1)$.

   1.2 Remove from run queue:
   Process doesn't have to be looked up (it's the active one, right?), i.e. only insert it into a queue. $\Rightarrow \mathcal{O}(1)$.

   1.3 Switch *active* with *expired*: Swap the pointers. $\Rightarrow \mathcal{O}(1)$.

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.
   - Caching
   - Less synchronization effort

Exercise 2 II

3. Why does CFS use a red-black tree instead of a list? No linked list look up ($\mathcal{O}(n)$) but tree look ups ($\mathcal{O}(\log(n))$)

Exercise 3

Exercise 3 I

Write a function `void string_reverse_inplace(char *s)` which reverses strings in-place. [. . .]

```
void string_reverse_inplace(char* str) {
    char *end = str;
    char temp;

    for (; *end != '\0'; ++end) ;
    for (; str < --end; ++str) {
        temp = *str;
        *str = *end;
        *end = temp;
    }
}
```

Exercise 4

Exercise 4 I

1. Write a function that swaps two `int` variables in-place without using any local variables.

   ```
   void swap(int a, int b) {
       a = a^b;
       b = a^b;
       a = a^b;
   }
   ```

2. Prove the correctness of your approach.

Exercise 4 II

1. Version: Let $a, \ b$ variables with binary base. Let $t = a$ XOR b. Thus t is 0 at place i if $a_i = b_i$ and 1 otherwise.

$$t \text{ XOR } a = b$$

$$t \text{ XOR } b = a$$

is to be proven. Starting with the first conjecture.

$$t \text{ XOR } a = a \text{ XOR } b \text{ XOR } a$$

Using that XOR is associative, commutative and self-inverse, we get:

$$a \text{ XOR } a \text{ XOR } b = 0 \text{ XOR } b = b$$

Similarly for the second one.

Exercise 4 III

2. Version: As Only bits-wise ops used, sufficient to show correctness for one bit (more bits follow inductively):

ab	a^=b	b^=a	a^=b
00	00	00	00
01	11	10	10
10	10	11	01
11	01	01	11

Exercise 5

Exercise 5 I

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.

$$2 \cdot \frac{4 \text{ ns}}{32 \text{ bit}} = 2 \frac{\text{ns}}{\text{bytes}}$$

$$4 \text{ GiB} \cdot 2 \frac{\text{ns}}{\text{bytes}} = 2^{32} \text{ bytes} \cdot 2 \cdot 10^{-9} \frac{\text{s}}{\text{bytes}}$$

$$2^{33} \cdot 10^{-9} \text{ s} = 8.589934592 \text{ s}$$

Exercise 5 II

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?

Algorithmus	12 MB	10 MB	9 MB
First Fit	20	10	18
Next Fit	20	18	11
Best Fit	12	10	11
Worst Fit	20	18	15

Exercise 6

Exercise 6 I

Write a program that reads text from standard input and outputs a histogram of word lengths.

Too long for slides, show in editor.

Exercise sheet 5

Exercise 1 (comprehension questions) $(10 \cdot 1 = 10$ points)

1. What problem does paging solve?
2. What is the difference between swapping and paging?
3. What is the difference between overlays and paging?
4. What is a page and what is a page frame?
5. What is the relation between the size of pages and the size of page frames?
6. Why do pages usually have a size that corresponds to a power of two?
7. What is the function of page tables?
8. What is a page fault?
9. What problem does segmentation solve?
10. Can paging and segmentation be combined?

- All in the book, mostly in the paging chapter

Exercise 2 (page faults) ($2 \cdot 2 = 4$ points)

Consider a two-dimensional array `int X[64][64]` on a computing system with four page frames, where each frame has space for 128 values of type `int`. Programs that access `X` are always located in page 0. The array `X` is stored and retrieved in the three other frames. Calculate the number of page faults for the following two code fragments:

```
// Fragment A
for (int j = 0; j < 64; ++j)
    for (int i = 0; i < 64; ++i)
        X[i][j] = 0;

// fragment B
for (int i = 0; i < 64; ++i)
    for (int j = 0; j < 64; ++j)
        X[i][j] = 0;
```

Note: `C` stores arrays in row-major order. For example `X[i][j]` and `X[i][j+1]` are stored in consecutive memory cells.

- How many iterations can be done with the contents of one page?
- Mind the iteration order: The *inner* loop decides how many items can be read from one page.

Programming exercise 3 (data types) (4 points)

Complete the table:

Expression	Description
v	The value of the variable v
&v	
*p	
a[2]	
**p	

- Check out the C Programming Language [1], chapter 5 (especially 5.3 (3 pages)).
- Remember & gives you the address of a value, * gives you the value at an address

Exercise 4 (page tables) ($4 \cdot 1 = 4$ points)

In the page table on the right, adresses are given in the decimal system. The size of the pages and frames is 1 KiB. Translate the following virtual addresses into physical addresses, or indicate that a page fault would occur:

1. 5555
2. 4100
3. 2048
4. 2047

Page	P/A Bit	R-Bit	M-Bit	Page frame
0	1	1	1	4
1	1	1	0	7
2	1	0	0	3
3	1	0	0	2
4	0	0	0	—
5	1	1	1	0

- 6 Pages, next power of 2 is $2^3 = 8 \Rightarrow$ 3 Bits for pages
- 1KiB are $2^?$
- $3 + ? =$ bits of a virtual address
- Use bit masks, logical and & shifts to extract page ID and offset
- Use bit shift and logical or to construct physical address from frame and offset

- Example: $1025_{10} = 0010000000001_2$.

  Page ID $=$ (Bit Mask & Virtual Address) $>>$ Offset length

  $$(1110000000000_2 \& 0010000000010001_2) >> 10 = 001_2 = 1$$

  Page Table Lookup: Page ID 1 is in Page frame 7.

  Offset $=$ Bit Mask & Virtual Address

  $$0001111111111_2 \& 0010000000001_2 = 00000001_2$$

  Physical address $=$ (Page frame in binary $<<$ Offset length) $|$ Offset

  $$\Rightarrow 1110000000000 | 00000001 = 1110000000001$$

Programming exercise 5 (page tables) (3 + 6 = 9 points)

1. Write a C function that represents an unsigned integer as a string in the binary system:

   ```
   void binary_to_string(unsigned long int b, unsigned char k, char* s);
   ```

   Here, b is the number to be represented in binary, k is the number of binary digits to output, and s is a buffer. You may assume that the buffer is at least $k + 1$ characters large.

   Example:

   ```
   char s[6];
   binary_to_string(6, 5, s);
   /* s == "00110" */
   ```

2. Write a C function that converts a virtual memory address to a physical memory address, thus performing the core calculation of a memory management unit:

   ```
   #include <stdint.h> // requires C99
   uint_least16_t translate_address(
           uint_least16_t virtual,
           uint_least32_t const* page_table
   );
   ```

   The first parameter is the virtual address to convert and the second parameter is the page table. The entries in the page table contain only the present/absent bit and the assigned page frame. For efficiency reasons, only bit operators as well as comparison and assignment operators may be used. You may use a local variable (would correspond to using a register). Address sizes and bit masks are given as preprocessor constants (see below).

   Test your function with the page table and the addresses from exercise 4.

   Use the following framework:

- GLHF
- `binary_to_string`
  - iterate over the number of bits to represent
  - What does this do?

    ```
    ((b >> (k - 1 - i)) & 1);
    ```
- `translate_address`
  - you need to use bit shifts and binary ops
  - Extract page table entry (using only the higher 16 - offset bits)
  - If the 32nd bit in the entry is 0 it's a page fault
  - else set the PA bit to 0 (e.g. by using `entry & ~PA_BIT`) and shift to the right place.
  - use virtual & offset mask
  - concat the two above

References I

[1] B. W. Kernighan und D. M. Ritchie, *The C programming language*. 2006.
[2] W. Stallings, *Operating systems: internals and design principles*. Upper Saddle River, NJ: Pearson/Prentice Hall, 2009.
[3] A. Silberschatz, P. B. Galvin und G. Gagne, *Operating system principles*. John Wiley & Sons, 2006.
[4] A. S. Tanenbaum und H. Bos, *Modern operating systems*. Pearson, 2015.