NTNU
Kunnskap for en bedre verden

Department of Computer Science

TDT4186 - Operating Systems

Exercise 8

Exam Prep: A bit of everything

Professor:
Di Liu
Teaching Assistant:
Roman K. Brunner
Student Assistants:
Eik Hvattum Røgeberg
Halvor Linder Henriksen
Ole Ludvig Hozman
Daniel Hansen

Handout: 18.04.2023

Introduction

Since this is the last exercise of the semester, it will contain questions with respect to a wide variety of topics discussed in the course. If you have problems with one or the other question, it might be an indicator of which subjects you should revise again before the exam.

For the multiple-choice questions, there might be multiple correct options. Select all that are correct.

1 C Programming

In the exam, we won't ask you to program a syntactically correct and compileable program. However, we expect that you know, can read, and understand C program code.

Includes have been omitted for brevity in the following samples.

1. **Pointer arithmetic:** Which of the following statements is true? (+1P/correct selected answer, -0.5/wrong answer, no negative points overall)

```c
typedef struct proc
{
    uint64_t pid;
    char name[16];
    uint64_t parent_pid;
} proc_t;

int main(int argc, char *argv[])
{
    proc_t procs[4];

    for (int i = 0; i < 3; i++)
    {
        procs[i].pid = i;
        strncpy(procs[i].name, "proc ", 16);
        procs[i].name[5] = (uint8_t)(i + 48);
        procs[i].name[6] = '\0';
        if (i > 0)
            procs[i].parent_pid = i - 1;
        else
            procs[i].parent_pid = 0;
    }
    proc_t *elem = (proc_t *)((procs + 1));
    *(((uint32_t *)((&(elem->parent_pid)) + 2)) + 2) = UINT32_MAX;

    for (int i = 0; i < 3; i++)
    {
        printf("%s (%ld), parent: %ld\n",
               procs[i].name, procs[i].pid, procs[i].parent_pid);
    }
}
```

Listing 1: Pointer Arithmetic Sample

☐ It outputs

```
proc 0 (0), parent:  0
proc 1 (1), parent:  0
proc 2 (2), parent:  1
```

☐ First two lines as above, but it crashes with a SEGFAULT when trying to print the

third line

- ☐ It segfaults when assigning on line 24
- ☐ It segfaults on line 16
- ☐ First two lines as in the first option, but the third looks like this
  `proc□□□□ (2), parent:  1`
- ☐ It does not compile because on line 23 an array is assigned to a pointer

2. **Bitwise Operators:** Which of the following statements is true? (+1P/correct selected answer, -0.5/wrong answer, no negative points overall)

```
1  typedef uint32_t pte_t;
2  #define FLAG_MASK 0x3FF
3  #define FLAG_BITS 0x8
4  #define PA(pte) ((pte >> FLAG_BITS) << FLAG_BITS)
5  #define PTE_V (0x1 << 0)
6  #define PTE_R (0x1 << 1)
7  #define PTE_W (0x1 << 2)
8  #define PTE_X (0x1 << 3)
9
10 int main(int argc, char *argv[])
11 {
12     pte_t entry1 = 0xC9CF4BD3;
13     pte_t entry2 = 0xC9CFFFD3;
14
15     if (!(PTE_V | entry1) || !(PTE_V | entry2))
16     {
17         return 1;
18     }
19     else
20     {
21         entry1 = entry1 | PTE_X;
22         entry2 = entry2 | PTE_W;
23     }
24
25     uint32_t pa1 = PA(entry1);
26     uint32_t pa2 = PA(entry2);
27
28     uint32_t diff = pa1 ^ pa2;
29     if (diff == 0)
30     {
31         return 2;
32     }
33     uint8_t len = 0;
34     while (diff > 0)
35     {
36         diff = diff >> 1;
37         len++;
38     }
39     printf("%d\n", len);
40 }
```

Listing 2: Bitwise Operations

- ☐ Line 21 and 22 each change a single bit in the respective entries
- ☐ The variable diff contains the value 1 at the end of the execution

- ☐ It will not compile because it can't cast `pte_t` to `uint32_t` in line 25
- ☐ It will compile but will exit on line 17, returning 1
- ☐ It will compile but it will exit on line 31, returning 2
- ☐ It will compile and reach the printf on line 39

3. What output will the program in Listing 2 generate? (3 Points)
4. **SPOILER ALERT: IF YOU HAVEN'T SOLVED THE FIRST EXERCISE, THE SOLUTION IS CONTAINED IN THE FOLLOWING QUESTION**
   Please explain where the value that is assigned in cde:pointers is gone. Why don't we see it in the output? (3 Points)

2 Scheduling, Processes, and Threads

1. Which of the following statements is true? (+1P/correct selected answer, -0.5/wrong answer, no negative points overall)
   - ☐ Cooperative scheduling relies on hardware support to turn over control to the scheduler
   - ☐ Direct execution always runs a program to completion/forever
   - ☐ Processes and Threads are different names for the same concept
   - ☐ The OS kernel runs in its own process
   - ☐ Timer interrupts are only required for preemptive schedulers
   - ☐ A process can directly access hardware and shared resources
2. Assume that we have a simple operating system which supports processes. Those processes can have the following states: CREATED, RUNNABLE, RUNNING, PAUSED, DEAD. Draw a state diagram of a processes lifecycle. Please label the edges in your diagram with which actions will lead to this transition. (2 points)
3. You are tasked with writing a small operating system for a handheld device. The user will be able to install and run third-party software (coming from developers you don't know). The device should be low power, and thus can at any point in time only run two processes to improve battery live. Which approach do you choose to enable multi-processing? What do you need to pay attention to?
4. You are working on a embedded system for self-driving cars. It needs to run a set of tasks periodically and complete each task before its deadline. Is it possible to schedule the jobs in Table 1 such that all jobs meet their deadline or do you need a second device to make it work? The arrival time describes when a task is added to the system for the first time, the execution time describes how long a task takes, the deadline describes how many time units after arriving a job needs to be finished, and the interval describes after how many time units another instance of the same job is added to the system (e.g. for Job 3, the next instance will be added at timeslot 13).

Job ID	Arrival Time	Execution Time	Deadline	Interval
1	0	2	3	5
2	0	3	10	10
3	3	3	3	10

Table 1: Jobs with deadline and interval

3 Memory Management

1. Take a look at the small program snippet below. Which of the following statements are true? Select all that you think are true. (1P for each correctly selected/not selected option, -0.5P for each wrongly selected/not selected option, minimum points overall: 0)

```
1  int add(int a, int b) {
2      int sum = 0;
3      sum += a;
4      sum += b;
5      return sum;
6  }
7
8  int main(int argc, char *argv[])
9  {
10     int size = 2;
11     if (argc > 1)
12         size = argc;
13     int *all_ints = malloc(argc * sizeof(int));
14     for (int i = 0; i < size; i++)
15         *(all_ints + i) = add(argc, i);
16     printf("all ints: ");
17     for (int i = 0; i < size; i++)
18         printf("%d, ", *(all_ints + i));
19     free(all_ints);
20 }
```

Listing 3: Sample Program, includes are left out for brevity

- ☐ The string for the printf on line 16 is stored in the heap
- ☐ The memory pointed to by all_ints lives on the heap
- ☐ Since all_ints is a local pointer variable it lives on the stack
- ☐ The size variable is allocated on the heap because it is initialised with an argument value
- ☐ The sum variable is allocated on the stack
- ☐ The sum function has its own stack that is independent of the main function stack

2. We learned that the stack grows downwards whenever new things are pushed onto the stack and shrinks upwards once they are popped off. How does the heap grow and shrink?

3. You are developping a new small operating system for an embedded device that runs a predefined set of processes. How do you partition the memory within the operating system between the processes? Give an explanation for your answer.

4 Paging, Page Tables & Translation Lookaside Buffer

In the exercises in this section we use the following page tables and page directory in tables 2 to 5.

1. Looking up physical addresses in a page table structure for every access is inefficient, especially for deeper nested page table structures. Suggest a possible solution to improve the virtual to physical address translation speed for the most commonly used addresses.

2. Translate the virtual address `0xC010FF00` to a physical address and explain how you end up at your solution. The Page Directory and the Page Tables have the same amount of entries and a Page contains 4096 bytes.

Index	20	12
	...	
0x034	0xDEAD0	flags
	...	
0x300	0xBEEF0	flags
	...	
0x3FB	0x2BAD2	flags
	...	

Table 2: Page Directory stored at 0xBADCAFE

	20	12
	...	
0x000	0x00000	flags
	...	
0x06F	0x11111	flags
	...	
0x0CC	0x22222	flags
	...	

Table 3: Page Table stored at 0xDEAD0000

	20	12
	...	
0x000	0x33333	flags
0x001	0x66666	flags
	...	
0x10F	0xCCCCC	flags
	...	

Table 4: Page Table stored at 0xBEEF0000

	20	12
	...	
0x1CE	0x99998	flags
	...	
0x2D2	0x33330	flags
	...	
0x2DC	0x66660	flags
	...	

Table 5: Page Table stored at 0x2BAD2000

3. Given the physical address 0x00000115, what is the corresponding virtual address?
4. You are given the following Page Directory/Page Table configuration. How many bits must an address contain for this configuration to be sound?
   - Page Directory: 64K Entries
   - Page Table: 4K Entries
   - Page Size: 8KB
5. What is the maximum size of memory the system could handle in the previous subtask? Calculate it based on the size of the address and the page directory/page table.

5 Processes, Threads & Synchronisation

1. Which of the following statements about locks is true?
   - ☐ To make sure only processes that are allowed to access a resource are accessing it
   - ☐ We don't really need locks, this is to better communicate the ordering in code with other developers
   - ☐ To ensure sequential consistency
   - ☐ We don't necessarily need locks; we can replace all locks by lock-free data structures
   - ☐ To exclude others (processes, threads) from changing certain state during a limited timeframe
   - ☐ Locks are only required when multi-threading as processes are isolated from each other
2. Explain the difference between a process and a thread. What is the main distinguishing factor between the two?
3. Go to https://deadlockempire.github.io/ and go through the samples there. Samples like that might be part of the exam.

6 I/O Scheduling, DMA & RAID

1. We discussed I/O scheduling mainly to reduce seek time on spinning hard disk drives. However, many modern systems use SSDs instead of HDDs. So is I/O scheduling in SSD only systems still important?

2. Which of the following statements is true?

   ☐ RAID is a technique to reduce latency when reading data from HDDs

   ☐ RAID 5 improves the reliability and the throughput when reading/writing HDDs

   ☐ DMA allows a device to access HDD storage directly, bypassing the CPU

   ☐ RAID is only useful if we expect a disk to fail

   ☐ RAID 0 improves the reliability of the system

   ☐ RAID 5 gives us more reliability than RAID 4

3. You are tasked with increasing performance in a system. You are allowed to make changes to the hardware as well. You see from the analysis that the CPU spends the most time just moving data from the network card to the memory. What could you do instead? How does this affect the system and how would it complicate the hardware design? Does this increase the systems concurrency?

7 File System & Unix I/O

1. Which of the following statements is true?

   ☐ An inode contains the filename

   ☐ Filesystems relatively constrained in design space as they have to follow the hardware's structure

   ☐ In order to guarantee consistency, the OS must always resolve paths from the inode structure

   ☐ We use bitmaps to indicate if an inode is used or not

   ☐ Multi-level indexes improve performance, since one can use binary search to search for a file in parallel

   ☐ The maximal file size is only limited by the amount of free storage on disk

2. Assume the numbers given for the Barracuda Disk in Figure 37.5 in chapter 37.4[1]. How long does it take to move a folder with 8 files, each file being of the size of 128 KB, into another folder? So how long does it take to execute `mv /a/f /b`, where `/a/f` is a directory, containing 8 files of the size 128 KB.

3. How would the speed change in the previous task if instead we were using an SSD? Assume the speeds given in chapter 44 of the ostep book, page 18. If you need latency, you can assume 0.5 ms. Explain why there is a difference in speed.

4. Propose an inode strcutre such that we can store files up to 8 GB. How much overhead does this introduce?

5. You created a new device that accelerates machine learning computations and you want to make it work under Linux, as most of today's machine learning happens on Linux machines. How would you expose that to the userspace? How would this be integrated into the operating system?

[1] You don't need to know these numbers by heart, we will provide whatever numbers you need to know during the exam.