8-hour Take-Home Exam in Computer Systems

Department of Computer Science, University of Copenhagen (DIKU)
Date: January 27, 2021

Preamble

Solution

Disclaimer: The reference solutions in the following range from exact results to sketch solutions; this is entirely on purpose. Some of the assignments are very open, which reflects to our solutions being just one of many. Of course we try to give a good solution and even solutions that are much more detailed than what we expect you to give. On the other hand, you would expect to give more detailed solutions that our sketches. Given the broad spectrum of solutions, it is not possible to directly infer any grade level from this document. All solutions have been written in the in-lined space with this colour.

This is the exam set for the 8 hour written take-home exam in Computer Systems (CompSys), B1+2-2020/21. This document consists of 28 pages excluding this preamble; make sure you have them all. Read the rest of this preamble carefully. Your submission will be graded as a whole, on the 7-point grading scale, with external censorship.

I addition to this document you are given two templates to write your answer. One is formatted as a LaTeX document, while the other is a pdf-file with included form fields. You can use either of these to make your answer.

- You can answer in either Danish or English.
- Remember to write your exam number (and only your exam number) on the hand-in.
- Do not exceed the line limits indicated in the answer-templates (see below).
- Only hand-in one pdf-file with your answers on eksamen.ku.dk. Note: you can only hand-in once.

Expected usage of time and space

The set is divided into sub-parts that each are given a rough guiding estimate of the size in relation to the entire set. However, your exact usage of time can differ depending on prior knowledge and skill.

Furthermore, all questions in the answer-templates includes limitation to the number of lines for answers in a standard page and font formatting. These limitations are more than enough to include a complete and satisfactory answer of the question; thus, full answers of the question does not necessarily use all available space.
These limitations are strict, meaning we reserve the right to *not* evaluate any part of the answer exceeding the given limits.

Exam Policy

This is an *individual*, open-book exam. Thus, you are not allowed to discuss the exam set with anyone else, until the exam is over (be aware of co-students that can have gotten extra time). It is expressly forbidden:

Exam number: _____

- To discuss or share any part of this exam, including, but not limited to, partial solutions, with anyone else.
- To help or receive help from others.
- To seek inspiration from other sources, including the Internet, without proper citation in your answer.
- Copy-paste a text (even if it partly edited) without proper citation in your answer.
- To post any questions or answers related to the exam on *any fora*, before the exam is over, including the course discussion forum on Absalon and Discord.

Breaches of this policy will be handled in accordance with the standard disciplinary procedures. Possible consequences range from your work being considered `void`, to *expulsion from the university*[1].

You may use the course book, notes and any documents printed or stored on your computer and other device. If you use any other material, you must cite this in your answer.

Errors and Ambiguities

In the event of errors or ambiguities in the exam text, you are generally expected to state your assumptions as to the intended meaning in your answer. Some ambiguities may be intentional.

If you are in doubt, you can contact `m.kirkedal@di.ku.dk` for clarification. But you will not get answers to clarification of course curriculum.

If there during the exam are corrections or clarifications to any question, these will be posted as an announcement on Absalon course page. Be sure that you receive notifications from this.

IMPORTANT
It is important to consider the context and expectations of the exam sets. Each question is designed to cover one of more learning goals from the course. The total exam set will cover all or (more likely) a subset of all the learning goals; this will vary from year to year.
The course has many more learning goals than are realistic to cover during a 4-hour exam. And even though we limit the tested learning goals, it will still be too much.
Therefore, it is not expected that you give full and correct answer to all questions. Instead you can focus on parts in which you can show what you have learned.
It is, however, important to note that your effort will be graded as a whole. Thus, showing your knowledge in a wide range of topics is better that specialising in a specific topic. This is specially true when considering the three overall topics: Arc, OS, and CN.

Specifically, for this exam set it was need to solve:
* about 40 % to pass
* about 55 % to get a mid-range grade
* about 75 % to get a top grade
* no one solved all parts correctly!
NB! we adjust the exam set from year to year, so the above is not written in stone and can change slightly for your exam.

[1] https://uddannelseskvalitet.ku.dk/docs/overordnede-dokumenter/Ordensregler-010914.pdf

8-hour Take-Home Exam in Computer Systems
Department of Computer Science, University of Copenhagen
Date: January 27, 2021

Exam number: _____

1 Machine architecture (about 33 %)

1.1 Assembler programming (about 13 %)

Consider the following program written in X86prime-assembler.

```
.L0:
    subq $8, %rsp
    movq %r11, (%rsp)
    cbe $0, %rsi, .L13
    leaq (%rdx), %r11
    movq $0, %r10
    movq %r10, (%r11)
    leaq (%rcx), %r11
    movq $0, %r10
    movq %r10, (%r11)
    cbae $1, %rsi, .L13
    movq $1, %eax
    jmp .L2
.L1:
    addq $1, %rax
    cbe %rax, %rsi, .L13
.L2:
    leaq (%rdi, %rax, 8), %r11
    movq (%r11), %r8
    movq (%rdx), %r10
    cbae %r10, %r8, .L1
    movq %r8, (%rdx)
    movq %rax, (%rcx)
    jmp .L1
.L13:
    movq (%rsp), %r11
    addq $8, %rsp
    ret %r11
```

Question 1.1.1: Describe the details of the program. This includes (but is not limited to):

- Identify the instructions that implements the call convention; makes it possible to call the piece of code as a function.
- Identify the control structures of the program (e.g. loops, conditionals, and function calls).
- Identify registers which must have specific values at call time, for the function to have a specific purpose.
- Specify for each used register, the C type that best reflects the use of the register. (If a register is used in more than one way, specify this.)

Instructions implementing the calling convention:
This is the 2 initial instructions (the prologue) and the final 3 instructions (the epilogue)

We can identify:
- an outer if
- a middle if (within the outer if)
- a loop (within the middle if, starting at L1)
- an if whithin the loop but there is also something that doesn't quite look like structured control flow, a jump directly into the loop, above translated as "goto L2." Marked as "WTF" above.

Deducing the structure above is considered a fully satisfying answer, even though it includes what appears to be a "goto".

The following registers must have specific values at entry, since they are read before being written by the code:
- `%sp` (stack pointer as usual, part of the calling convention)
- `%r11` (return address as usual, part of the calling convention)
- `%rsi`
- `%rdx`
- `%rcx`
- `%rdi`

C types for each register used in the program (omitting `%sp` and `%r11` at entry/exit) arguments
- `%rsi`: `unsigned long`
- `%rdx`: `unsigned long*`
- `%rcx`: `unsigned long*`
- `%rdi`: `unsigned long*`

other variables
- `%r11`: `unsigned long*`
- `%r10`: `unsigned long`
- `%r8`: `unsigned long`

Aside: (the following is not required to be part of an answer)
This code is the result of a tricky compiler optimization. Assume you've got:

```
a = 0;
while (a < b) {
  if (...) { ... }
  ++a;
}
```

where the compiler wants to optimize the if inside the loop, such that its closing brace can be implemented by a straight jump to the start of the loop. The problem is that the "++a" prevents this. So the compiler "rolls" the loop, moving the ++a to the beginning of the loop and fusing it with the loop condition:

```
a = -1;
while (++a < b) {
  if (...) { ... }
}
```

this requires 'a' to be initially offset by -1. But -1 is not representable as an unsigned variable. And in this case 'a' is unsigned (you can infer that from the conditionals, they are unsigned comparisons).
So the compiler employs a trick to avoid generating the value -1. Instead it generates:

```
a = 0;
goto Lbypass;
while (++a < b) {
  Lbypass:
  if (...) { ... }
}
```

thereby skipping the initial increment and condition evaluation. In oder for this transformation to be correct, however, it needs to make sure that the (now bypassed) condition would indeed evaluate to true. It ensures this by:

```
if (b > 1) {
  a = 0;
  goto Lbypass;
  while (++a < b) {
    Lbypass:
    if (...) { ... }
  }
}
```

and finally we end up with the program structure in this exercise.

Question 1.1.2: Rewrite the above X86prime-assembler program to a C program. The resulting program must not have a goto-style and minor syntactical mistakes are acceptable.

```
void max(unsigned long* array,
         unsigned long size,
         unsigned long* max_value,
         unsigned long* index_of_max_value) {
  if (size) {
    unsigned long index = 0;
    *max_value = 0;
    *index_of_max_value = 0;
    index++;
    while (index < size) {
      if (array[index] > *max_value) {
        *max_value = array[index];
        *index_of_max_value = index;
      }
      ++index;
    }
  }
}
```

An answer which has the more complex control flow isolated in 1.1.1 is also accepted as an answer here.

Question 1.1.3: Descripe shortly the functionality of the program.

The program search an array of unsigned integers to find the largest element. It produces the value of the largest element and the index into the array of the largest element. The value and index are stored at pointers that are given as arguments to the function. So the function formally returns void, but it really "returns" size and position of largest element within an array of elements.

1.2 Performance / Pipeline (about 10 %)

Consider the following execution graph (afviklingsplot) that illustrates the execution of a sequence of instructions on a 3-step pipeline machine, similar to the one used in A2.

```
.L3:
    movq (%r14), %rax           FXM
    cbe %rax, %rsi, .L1          FXXM            1)
.L4:
    leaq (%rdi, %rax, 8), %r11    FFXM           2)
    movq (%r11), %r8                FXM
    movq (%rdx), %r10                FXM
    cbae %r10, %r8, .L3               FXXM       3)
    movq %r8, (%rdx)                   FFXM      4)
    movq %rax, (%rcx)                     FXM
    jmp .L3                                FXM
.L3:
    addq $1, %rax                             FXM
```

Question 1.2.1: The numbes 1) to 4) indicates the four situations, where an instruction is in the same pipeline step for more than one machine cycle.

For each of the four cases describe why this is the case.

Cause for stall in each of the four cases:

1) Instruction needs to wait in X stage to get valid operand from the register file.

2) Instruction needs to wait in F stage, because previous instruction is waiting in X stage.

3) as 1.

4) as 2.

Exam number: _____

Question 1.2.2: In the note on execution graphs another "simple" pipeline with 5 steps is described; see https://x86prime.github.io/afviklingsplot/pipeline/ for more details.

Make an execution diagram, showing the execution of above code on this machine. Describe the interesting parts; e.g. when and why you stall the pipeline.

	Code	Timing																		
	`.L3:`																			
1	`movq (%r14), %rax`	F	D	X	M	W														
2	`cbe %rax, %rsi, .L1.`		F	D	D	X	M	W												1)
3	`.L4:`																			
4	`leaq (%rdi, %rax, 8), %r11`			F	F	D	X	M	W											2)
5	`movq (%r11), %r8.`					F	D	X	M	W										
6	`movq (%rdx), %r10`						F	D	X	M	W									
7	`cbae %r10, %r8, .L3`							F	D	D	X	M	W							3)
8	`movq %r8, (%rdx)`								F	F	D	X	M	W						4)
9	`movq %rax, (%rcx)`										F	D	X	M	W					
10	`jmp .L3`											F	D	X	M	W				
11	`.L3:`																			
12	`addq $1, %rax`													F	D	X	M	W		

13 cycles (X to X, both included)

There is a bug in the exercise, since the instruction following L3 the second time is different from the one following L3 the first time. This is not possible.
The stalls are on the same instructions as given in 1.2.1 and for the same reasons:
for 1) and 3) an instruction must wait for the result of a previous load from memory.
for 2) and 4) an instruction must wait because the previous instruction waits.

Furthermore, The reference to the note on execution plots is (unintendedly) imprecise, as it refers to a page with multiple sections, and these sections do not describe the same machine. The sections on "Pipeline faser og ressourcer" and on "Eksempel: Simpel pipeline mikroarkitektur" are fine (and form the intended background for this question), but the following sections from "Latenstid af faser" appears to contradict or at least presents a slightly different pipeline where memory references spend an extra cycle in the M-stage. Because of this confusion, answers can be based either on the first part or on the second part, yielding different results. Both are accepted.

The section (in the note) on control dependencies does not explicitly mention handling of an unconditional jump (only call, ret and conditional jumps are mentioned), leaving it up for interpretation. The correct interpretation is that a jump can redirect instruction fetch as fast as a call, so the F-stage of the target instruction has to follow the D-stage of the jmp.

Question 1.2.3: A 5-step pipeline is expected to perform less work in each step and thus have a shorter longest signal path. Assume therefore that the 5-step pipeline is 25 % faster (in the sense that it's clock frequency is 25 % faster) than the 3-step pipeline.

Which of the two pipelines executes the program sequence fastest? Why is this?

The 5-stage pipeline is 25% percent faster than the 3-stage pipeline.
Since the two pipelines uses the same number of cycles (meassured from e.g. X to X on the first respectively the last instruction), the improvement in clock frequency directly gives the total improvement.

In case the answer is based on the second half of the pipeline note (see previous exercise/answer), 4 additional stall cycles will occur. This is enough to offset the gain in frequency, so the 3-stage pipeline will be faster.

However - if the 5-stage pipeline takes advantage of the fact that the jmp instruction does not actually need the X-stage, it could allow the jump target at L3 to advance to X one cycle earlier. This will make the 5-stage pipeline slightly faster.

1.3 Data Cache (about 10 %)

Question 1.3.1: You can choose between a "direct mapped" and a 4-way associative cache. The two caches have same total size and access time.

Which type of cache will you choose and why?

The 4-way associative cache is to be preferred over the direct mapped. The reason is that it will on average have a lower miss rate due to more flexibility in where to place data. Consider a situation where multiple addresses map to the same cache index, but still hits different blocks. A 4-way associative cache can accommodate 4 such conflicting references per index, while the direct mapped can hold only one.

Question 1.3.2: Explain shortly the difference between "write-back" and "write-through" caching.

When writing to memory 2 different policies are possible:
a) you can immediately update the memory (write-through) or
b) you can update the cache only, saving the write to memory for later (write-back).

Question 1.3.3: In the following we examine a 4-way set-associative data cache with 16 cache-blocks of 16 bytes each.

Indicate for each of the references in the following stream, if the cache access is a miss or a hit. Addresses are given in heximal notation, and touches a single byte. Assume the cache is cold on entry.

`0x000, 0x001, 0x01F, 0x102, 0x203, 0x304, 0x10C, 0x705, 0x002, 0x10F, 0x210, 0x010.`

Reference	Hit/Miss
`0x000`	M
`0x001`	H
`0x01F`	M
`0x102`	M
`0x203`	M
`0x304`	M
`0x10C`	H
`0x705`	M
`0x002`	M
`0x10F`	H
`0x210`	M
`0x010`	H

Address contains Tag:Index:Offset. Sizes: offset: 4 bits, Index: 2 bits, Tag: the rest
We assume LRU replacement. Relevant cache content is given as a list of index:tags, where tags is a list of tags in the set in order of reference, least recently used is last.
Least 2 significant bits of Tag is omitted, they are always 0 in this exercise.

Address	Index	Tag	Hit/Miss	Relevant cache content after access
0x000	0	0	miss	0:{0}
0x001	0	0	hit	0:{0}
0x01F	1	0	miss	0:{0}, 1:{0}
0x102	0	1	miss	0:{1,0}, 1:{0}
0x203	0	2	miss	0:{2,1,0}, 1:{0}
0x304	0	3	miss	0:{3,2,1,0}, 1:{0}
0x10C	0	1	hit	0:{1,3,2,0}, 1:{0}
0x705	0	7	miss	0:{7,1,3,2}, 1:{0}
0x002	0	0	miss	0:{0,7,1,3}, 1:{0}
0x10F	0	1	hit	0:{1,0,7,3}, 1:{0}
0x210	1	2	miss	0:{1,0,7,3}, 1:{2,0}
0x010	1	0	hit	0:{1,0,7,3}, 1:{0,2}

Question 1.3.4: Now assume that the cache is only 2-way set-associative, but the total size is the same as the one above.

Again indicate for each of the references in the following stream, if the cache access is a miss or a hit. Addresses are given in heximal notation, and touches a single byte. Assume the cache is cold on entry.

`0x000, 0x001, 0x01F, 0x102, 0x203, 0x304, 0x10C, 0x705, 0x002, 0x10F, 0x210, 0x010.`

Reference	Hit/Miss
0x000	M
0x001	H
0x01F	M
0x102	M
0x203	M
0x304	M
0x10C	M
0x705	M
0x002	M
0x10F	M
0x210	M
0x010	H

Same size, but two way associative. Implies that there are 8 sets instead of 4.
2-way set associative cache with 16 cache blocks of 16 bytes each. Address contains Tag:Index:Offset. Sizes: offset: 4 bits, Index: 3 bits, Tag: the rest
Least significant bit of Tag is omitted, it is always 0 in this exercise.

Address	Index	Tag	Hit/Miss	Relevant cache content after access
0x000	0	0	miss	0:{0}
0x001	0	0	hit	0:{0}
0x01F	1	0	miss	0:{0}, 1:{0}
0x102	0	1	miss	0:{1,0}, 1:{0}
0x203	0	2	miss	0:{2,1}, 1:{0}
0x304	0	3	miss	0:{3,2}, 1:{0}
0x10C	0	1	miss	0:{1,3}, 1:{0}
0x705	0	7	miss	0:{7,1}, 1:{0}
0x002	0	0	miss	0:{0,7}, 1:{0}
0x10F	0	1	miss	0:{1,0}, 1:{0}
0x210	1	2	miss	0:{1,0}, 1:{2,0}
0x010	1	0	hit	0:{1,0}, 1:{0,2}

Question 1.3.5: Explain the difference between the 2 sequences of events? Why do you see these differences?

The 2-way associative cache has fewer hits than the 4-way associative cache.
This difference is due to more conflicts on access to the 2-way associative cache compared to the 4-way associative cache. Conflicts are situations where addresses map to the same set, but since each set is smaller, less references can stay in the set at any one time. This increases the number of times when one reference causes eviction of an earlier one, even though a following reference may need it later, leading to extra cache-misses in the case of the 2-way associative cache.

2 Operating Systems (about 33 %)

2.1 Multiple Choice Questions (about 6 %)

In each of the following questions, you may put one or more answers. Use the lines to argue for your choices.

Question 2.1.1: Which of the following are states that a Unix process can be in?

☐ **a)** Crashed

☒ **b)** Running

☐ **c)** Reaped

☒ **d)** Stopped

☒ **e)** Waiting

☒ **f)** Zombie

Question 2.1.2: Why is virtual memory useful?

☐ **a)** To download RAM from the Internet

☒ **b)** So processes can use more memory than will fit in RAM

☐ **c)** Protecting a process's memory from being accessed by the kernel

☒ **d)** Protect a process's memory from being accessed by another process

☒ **e)** More efficient use of RAM

☐ **f)** Because otherwise `malloc()` could not be implemented.

Question 2.1.3: What is the purpose of the dirty bit in page table entries?

☒ **a)** Avoiding unneeded disk writes

☐ **b)** Marking the page as read-only.

☐ **c)** Maintains LRU information for page replacement stragies.

☐ **d)** Whether the entry is valid.

2.2 Short Questions (about 12 %)

Question 2.2.1: Does the following program contain race conditions? Why or why not?

```c
#include <unistd.h>
#include <stdio.h>

int x = 0;

void* worker(void* p) {
  for (int i = 0; i < 1000; i++)
    x++;
  return NULL;
}

int main() {
  pthread_t t;
  pthread_create(&t, NULL, worker, NULL);
  pthread_join(t);
  pthread_create(&t, NULL, worker, NULL);
  pthread_join(t);
  printf("%d\n", x);
}
```

No race conditions, because only one thread is ever accessing the variable `x` concurrently.

Exam number: _____

Question 2.2.2: Consider a system with the following properties:

- Memory is byte-addressed.
- Virtual addresses are 14 bits wide.
- Physical addresses are 13 bits wide.
- The page size is 16 bytes.
- The TLB is 3-way set associative with 8 sets and 24 total entries. Its initial contents are:

Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	0F	10	1	02	00	1	09	00	0
1	20	10	0	11	15	1	1F	2F	1
2	12	15	1	00	12	0	19	0A	0
3	4F	10	1	12	00	0	19	00	0
4	21	34	1	11	00	0	30	0A	1
5	30	10	0	41	15	1	2F	3F	1
6	22	15	1	10	12	0	29	FA	0
7	01	00	0	31	34	1	20	00	1

- The page table contains 12 PTEs:

VPN	PPN	Valid	VPN	PPN	Valid	VPN	PPN	Valid	VPN	PPN	Valid
02	33	1	0C	0D	0	08	17	1	13	11	1
32	33	1	01	02	0	41	01	1	04	01	1
10	21	0	00	00	1	A2	32	0	03	43	1

Note that all addresses are given in hexadecimal. In the following questions, you are asked, for various virtual addresses, to show the translation from virtual to physical addresses in the memory system just described. *Hint: there is one TLB hit, one page table hit, and one page fault (not necessarily in that order). This should help you double-check your work.*

The three addresses in the following did by mistake all have a ending "4": e.g. `14e54` instead of `14e5` as shown now. This was communicated to the students 40 minutes into the exam.

Exam number: _____

Virtual address: `14e5`

1. Bits of virtual address

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	1	1	1	0	0	1	0	1

2. Address translation

Parameter	Value
VPN	14e
TLB index	6
TLB tag	29
TLB hit? (Y/N)	Y
Page fault? (Y/N)	N
PPN	fa

3. Bits of phys. (if any)

12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	0	1	0	0	1	0	1

Virtual address: `87`

1. Bits of virtual address

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	0	0	1	1	1

2. Address translation

Parameter	Value
VPN	8
TLB index	0
TLB tag	1
TLB hit? (Y/N)	N
Page fault? (Y/N)	N
PPN	17

3. Bits of phys. (if any)

12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	1	0	1	1	1

Exam number: _____

Virtual address: `14df`

1. Bits of virtual address

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	1	1	0	1	1	1	1	1

2. Address translation

Parameter	Value
VPN	14d
TLB index	5
TLB tag	29
TLB hit? (Y/N)	N
Page fault? (Y/N)	Y
PPN	

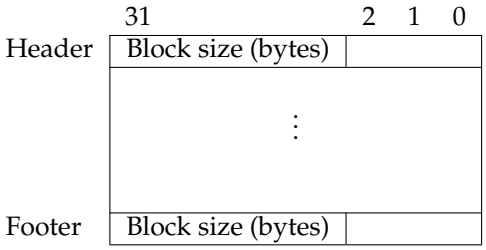
3. Bits of phys. (if any)

12	11	10	9	8	7	6	5	4	3	2	1	0

Also answer the following: Describe shortly how you calculated your answers.

2.3 Long Questions (about 15 %)

Question 2.3.1: Consider an allocator that uses an implicit free list. The layout of each allocated and free memory block is as follows, with one 32-bit word per row:



Each memory block, either allocated or free, has a size that is a multiple of eight bytes, rounding up allocations if necessary. Thus, only the 29 higher order bits in the header and footer are needed to record block size, which includes the header and footer. The usage of the remaining 3 lower order bits is as follows:

- `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
- `bit 1` indicates the use of the previous adjacent block: 1 for allocated, 0 for free.
- `bit 2` is unused and is always set to be 0.

Important: We must *never* create blocks with zero payload (i.e. we must *never* create blocks with size 8).

Given the heap shown on the left, show the new heap contents after *consecutive* calls to

1. `free(0x500c010)`.
2. `realloc(0x500c000, 12)`. Assume the the return value is `0x500c000`, meaning that the existing allocation is resized.

Your answers should be given as hex values. Note that the address grows from bottom up. Assume that the allocator uses immediate coalescing, that is, adjacent free blocks are merged immediately each time a block is freed.

Exam number: _____

Address	Original value	After `free`	After `realloc`
0x500c028	0x00000013	0x11	0x13
0x500c024	0x500c611c	0x500c611c	0x500c611c
0x500c020	0x500c512c	0x500c512c	0x500c512c
0x500c01c	0x00000013	0x11	0x13
0x500c018	0x00000013	0x12	0x23
0x500c014	0x500c511c	0x500c511c	0x500c511c
0x500c010	0x500c601c	0x500c601c	0x500c601c
0x500c00c	0x00000013	0x12	0x12
0x500c008	0x00000013	0x13	0x13
0x500c004	0x500c601c	0x500c601c	0x500c601c
0x500c000	0x500c511c	0x500c511c	0x500c511c
0x500bffc	0x00000013	0x13	0x23

Also answer the following: Does the resulting heap exhibit internal or external fragmentation? Explain your answer.

Yes, the block whose payload starts at 0x500c000 was asked to contain 12 bytes of payload, but ended up being a 32-byte block due to the 8 bytes of headers/footers (necessary), but also having to round up the allocation to be a multiple of 8 bytes, and then further expanding to avoid a following zero-payload block.

Exam number: _____

Question 2.3.2: Explain, in pseudocode or prose, how semaphores can be implemented in terms of mutexes and integer variables. Show how to initialise a semaphore with initial value v, how to implement the P operation, and how to implement the V operation. Could your implementation be made more efficient by also using condition variables?

We represent a semaphore with a counter c and a mutex m. Initialisation: Set $c \leftarrow v$ and $m \leftarrow$ unlocked. P: (1) Lock m; (2) If $c = 0$, unlock m and go to (1); Otherwise, $c \leftarrow c - 1$, and unlock m. V: (1) Lock m; (2) $c \leftarrow c + 1$ (3); Unlock m. Condition variables could make the implementation of P more efficient, as instead of continuously looping, we would wait on the condition variable, and expect V to wake us up.

3 Computer Networks (about 34 %)

3.1 Application layer (about 6 %)

Question 3.1.1: Suppose you are starting a new company and want to host the web-page of the company. Discuss the difference between hosting the web-page using a content distribution network like Akamai in comparison to hosting it on peer-to-peer networks like BitTorrent.

A CDN would be beneficial for hosting web-content compared to P2P because of the following reasons:
Availability and Fault Tolerance: CDN ensures that the availability of the content is not dependent on the number of active clients, their bandwidth limitations, P2P configuration settings but rather on the load distribution on the server side in the CDN. This also includes fault tolerant guarantees.
Level of Control: Usage of a CDN would provide a high-level of control to the service to configure the locations of the CDN services, load-balancing of available servers which is not possible to do using a P2P service. Configuration based on geographic locations, distance to clients.
Caching Implications: For content that changes often, CDNs can utilize efficient caching strategies to minimize the cost of propagation of changes. Managing this in a P2P service would require more sophisticated version management and is often impractical to implement without client intervention.
Maintenance Overhead: Usage of CDN allows the hosting service to make all possible optimizations at different layers of the networks stacks (DNS load balancing, usage of transport layer protocols, usage of network layer multi-casting protocols) which is not possible to do by only utilizing application-level P2P service.
Economic Implications: (Extra) Usage of a P2P service would put the burden of network transfer costs on the the client for distributing the web-pages of the company.

Question 3.1.2: What role does DNS play for content distribution networks and how is it used?

The above answer is way more detailed than expected, and I expect the students would focus most on availability and control. Here the students are reminded and can detail the caching and maintenance parts.

3.2 Reliable Data Transfer (about 8 %)

Question 3.2.1: How does flow control in TCP help with congestion control? Why is flow control not enough to deal with congestion control and conversely congestion control not enough to deal with flow control?

TCP implements a flow-control service to ensure that the receive buffers on a receiver are not overwhelmed by a fast sender. It implements congestion control to guarantee senders can throttle its sending rate as a function of the *perceived* network traffic.
By implementing a flow control service, a sender ensures that it never overflows the receive buffers on the receiver by tracking the receive window on the receiver. This ensures that there are no wasted packets sent by the sender which cannot be stored in the receive buffer of the receiver and hence reduces the strain on the congestion in the network and helps with congestion control.
Implementing flow control is not enough for congestion control because flow control only ensures that the senders do not overwhelm the receiver's buffer but does not ensure that the cumulative effect of sending and receiving between multiple senders and receivers does not exceed the capacity of the links and the routers.
With a single receiver and sender, congestion control can also be used to infer the receive window size using the congestion window size albeit inefficiently. However, in the presence of multiple senders and receivers, using congestion control only will cannot guarantee correct estimation of the receive window sizes and hence is not enough to deal with flow control.

Question 3.2.2: Suppose there are k TCP connection ongoing over a shared link with bandwidth R. In addition to the k TCP connections, another TCP connection is initiated over this shared link. What is the bandwidth available to the new TCP connection and why?

Assuming equal RTT, window sizes and each of the TCP connections operating are under AIMD phase of congestion control algorithm (see Section 3.7.1 in KR), the bandwidth of the shared connection link would be equally shared between the k+1 TCP connections because of the nature of the congestion control algorithm. This would imply each of the TCP connections would have a bandwidth equal to R/(k+1) available. The assumption of all TCP connections entering the AIMD phase of the congestion control algorithm is important for the fair sharing guarantee.

Question 3.2.3: If instead of TCP, UDP had been used for the k connections, how much bandwidth would be available to the new UDP connection and why?

No such fair-sharing of bandwidth happens in UDP which will have potentially the entire bandwidth available and packet losses will happen depending upon the rates at which the packets are pumped into the link by the senders and the capacity of the link.

3.3 Network Layer (about 10 %)

Question 3.3.1: Why is an IP address considered hierarchical? What problems do hierarchical IP addresses solve and how?

IP addresses are hierarchical because they have a network portion and a host portion where the network portion of the address is used to identify the network formed by routers organized into hierarchies.

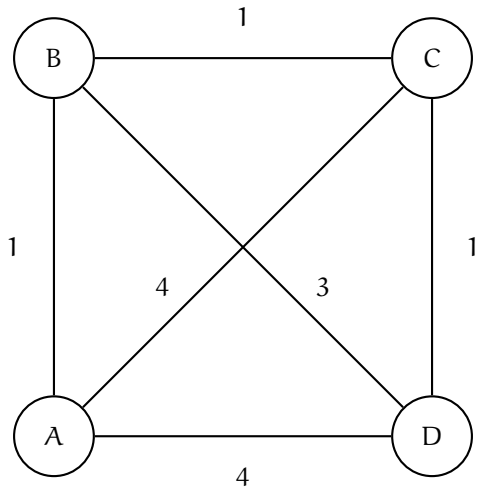
Hierarchical addresses solve the scalability issue faced by routers in storing and retrieving forwarding table entries by grouping multiple addresses into group entries or hierarchies. This allows group-based storage and retrieval mechanisms and introduces hierarchies which ensure that a single router need to be aware of all the addresses or ways of forwarding for all addresses.

Question 3.3.2: What is the purpose of MAC addresses? Why are MAC addresses not hierarchical?

MAC addresses are used to identify a physical device on the link layer and is only used for resolution of IP addresses to identify the physical device. Since the IP addresses are already hierarchical, MAC addresses reap all its benefit, additionally, when the MAC address is needed the network portion of an IP address is already resolved and only the host to MAC address resolution is done to transmit the frame over the link layer (especially in shared mediums). As such the role of MAC address is only to uniquely identify the physical device (card) over the shared medium and not for routing across networks.

Exam number: _____

Consider the network topology outlined in the graph below and suppose we have used the distance vector routing algorithm to calculate shortest paths.



Question 3.3.3: Now the cost of the edge between A and D changes to 1, would a re-run of the algorithm converge? If yes, how many iterations of the algorithm would be run before convergence and why? If no, explain your reasoning. *(Hint: it is recommended that you run the distance vector algorithm on paper to explore the answer.)*

After the edge cost between A and D changes to 1, the algorithm would terminate after 3 iterations.
* In the first iteration, D will detect its change in edge cost to A and re-compute its distance vectors.
* D will now propagate its distance vectors to all the other nodes which will recompute their distance vectors again.
* Since only the distance vector changes on A, it will propagate its updated distance vectors to all the other nodes which will again recompute the distance vectors.
* Now, since the distance vectors remain unchanged, the algorithm will now terminate.

Question 3.3.4: Why can there be a problem with count-to-infinity and when can is occur? How could it happen in the graph given above.

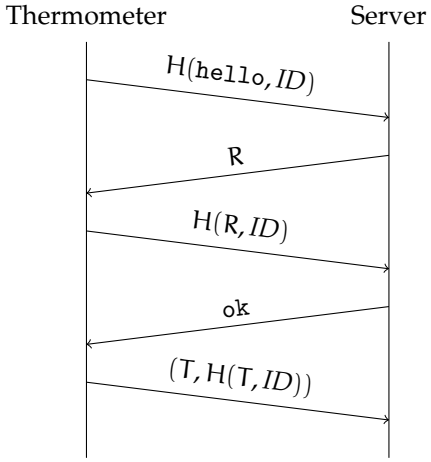
Count-to-infinity can occur when there is a sudden large increase in the cost of an edge (e.g. link failure), which is included in a shortest path. In this case a node can announce that the new goes though the node that it announces to.

In the above case, it could happen en the edge between B and C fails.

3.4 Network Security (about 10 %)

You have been hired at a company that makes home-IoT devices and have been tasked to evaluate their device that at intervals registers the room temperature and reports this to a central server. This is used as part of their smart-home setup to control automatic thermometers and ventilation.

The design is based on the following communication diagram, that communicates using TCP. H is a cryptographic hash function, ID is a unique identifier for each thermometer that only the thermometer and server knows.



- The thermometer sends a hashed hello message with the ID. The values are hashed, so the ID remains secret.
- The server finds the thermometer ID by looking up $H(\texttt{hello}, ID)$ in its database after which it replies with a nonce, R.
- The thermometer returns the hashed value of the nonce with the ID.
- The server checks that $H(R, ID)$ is correct and replies with an ok message.
- The thermometer sends the temperature, T, with the hashed ID and T.
- The server checks that $H(T, ID)$ matches with T and registers the temperature.

Question 3.4.1: Is this system vulnerable to a *replay-attack*? Explain your answer. If the system is secure, point to what makes it secure. If the system is vulnerable, explain what makes is wrong and how an attack can be performed.

No!

We are using a cryptographic hash function, which means that the attacker cannot get to know ID. A property of a cryptographic hash function is that it is vary hard to find an input from an output.

This mean that ID remain a shared secret between the thermometer and server. When the server sends a nonce and the thermometer returns a cryptographic hash of the nonce and shared secret, the server know that the sender must have had access to the shared secret. When an attacker replays, the attacker cannot generate this value correctly.

In the material the secure protocol 4.0 would here return the nonce encrypted with a shared secret, but calculating the cryptographic hash here gives the same. So it is still not best practice to use the shared secret in this way. The attacker can replay the first hello-message (as expected), but not the third.

Question 3.4.2: Is this system vulnerable to a *man-in-the-middle*? Explain your answer. If the system is secure, point to what makes it secure. If the system is vulnerable, explain what makes is wrong and how an attack can be performed.

Yes!
The attacker will never know the ID, but can listen and forward all communication between the thermometer and sender. This way is can build a complete database of temperatures and correct hash values. Thus the attacker can report any room temperature to the server and thus indirectly control all other smart devices that is adjusted based on the room temperature.
The attacker cannot calculate the third message, but can send the nonce to the thermometer to get this calculated.
A partial solution could be that the hash value is updated to include the nonce, $H(T, R, ID)$. This would mean that the hash values with equal temperatures would have different values. Thus the mitm would not have control over the reported room temperature.
But can still listen in on the temperature reports, which can be used to find information if the people are home.

Question 3.4.3: The protocol does not use encryption at all. How can you use encryption to make the protocol more secure? Explain which kind of encryption you would use and in which stage of the protocol you use it.

There are two problems:
* ID is used as the shared secret
* The temperature is send unencrypted

In the simple solution the ID can be used on server-side to look up the shared secret. Then all communication can be symmetrical encrypted using the shared secret. Thus an attacker cannot listen on the temperature.

The simple solution means that the thermometers will have a fixed shared secret, which can be lost/stolen and is very hard to update. A better approach is that the thermometers has a certificate (or a public key) for the server. The thermometers will then give a new shares secret to the server for each new communication using asymmetric encryption. After this the protocol follows the simple version above and uses symmetric encryption secure the communication.
This means that all thermometers can be updated with new certificates (public keys) if needed. (Secure updates is a different topic, we will not touch here.)