4-hour Written Exam in Computer Systems

Department of Computer Science, University of Copenhagen (DIKU)
Date: January 22, 2020

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 4 hour written exam in Computer Systems (CompSys), B1+2-2019/20. This document consists of 22 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.

- You can answer in either Danish or English.
- Remember to write your exam number on all pages.
- You do not have to hand-in this preamble.

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 of the entire set. However, your exact usage of time can differ depending on prior knowledge and skill.

Furthermore, all questions includes formatted space (lines, figures, tables, etc.) for in-line answers. Please use these as much as possible. The available spaces are intended to be large enough to include a satisfactory answer of the question; thus, full answers of the question does not necessarily use all available space.
If you find yourself in a position where you need more space or have to redo (partly) an answer to a question, continue on the backside of a paper or write on a separate sheet of paper. Ensure that the question number is included and that you in the in-lined answer space refers to it; e.g. write "*The [rest of this] answer is written on backside of/in appended page XX.*"

For the true/false and multiple-choice questions with one right answer give only one clearly marked answer. If more answers are given, it will be interpreted as incorrectly answered. Thus, if you change your answer, make sure that this shows clearly.

Exam Policy

This is an *individual*, open-book exam. You may use the course book, notes and any documents printed or stored on your computer and other device, but you may not search the Internet or communicate with others to answer the exam.

Exam number: _____

Errors and Ambiguities

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

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.

Exam number: _____

1 Machine architecture (about 33 %)

1.1 Assembler programming (about 12 %)

Consider the following program written in X86prime-assembler.

```
.entry:
    subq $8, %rsp
    movq %r11, (%rsp)
    leaq (%rdi, %rsi, 8), %rcx
    movq $0, %eax
    jmp .L5
.L6:
    addq %rdx, %rax
.L4:
    addq $8, %rdi
.L5:
    cbbe %rcx, %rdi, .L1
    movq (%rdi), %rdx
    cbg $0, %rdx, .L6
    subq %rdx, %rax
    jmp .L4
.L1:
    movq (%rsp), %r11
    addq $8, %rsp
    ret %r11
```

Assembler programming, 1.1.1: Rewrite the above X86prime-assembler program to a C program. The resulting program should not have a goto-style and minor syntactical mistakes are acceptable.

```
long abssum(long* a, long sz) {

long* limit = a + sz;

long res = 0;

while (a < limit) {

long v = *a;

if (v < 0) { // <--- This should have been v > 0

res += v;

}

else {

res -= v;

}

++a;

}

return res;

}
```

Exam number: _____

Assembler programming, 1.1.2: Argue for your choice of statements and explain shortly the functionality of the program.

Vi kan se at `%rdi` indeholder en pointer, da der "loades" værder vha den. Den kalder vi `a`. Den loadede værdi er en midlertidig variabel, vi kalder den `v`.

Afhængig af om `v` er større eller mindre end 0 lægges den enten til eller trækkes fra en variabel i `%rax`. Det er også returværdien. Den initialiseres til 0. Vi kalder den `res`. Pointeren i `a` bumpes (`++a`) ved gennemløb af løkken.

Løkken er baseret på sammenligning mellem `a` i `%rdi` og en grænseværdi i `%rcx`. Grænseværdien kalder vi `limit`. Den beregnes i den tredje instruktion (`leaq (%rdi, %rsi, 8), %rcx` - hvilket i C svarer til `limit = &arg0[arg1]` eller simplere `limit = arg0 + arg1` hvor `arg0` er typen `long*` og `arg1` er typen `long`.

Fejl i det udleverede program? Betingelsen der styrer tildelingen til `res` er sandsynligvis forkert. Det ville give mening, hvis programmet beregnede summen af de absolutte værdier i et array - men den beregner 0 - summen i stedet.

1.2 Pipeline (about 12 %)

A given machine is organised as a scalar pipeline (1 instruction per clk) with the followin 8 phases:

- `F`: Fetch I – start of instruction fetch
- `U`: Fetch II – end of instruction fetch
- `D`: Decode
- `R`: Reading of register file
- `X`: eXecute
- `M`: Memory I – start of reading form data cache
- `Y`: Memory II – end of reading form data cache
- `W`: Writeback

All instructions pass through all 8 phases in the above order. The machine have full-forwarding: results can be forwarded to the `X` phase of any of the following instructions.

- For all arithmetic instructions except multiplication the result will be ready at the end of the `X` phase.
- For multiplication the result is ready at the end of the `Y` phase.
- For reading from the data cache the result is also ready at the end of the `Y` phase.
- For writing to the data cache, the value to be written can be forwarded to the `X`, `M` or `Y` phases of the instruction that writes.

The machine uses dynamic branch prediction.

- For conditional branches that are predicted correct, there is no extra delays.
- For conditional branched that are wrongly predicted, the `F` phase of the next instruction can at the earliest only be taken after the `X` phase of the wrongly predicted instruction.

Pipeline, 1.2.1: What is the latency (latens-tid in Danish) of these instructions? Shortly state why.

Den er 1. Med fuld forwarding kan en efterfølgende instruktion starte sin `X` fase en cyklus senere.

```
add %rax,%rax,%rax FUDRXMYW
```

```
add %rax,%rax,%rax FUDRXMYW
```

Pipeline, 1.2.2: What is the latency (latens-tid in Danish) of multiplication? Shortly state why.

Den er 3. Med fuld forwarding kan der forwardes fra `Y` til `X`.

```
mul %rax,%rax,%rax FUDRXMYW
```

```
add %rax,%rax,%rax FUDRRRXMYW
```

Exam number: _____

Consider the following program fragment

```
Loop:
    movq (%r10),%r11
    cbe $0,%r11,Done
    multq %r12,%r11
    movq %r11,(%r10)
    addq $8,%r10
    jmp Loop
Done:
```

Pipeline, 1.2.3: Assume that the conditional jump (instruction number 2) is correctly predicted as not taken and that all access to the cache (both instruction and data cache) are hits.

Make an execution diagram, showing the execution of above code on this machine.

	Code	Timing
	`Loop:`	Line 4: forward mul result into Y.
1	`movq (%r10),%r11`	F U D R X M Y W
2	`cbe $0,%r11,Done`	F U D R R R X M Y W
3	`multq %r12,%r11`	F U D D D R X M Y W
4	`movq %r11,(%r10)`	F U U U D R X M Y W
5	`addq $8,%r10`	F F F U D R X M Y W
6	`jmp Loop`	F U D R X M Y W
7		
8		
9		

Pipeline, 1.2.4: Assume instead that the conditional jump (instruction number 2) is wrongly predicted.

Make the updated execution diagram, showing the execution of above code on this machine.

	Code	Timing
	`Loop:`	Line 3: delayed fetch by misprediction.Line 4: forward mul result into Y.
1	`movq (%r10),%r11`	F U D R X M Y W
2	`cbe $0,%r11,Done`	F U D R R R X M Y W
3	`multq %r12,%r11`	F U D R X M Y W
4	`movq %r11,(%r10)`	F U D R X M Y W
5	`addq $8,%r10`	F U D R X M Y W
6	`jmp Loop`	F U D R X M Y W
7		
8		
9		

Exam number: _____

Pipeline, 1.2.5: What is the cost in clock-cycles of wrongly predicting the branch?

(extended answer) Den efterfølgende instruktion hentes 5 clock cykler senere end ved korrekt forudsigelse. Så isoleret set koster den konkrete fejlforudsigelse således 5 cykler.

Men ved korrekt forudsigelse forsinkes efterfølgende instruktioner 2 cykler, fordi det betingede hop afventer i 'X' værdien fra den forudgående instruktion. Der er ikke en tilsvarende "prop" hvis der forudsiges forkert.

Den effektive omkostning for hele kodesekvensen er derfor 3 cykler.

Pipeline, 1.2.6: Describe the concept of "pipelining" and explain why it is expected that a pipelined machine can execute programs faster than a single-cycle machine.

Ideen i pipelining er at inddele et arbejde i tidsligt ca lige store dele og så udføre dem som et samlebånd.

Selvom den samlede udførelsestid er (lidt) større kan der udføres flere arbejdsopgaver samtidigt.

Hvis vi siger at et arbejde består af delene A, B og C der hver tager en tidsenhed, så vil det tage 3 tidsenheder at udføre arbejdet.

Hvis vi udfører flere stykker arbejde efter hinanden (ikke pipelinet) så får vi

```
arb1 ABC
arb2    ABC
arb3       ABC
```

Mens en pipeline gør det sådan her:

```
arb1 ABC
arb2  ABC
qarb3  ABC
```

Det ses tydeligt at pipelining udfører arbejdet hurtigere.

Exam number: _____

1.3 Data Cache (about 9 %)

In a given machine 6 bits is used for the byte-offset and 8 bits for the index.

Data Cache, 1.3.1: How large is the block size of the data cache? Specify your calculation.

Et byte offset på 6 bits betyder at block størrelsen er $1 << 6 = 64bytes$.

It is given that the data cache is a set-associative cache of totally 64 kb.

Data Cache, 1.3.2: What is the associativity of the given data cache? Specify your calculation.

Men en størrelsen på 64 kb og block størrelse på 64 bytes, så er der 1024 blokke i alt i cachen.

Hvis der bruges 8 bits på at angive index, så er der 256 "sets" i cachen.

Der er således 4 blokke i hvert "set", dvs det er en 4-vejs associativ cache.

In the following we examine a 2-way set-associative data cache with a 16 cache-blocks of 16 bytes each.

Data Cache, 1.3.3: How many bytes is the total size of the cache?

16 blocks a 16 bytes -> 256 bytes.

Exam number: _____

It is furthermore given that the data cache uses an LRU replacement policy.

Data Cache, 1.3.4: Indicate for each of the references in the following stream, if the cache access is a miss or a hit. Addresses are given in decimal notation, read tables left to right. Assume the cache is cold on entry.

Reference	Hit/Miss
0	m
8	h
16	m
256	m
4	h
260	h
1024	m

Reference	Hit/Miss
4	m
1280	m
8	h
256	m
260	h
1028	m

Data Cache, 1.3.5: Describe shortly the concept of "spatial locality of reference". State the consequence that spatial locality has on the typical cache organisation.

Rumlig referencelokalitet indebærer at hvis et program har refereret til en celle i lageret, så er det mere sandsynligt at det vil referere til celler der er tæt på.

Derfor er caches organiseret i blokke som er noget større end en enkelt reference. En reference til en lager celle vil bringe en hel blok ind i cachen. Denne blok vil indeholde nogle tæt ved liggende celler, og disse celler er der forøget sandsynlighed for at der bliver brug for, når programmet har rumlig referencelokalitet.

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. If needed use the lines to argue for your choices.

Multiple Choice Questions, 2.1.1: Which of the following exceptions often lead to the triggering instruction being re-executed?

- [x] **a)** Page fault
- [ ] **b)** Division by zero
- [ ] **c)** System call (trap)
- [x] **d)** Timer interrupt
- [ ] **e)** Invalid instruction

Multiple Choice Questions, 2.1.2: Which of the following operations are guaranteed to execute atomically in a multi-threaded program?

- [ ] **a)** `printf()`
- [ ] **b)** `write()`
- [x] **c)** `pthread_mutex_lock()`
- [ ] **d)** `x = y` (when `x` and `y` are `int`)
- [ ] **e)** `exit(0)`
- [ ] **f)** `x++` (when `x` is `sig_atomic_t`)

Multiple Choice Questions, 2.1.3: Which of the following can be straightforwardly used as an *asynchronous* mechanism for sharing non-trivial data (say, hundreds of bytes) between processes, where the writer can finish writing some data, which can then be read by the recipient later?

- [ ] **a)** `signal()`
- [x] **b)** Sharing pages with `mmap()`
- [ ] **c)** A pipe
- [x] **d)** File on disk
- [ ] **e)** Thread pool
- [ ] **f)** A socket

Exam number: _____

2.2 Short Questions (about 12 %)

Short Questions, 2.2.1: Consider the following program. Will the `assert()` ever fail? If so, why, and how could you fix the program? If not, why not?

```
#include <pthread.h>
#include <assert.h>

int counter = 0;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void* thread(void* unused) {
  while (1) {
    pthread_mutex_lock(&mutex);
    if (counter == 0) {
      pthread_cond_wait(&cond, &mutex);
    }
    assert(counter > 0);
    counter--;
    pthread_mutex_unlock(&mutex);
  }
}

int main() {
  int n = 10;
  pthread_t tids[n];
  for (int i = 0; i < n; i++) {
    pthread_create(&tids[i], NULL, thread, NULL);
  }
  while (1) {
    pthread_mutex_lock(&mutex);
    counter++;
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&mutex);
  }
}
```

The program is at risk of *spurious wakeup*, because a single call to `pthread_cond_signal()` may wake more than one thread (and a thread may also return from `pthread_cond_wait()` for other reasons). The simplest solution is to turn the `if` into a `while` loop.

Exam number: _____

Short Questions, 2.2.2: Consider a system with the following properties:

- Memory is byte-addressed.
- Virtual addresses are 15 bits wide.
- Physical addresses are 14 bits wide.
- The page size is 32 bytes.
- The TLB is 3-way set associative with four sets and 12 total entries. Its initial contents are:

Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	07	0	09	0D	0	11	51	0
1	07	2D	1	12	51	1	00	00	1
2	0A	00	0	09	01	1	10	34	1
3	10	A1	0	03	0D	0	10	A0	1

- The page table contains 12 PTEs:

VPN	PPN	Valid	VPN	PPN	Valid	VPN	PPN	Valid	VPN	PPN	Valid
12	12	1	01	02	1	02	01	1	31	33	1
41	01	1	09	17	1	02	33	0	0B	0D	0
00	00	1	10	21	0	13	32	0	21	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.*

Virtual address: `0x0019`

1. Bits of virtual address

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1	1	0	0	1

2. Address translation

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

3. Bits of phys. (if any)

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

Exam number: _____

Virtual address: `0x0853`

1. Bits of virtual address

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	0	1	0	1	0	0	1	1

2. Address translation

Parameter	Value
VPN	42
TLB index	2
TLB tag	10
TLB hit? (Y/N)	Y
Page fault? (Y/N)	N
PPN	34

3. Bits of phys. (if any)

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

Virtual address: `0x1337`

1. Bits of virtual address

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	1	1	0	0	1	1	0	1	1	1

2. Address translation

Parameter	Value
VPN	99
TLB index	1
TLB tag	26
TLB hit? (Y/N)	N
Page fault? (Y/N)	Y
PPN	

3. Bits of phys. (if any)

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

2.3 Long Questions (about 15 %)

Long Questions, 2.3.1: Consider the problem of compiling a potentially large number of `.c` files to `.o` files. Since each such compilation can be done independently of all others, we can use parallelism to reduce the total run time—this is in fact how `make` works. Consider the following two ways parallelising this task on a system with 8 CPU cores:

1. Immediately launch one compiler process for every `.c` file.
2. Split the list of `.c` files into 8 sublists of equal liength, then for each of the 8 sublists run a single process (so 8 processes in total) that *sequentially* traverses the sublist and compiles each file in turn.

What are the disadvantages of the two approaches? Describe the workloads where each approach performs well or poorly. Can you come up with a third approach that avoids some of these disadvantages?

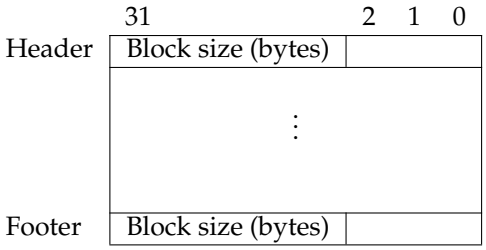
Approach 1 suffers from potentially significant overhead if the number of `.c` files is large. Having much more than 8 processes running is pure overhead, without exploiting the machine any better. Approach 2 is at risk of poor load balancing, as there is no guarantee that each sublist takes the same amount of time to compile. A better approach is to use some form of work queue, where 8 worker processes or threads are launched, each of which retrieve work from a shared pool of `.c` files that have yet to be compiled.

Exam number: _____

Long Questions, 2.3.2:

The following problem concerns dynamic storage allocation.

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. 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.

On the following page, five helper functions are defined to facilitate the implementation of `free(void *p)`. The functionality of each function is explained in the comment above the function definition. Fill in the body of each helper function to implement the corresponding functionality correctly.

Exam number: _____

```
// Given a pointer p to an allocated block, i.e., p is a
// pointer returned by some previous malloc()/realloc() call;
// returns the pointer to the header of the block
void *header(void *p)
{



  return ((char*)p-4)
}
```

```
// Given a pointer to a valid block header or footer,
// returns the size of the block.
int size(void *hp)
{



  return (*(int*)hp)&( 3)
}
```

```
// Given a pointer p to an allocated block, i.e. p is
// a pointer returned by some previous malloc()/realloc() call;
// returns the pointer to the footer of the block.
void *footer(void *p)
{



  return (char*)p+size(header(p))-8
}
```

```
// Given a pointer to a valid block header or footer,
// returns the usage of the currect block,
// 1 for allocated, 0 for free.
int allocated(void *hp)
{



  return (*(int*)hp)&1
}
```

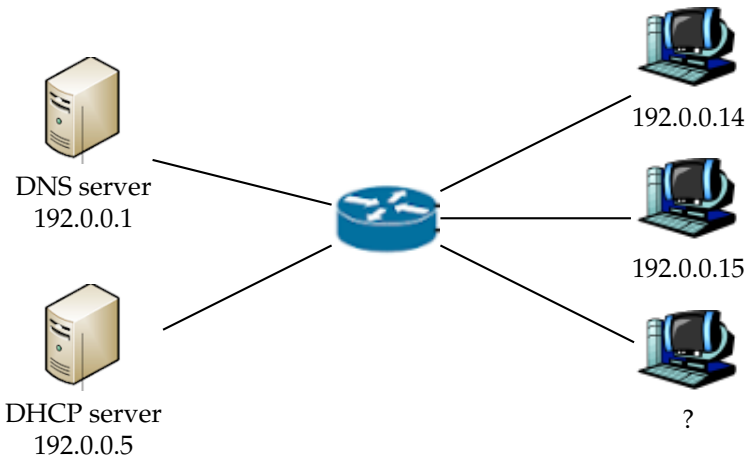
```
// Given a pointer to a valid block header,
// returns the pointer to the header of previous block in memory.
void *prev(void *hp)
{



  return (char*)hp - size((char*)hp-4)
}
```

Exam number: _____

3 Computer Networks (about 34 %)

3.1 IP retrieval (about 8 %)

Consider the following network topology, where a new host (indicated with a questionmark) connects to a LAN.



IP retrieval, 3.1.1: Which protocol is central for the host to get a new IP? What role does the MAC address play?

The ARP protocol and the DHCP protocol is accepted answers.

MAC adresses are used as unique identifiers for hosts. MAC addresses are intended to be unique in a global scale, but is practice it is only needed for each LAN.

Exam number: _____

IP retrieval, 3.1.2: Describe which messages are exchanged. Specify which IPs are send to and from.

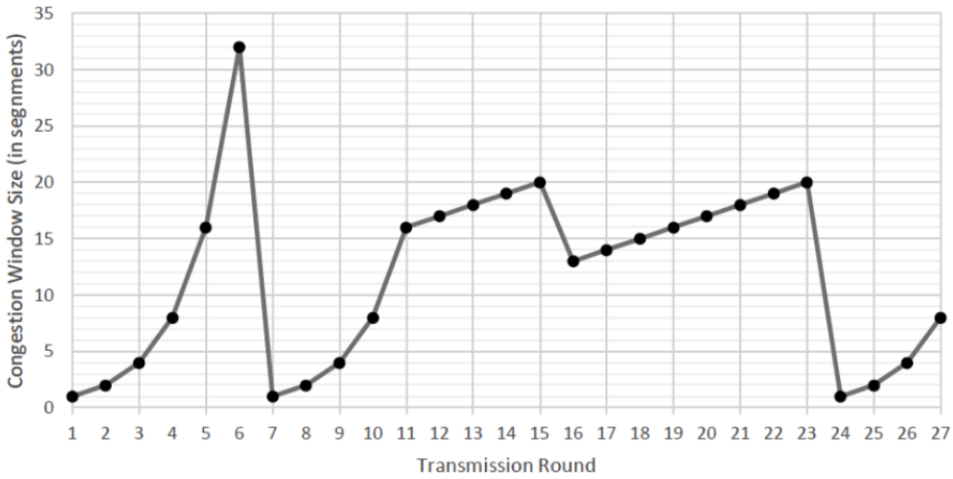
For ARP: New host (without IP) first broadcasts a discover message.

DHCP server (IP 192.0.0.5) responds by broadcasting a possible new IP.

New host responds by broadcasting its choice of IP.

DHCP server responds by broadcasting an acknowledgement message.

3.2 Reliable Data Transfer (about 8 %)



Reliable Data Transfer, 3.2.1: Consider the above graph showing the window size of a TCP connection over time (given in rounds). Explain the behaviour from round 6 to round 14. Include what is happening with the network connection and which TCP effects are used at different times.

At round 6 the connection is lost; that is package time-out and not collision. The connection then goes into slow-start (double the window-size), which continues until half the window-size at time-out (round 6) is reached. This happens at round 11. After this the connection continues by incrementing the window size.

(There is more space on next page.)

Reliable Data Transfer, 3.2.2: What is the importance of Sequence Number and Acknowledgement Number as part is the TCP header? How are these determined?

Sequence Number is a 32-bit field which indicates the amount of data that is sent during a TCP session. By sequence number, the sender can be assured that the receiver received the data because the receiver uses this sequence number as the acknowledgement number in the next segment it sends to acknowledge the received data. When the TCP session starts, the initial sequence number can be any number in the range 0-4,294,967,295.

Acknowledgement number is used to acknowledge the received data and is equal to the received sequence number plus 1.

Exam number: _____

3.3 Network Forwarding (about 4 %)

Consider a datagram networks using 32-bit host addresses. Suppose a router has four links, numbered 0 through 3, and packets are to be forwarded to the link interface as follows:

```
Destination Address Range                          Link interface

    11100000 00000000 00000000 00000000
            ..............                         0
    11100000 11111111 11111111 11111111

    11100001 00000000 00000000 00000000
             ............                          1
    11100001 00000000 11111111 11111111

    11100001 00000001 00000000 00000000
             ...........                           2
    11100001 11111111 11111111 11111111

            otherwise                              3
```

Network Forwarding, 3.3.1: Give a forwarding table that has four entries, uses longest-prefix matching, and forwards packets to the correct link interfaces.

224.0.0.0/8	0
225.0.0.0/16	1
225.0.0.0/8	2
deafult	3

Network Forwarding, 3.3.2: Describe how your forwarding table determines the appropriate link interface for datagrams with the following destination addresses.

```
11001000 10010001 01010001 01010101
11100001 00000000 11000011 00111100
11100001 10000000 00010001 01110111
```

1. IP: 200.X.X.X; Maps to 3 as first part of IP is different from both 224 and 225 (thus default).

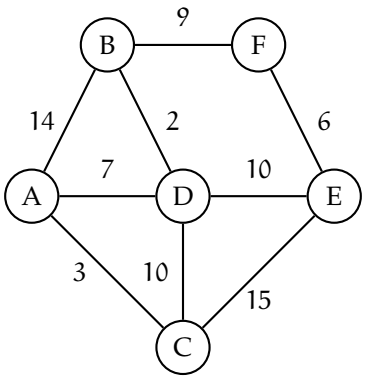
2. IP: 225.0.X.X; Maps to 1 as it first the first 16 bit of second entry.

3. IP: 225.128.X.X; maps to 2 as first part is 225 but second is different from 0.

Exam number: _____

3.4 Network Routing (about 4 %)

Consider the network topology outlined in the graph below



Network Routing, 3.4.1: Apply the link state routing algorithm and compute the forwarding table on node A by filling out the following tables.

Steps of the algorithm:

Step	N′	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	14,A	3,A	7,A	∞	∞
1	AC	14,A	3,A	7,A	18,C	∞
2	ACD	9,D	3,A	7,A	17,D	∞
3	ACDB	9,D	3,A	7,A	17,D	18,B
4	ACDBE	9,D	3,A	7,A	17,D	18,B
5						

Forwarding table on node A:

Destination node	Edge
B	(A,D)
C	(A,C)
D	(A,D)
E	(A,D)
F	(A,D)
G	

Exam number: _____

3.5 Network Security (about 10 %)

When setting up an HTTPS connection between two hosts, the hosts start by using public-key (asymmetric) encryption to share a private session-key used for symmetric encryption.

Network Security, 3.5.1: Describe why both forms of encryption is used. What effect does this have on efficiency and security?

Asymmetric and symmetric encryption have different properties when considering execution speed and security. Symmetric encryption is much faster than asymmetric encryption, but requires a shared secret. Asymmetric encryption do not need a shared secret. Thus asymmetric encryption is used to share a secret key, that is then used by encrypt the stream with symmetric encryption.

Exam number: _____

Network Security, 3.5.2: What does it mean that the symmetric key is a session-key? Why is this important?

That it is a session-key, means that it is only used for a specific session. A new connection should use a new randomly generated key. Key reuse can result in connection being vulnerable to replay attacks or brute-force if it limits the key space.

Network Security, 3.5.3: What is a nonce, when is it used in the HTTPS protocol and to what effect?

A nonce is a one-time use value; hosts should never (within reasonable time) use the same nonce. It is used with setting up the connection and is a defence against replay attacks.