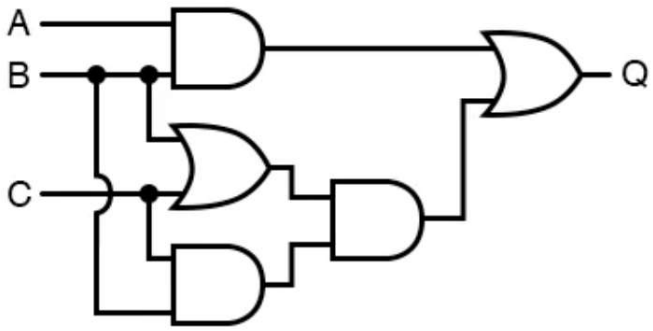
Operating Systems & C : Mock Exam

Willard Rafnsson & Niclas Hedam, IT University of Copenhagen, Fall 2024

This is the hand-out for the mock exam for Operating Systems and C, Fall 2024.
This mock exam is a modified version of the Operating System and C, Fall 2023 exam.
Were this the real exam, then your course grade would be based on your answers to the questions below.

logic (6 points)

Here is a logic diagram.



(a)

Which of the following Boolean expression are equivalent with Q (the output of the circuit)?
Pick all that apply.

- [x] A&B | B&C & B&C
- [ ] A&B&C
- [ ] A&B | B&C&(B|C)
- [x] B&(A|C)

(b)

Which of the following are entries in the truth table for this circuit?
Pick all that apply.

- [x] A=1, B=0, C=1, Q=0
- [x] A=1, B=1, C=0, Q=1
- [x] A=0, B=1, C=1, Q=1
- [ ] A=1, B=1, C=0, Q=0

assembly semantics (9 points)

(a)

Consider the following Y86-64 instruction.

```
CALL 0x720bef28
```

Show how this instruction executes on SEQ hardware, by filling in the correct values in the computation stages below.
At the time this instruction is executed, `%rsp` has value `0x5568`, and `PC` has value `0x720bee10`.
The values you fill into these fields should be in hexadecimal notation. Please leave out the leading `0x`.

```
Fetch       icode [  8  ] : ifun [ 0   ] <- M_1 [     ]720bee10
            valC <- M_8 [     ]720bee11
            valP <- [     ]720bee1a
Decode      valB <- [ 5568 ] <- R [     ]
Execute     valE <- [     ] <- [ valB ] + (-8)
Memory      M_8 [  valE] <- [ valP ]
Writeback   R [ RSP ] <- [ valE ]
PC update   PC <- [ valC ]
```

(b)

Consider the following AVX256 instruction.

```
VPUNPCKLBW ymm0, ymm1, ymm2
```
X86ASM

(this instruction behaves like the short-vector unpack-low that you almost-certainly used in prflab, except this instruction unpacks byte-vectors.)

Which value will this instruction write into the last two bytes of `ymm0` ?

At the time this instruction is executed,

```
ymm1 = 0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ymm2 = 0xBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
```

The value you provide should be in hexadecimal notation. Please leave out the leading `0x` .

assembly program execution (9 points)

Here is a short program written in Y86-64.

```
   irmovq $D, %rdi      # instruction nr. 0
   irmovq $S, %rsi      # instruction nr. 1
   irmovq two, %rax     # instruction nr. 2
   mrmovq 0(%rax), %rax # instruction nr. 3     the value of rax is set to 2
main:
   addq %rdi, %rax      # instruction nr. 4
   addq %rax, %rax      # instruction nr. 5
   subq %rdi, %rsi      # instruction nr. 6     Adds d to rax, rax+rax, s-d, if s = 0 halt, else repeat.
   je done              # instruction nr. 7
   jmp main             # instruction nr. 8
done:
   irmovq $0, %rdi      # instruction nr. 9
   halt                 # instruction nr. A
two:
   .quad 0x000000000002
```
X86ASM

(a)

Indicate, for each of the following values in place of D and S, the value of register %rax after program execution.

Assume that the program is running on SEQ hardware.

- 1) D = 1, S = 3 — 30
- 2) D = 2, S = 6 — 44
- 3) D = 2, S = 3 — Undefined (Infinite loop?)

(b)

Indicate, for each of the following clock cycles during program execution, which program instruction is in which stage of the pipeline (i.e. in which of the F, D, E, M, W pipeline registers). In each field, indicate the instruction number of the instruction there, or blank for pipeline bubble.

Assume D = 1, S = 3, and that the program is running on PIPE hardware (incl. forwarding- and pipeline-control-logic).

Hint: We have pre-filled the stages for the first four cycles.

```
    F   D   E   M   W
1) [0] [ ] [ ] [ ] [ ]
2) [1] [0] [ ] [ ] [ ]
3) [2] [1] [0] [ ] [ ]
4) [3] [2] [1] [0] [ ]
5) [4] [3] [2] [1] [0]
6) [4] [3] [ ] [2] [1]
7) [5] [4] [3] [ ] [2]
8) [6] [ 5] [4] [3] [ ]
9) [7] [6] [5] [4] [3]
A) [9] [7] [6] [5] [4]
B) [A] [9] [7] [6] [5]
C) [8] [ ] [ ] [7] [6]
D) [4] [8] [ ] [ ] [7]
E) [5] [4] [8] [ ] [ ]
F) [6] [5] [4] [8] [ ]
```

instruction 3 needs a value instruction 2 is going to change, therefore a hazard. Bubble to avoid

Branch is created by the je instruction

Predicts that it jumps, and anticipates instruction 9 and A

After executing 7, we know that the prediction was wrong, and the predicted instructions are replaced with bubbles from here (A and 9)

locality (6 points)

Consider `naive_rotate` from **prflab**:

```c
#define RIDX(i,j,n) ((i)*(n)+(j))
void naive_rotate(int dim, pixel *src, pixel *dst)
{
    int i, j;

    for (i = 0; i < dim; i++)
        for (j = 0; j < dim; j++)
            dst[RIDX(dim-1-j, i, dim)] = src[RIDX(i, j, dim)];
}
```

(a)

What kind of locality of data does this program exhibit whilst referencing the following?

- elements of `src` spatial, iterates of rows
- elements of `dst` neither Why is this neither, but the inner loop body is both?
- `i` temporal

For each, choose one of: spatial, temporal, both, or neither.

(b)

What kind of locality of instructions does this program exhibit whilst references the following?

- `i=0` instruction neither
- `j++` instruction temporal
- `inner loop body` both

For each, choose one of: spatial, temporal, both, or neither.

caching (6 points)

(a)

Why does thrashing happen?

Pick all that apply.

- High miss penalty
- Cache incoherence
- Cache miss ●
- Capacity miss ●

(b)

A cache coherence protocol ensures that...

Pick one that applies.

- ... a read from an address gives you its most up-to-date value. ●

- ... data in the cache can be accessed in a coherent way.
- ... contents of the cache is consistent with what is contained in memory.

memory (6 points)

(a)

What is a segmentation fault?
Pick one that applies.

- The kernel failed to segment memory into pages in response to a request made by the process.
- The data structures in the heap of the process are no longer aligned in memory.
- The process attempted to access a region of memory which it is not privileged to access.

(b)

What do we gain by having a virtual memory mapped region for shared libraries?
Pick all that apply.

- Backwards compatibility (it's a legacy feature).
- Simplifies access to shared libraries.
- Programs will run faster. locality
- Programs will consume less memory.

virtual memory (8 points)

Consider the following memory system.

- page size is 256 bytes.
- virtual addresses are 14 bits long.
- physical addresses are 12 bits long.
- there is a 3-way TLB with 4 sets.
- memory is byte-addressable.

(a)

How many bits are used for the following system parameters?

```
VPO [8 ]
PPO [ 8]
VPN [6 ]
PPN [ 4 ]
```

virtual address size - page size --> 14-8=6

physical address size - page size --> 12-8=4

(b)

Suppose the page table and TLB contain the following data.

VPN	PPN	Valid
10	7	1
11	6	1
12	1	1
13	3	1
14	2	1
15	4	0
16	5	0
17	0	0

VPN	PPN	Valid
28	b	1
29	8	0
2a	f	1
2b	d	0
2c	e	1
2d	a	0
2e	9	1
2f	c	0

Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	4	7	1	5	2	1	a	b	1
1	4	6	1	5	4	0	a	8	0
2	4	1	1	5	5	0	a	f	1
3	4	3	1	5	0	0	a	d	0

Show how the virtual address 0x11f2 is translated to a physical address. Provide values in hexademical notation (leaving out the leading `0x`), except where a yes/no answer is requested (indicated by `Y/N`).

address in binary: 01 0001 1111 0010

```
VPN          [ 11   ]01 0001 --> 0x11        VPN beregnede vi til at være 6 bits, og den er de højeste bits, så bits 14 - 8
TLB index    [ 1    ]sidste to bits af VPN, konverteret til hex
TLB tag      [ 4    ] resterende bits af VPN, konverteret til hex
TLB hit?     (Y/N) Y
Page fault?  (Y/N) N
PPN          [ 6    ]
```

i/o (6 points)

(a)

When a program reads from a file on disk, which of the following events occurs as a result?

Pick all that apply.

- The program executes the `IN` instruction (to input from I/O port).
- The program executes the `INT` (aka. `SYSCALL`) instruction. ●
- The kernel performs a context switch.
- The disk transfers data to a buffer in the CPU. ●
- The disk interrupts the CPU when the transfer is complete. ●

(b)

Why is it important to close file descriptors once you are done using them?

Pick one that applies.

- Not doing so introduces a memory leak. ●
- Otherwise data may still reside in buffers which haven't been flushed.
- The entity at the other end will not know that the session is complete otherwise.

threads (6 points)

Consider the following C source file `t.c`.

C

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

int n = 3;
int k = 0;
int s = 0;

void *incr(void* arg) {
  int tk = ++k;
  int ts = s;
  int i;
  for (i = 0; i < n; i++) {
    ts += tk;
  }
  usleep( ( (float)rand() / RAND_MAX ) * 10 );
  s = ts;
}

int main() {
  pthread_t* tids = malloc( n * sizeof(pthread_t) );
  int i;
  srand(time(NULL));
  for (i = 0; i < n; i++) {
    pthread_create( &(tids[i]), NULL, incr, (void*)NULL );
  }
  for (i = 0; i < n; i++) {
    pthread_join(tids[i], NULL);
  }
  printf("%d\n", s);
}
```

It compiles without error with the following command:

SHELL

```
gcc -pthread -o t t.c
```

(a)

Upon executing this program (`./t`), which of the following are possible outputs of the program?
Pick all that apply.

- 30
- 15
- 3
- 9
- 12

(b)

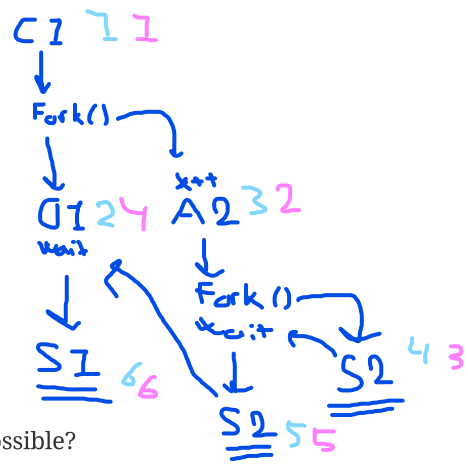
Which of the following variables would need to be protected (e.g. by a `mutex`) to prevent race conditions?
Pick all that apply.

- `n`
- `k`
- `s`
- `tk`
- The `mutex` itself.

processes (3 points)

Consider the following Linux program snippet.

```c
int main(){
  int status;
  int x = 1;
  printf("C%d\n",x);
  if(fork()==0){
    x++;
    printf("A%d\n",x);
    if(fork()>0){
      wait(&status);
    }
  }
  else {
    printf("O%d\n",x);
    wait(&status);
  }
  printf("S%d\n",x);
}
```



Which of the following four outputs are possible?
Assume that `printf` is atomic, and that no error occurs during execution.
Pick all that apply.

```
C1
O1
A2
S2
S2
S1
```

```
C1
A2
O2
S2
```

```
C1
A2
O1
S1
```

```
C1
A2
S2
01
S2
S1
```

monitors (8 points)

(a)

Consider the following Linux system.

```
$ ls -l exam-solution.pdf
-rw-rw----+ 1 amy student 102400 Dec 25 23:25 exam-solution.pdf

$ getfacl exam-solution.pdf
# file: exam-solution.pdf
# owner: amy
# group: student
user::rw-
group::---
group:osc24adm:rw-
group:faculty:r--
mask::rw-
other::---

$ groups amy bob clo dan
amy : amy faculty
bob : bob student
clo : clo osc24adm
dan : dan faculty

$ pwd
/
```

```
$ getfacl example.txt
# file: example.txt
# owner: user1
# group: group1
user::rw-
user:user2:rw-    # ACL entry for a specific user (user2)
group::r--        # Permissions for the group
mask::rw-         # Maximum permissions for additional ACL entries
other::r--        # Permissions for others
```

Which of the following users have write permissions to `exam-solution.pdf` ?

- amy
- bob
- clo
- dan

(b)

Consider a Mandatory Protection System (MPS) based on labels drawn from the following lattice of confidentiality levels (security clearance levels used by the US Department of Defense).

```
T    (top secret)
|
S    (secret)
|
C    (confidential)
```

(i.e. C is less confidential than S, which is less confidential than T).
Fill out the fields of the below access matrix for this MPS. Each field can either be blank, `r` , `w` , or `rw` .

	T	S	C
T	rw	r	r
S		rw	r
C			rw

solution siger at der skal være w på de resterended tomme felter. Ved ikke hvorfor

cuda (9 points)

Consider the following code, which increments the entries of an `int` array of 640 elements.

```c
for ( int i = 0; i < 640 ; i++ )
  data[i]++;
```

The programmer wishes to speed up this part of their program, by utilizing the GPU they have on their system.
They write a CUDA program with a kernel function, `kernel` , defined as follows.

```c
int i = threadIdx.x + blockIdx.x * blockDim.x;
data[i]++;
```

Their GPU has 16 SMs with 32 cores each. It takes the original program ~4.5 microseconds to execute. It takes the kernel ~12 nanoseconds to execute on the GPU. The data transfer rate is 1 GB/s.
Which of the following statements about this kernel implementation are true?
Pick all that apply.

- running `kernel<< 16, 40 >>` is faster than running `kernel<< 10, 64 >>`.
- the kernel implementation minimizes the cache miss rate of the SMs.
- the kernel implementation indexes the data array out of bounds.
- the CUDA program finishes faster than the original CPU program.

man vil gerne have threadcount % 32 = 0

Måske bruges der for mange threads, i forhold til opgavens størrelse

miscellaneous (18 points)

(a)

Why does an optimizing C compiler not do code motion on function calls?
Pick one that applies.

- Not all functions are referrentially transparent.
- Statements in a function body cannot substitute a function call expression.
- It is too difficult to detect function calls in source code.

(b)

Suppose process P1 allocates a region on the heap using `malloc`, and gets back an address 0x00000020. Process P1 then sets the contents of the region to all 1s. Suppose that process P2 then attempts to read from 0x00000020. What happens?
Pick all plausible outcomes.

- P2 reads in the value 0x00000020 had before P1 overwrote it with 1s (P2 does not have access to P1's address space).
- P2 reads in the the all 1s that P1 wrote to 0x00000020.
- P2 reads in the contents of the physical page that 0x00000020 maps to in its own address space.
- P2 experiences a segmentation fault.

(c)

On mainstream computer systems, when one process is currently running, how do other processes get to run?
Pick one that applies.

- The running process relinquishes control of the processor core once it is ready to do so.
- An interrupt forces the processor to context-switch to the kernel at regular intervals.
- The kernel process suspends the running process, and decides which process goes next.

(d)

How can a program, running on a core, ensure that an otherwise non-atomic instruction gets executed atomically?
Pick one that applies.

- By pausing execution of all other cores.
- By utilizing a special instruction modifier that locks the memory bus.
- By asking the kernel to perform the instruction on its behalf.

(e)

What is the benefit of the "Everything is a file" idea implemented in Unix systems?
Pick one that applies.

- Can easily back up and restore all kinds of data.
- Facilitates adding, modifying, and deleting, devices on your system.
- A unified API for doing I/O on all kinds of resources.

(f)

Which namespaces get created when the following command is run?

```
$ unshare -mipfnuUr /bin/bash
```

pick all that apply.

- mount 
- fork
- network 
- root

Last updated 2024-12-27 14:38:38 +0100