UNIVERSITETET I BERGEN

KANDIDAT

149

PRØVE

INF214 0 Multiprogrammering

Emnekode	INF214
Vurderingsform	Skriftlig eksamen
Starttid	21.11.2023 08:00
Sluttid	21.11.2023 11:00
Sensurfrist	--
PDF opprettet	03.05.2024 11:03

Exam structure

Oppgave	Tittel	Status	Poeng	Oppgavetype
i	Exam structure			Informasjon eller ressurser

Theoretical questions

Oppgave	Tittel	Status	Poeng	Oppgavetype
1	Task 1	Besvart	7/7	Langsvar
2	Task 2	Riktig	4/4	Flervalg (flere svar)
3	Task 3	Besvart	7.5/10	Langsvar

Semaphores

Oppgave	Tittel	Status	Poeng	Oppgavetype
4	Task 4	Besvart	10/25	Programmering

Monitors

Oppgave	Tittel	Status	Poeng	Oppgavetype
5	Task 5	Delvis riktig	10/11	Nedtrekk
6	Task 6	Delvis riktig	6/8	Nedtrekk

"Modern" CSP (1985)

Oppgave	Tittel	Status	Poeng	Oppgavetype
7	Task 7	Besvart	4.5/5	Programmering

JavaScript generators and iterators

Oppgave	Tittel	Status	Poeng	Oppgavetype
8	Task 8	Riktig	5/5	Sammensatt

JavaScript promises

Oppgave	Tittel	Status	Poeng	Oppgavetype
9	Task 9	Ubesvart	10/10	Langsvar
10	Task 10	Delvis riktig	7.650000095367432/9	Fyll inn tekst
i	Cheat sheet about the semantics of promises			Informasjon eller ressurser
11	Task 11.1	Riktig	1/1	Flervalg
12	Task 11.2	Riktig	1/1	Flervalg
13	Task 11.3	Riktig	1/1	Flervalg
14	Task 11.4	Riktig	1/1	Flervalg
15	Task 12	Delvis riktig	1/2	Paring

1 Task 1

Consider the following program:

```
int x = 1;
int y = 3;
co
  < x = x * y; >
||
  x = x + y;
oc
```

Answer the following questions:

- **Does the program meet the requirements of the At-Most-Once-Property? Explain your answer.**
- **What are the possible final values of x and y? Explain your answer.**

Fill in your answer here

it does not fullfill the at-most-once property because the atomic statement can run between the reading and writing of x in the second process.
the second process has a critical reference (x) that is read by another process

the possible values:
if the atomic statement runs first:
x = 6
y = 3

if the atomic statement runs last:
x = 12
y = 3

if the atomic statement runs in between the reading and writing in the second statement:
x = 4
y = 3
because the second process overwrites the first atomic process. it will therefore be the same as ignoring the first process.

Ord: 106

Maks poeng: 7

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

0 8 4 4 3 9 5

2 Task 2

Consider the following program written in the AWAIT language:

```
bool should_continue = true;
bool can_proceed = false;
co
    while (should_continue) {
      can_proceed = true;
      can_proceed = false;
    }
||
    <await (can_proceed) should_continue = false;>
oc
```

Which of the following statements hold for this program?

Select one or more alternatives:

- [x] If the scheduling policy is strongly fair, this program will eventually terminate ✔
- [ ] If the scheduling policy is strongly fair, this program will never terminate
- [x] If the scheduling policy is weakly fair, this program might not terminate ✔
- [ ] If the scheduling policy is weakly fair, this program will never terminate

Maks poeng: 4

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

4 8 3 2 9 3 4

3 Task 3

Describe the difference between **synchronous** and **asynchronous** message passing.

Fill in your answer here

with synchronous message passing, the message sender has to wait for a response from the message reciver before they can continue execution. so they are blocked by the messege sending operation.

with asynchronous message passing, the message sender is not blocked and may continue execution

Ord: 45

Maks poeng: 10

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

2 5 0 2 7 7 9

4 Task 4

Three persons P_1, P_2, and P_3 were invited by their friend *F* to make some smørbrød (sandwich made of bread, eggs, and tomato) together.

To make a portion of smørbrød, three ingredients are needed: a slice of bread, a slice of tomato, and a slice of an egg.

Each of these persons P_1, P_2, P_3 has only one type of each of the ingredients:

- person P_1 has slices of bread;
- person P_2 has slices of tomato;
- person P_3 has slices of egg.

We assume that persons P_1, P_2, and P_3 each has an unlimited supply of these ingredients (i.e., slices of bread, slices of tomato, slices of egg), respectively. Their friend *F*, who invited them, also has an unlimited supply of **all** the ingredients.

Here is what happens: the host *F* puts two random ingredients on the table. Then the invited person who has the third ingredient picks up these other two ingredients, and makes the smørbrød (i.e., takes a slice of bread, puts on it a slice of tomato, and puts on top a slice of egg), and then eats the smørbrød. The host of the party *F* waits for that person to finish. This "cycle" of is then infinitely repeated.

Write code in the AWAIT language that simulates this situation.

- **Represent the persons P_1, P_2, P_3, F as processes.**
- **You must use SPLIT BINARY SEMAPHORE for synchronization.**
- **Make sure that your solution avoids deadlock.**
- **EXPLAIN very briefly the advantages of using the split binary semaphore.**

Fill in your answer here

```

sem make = 1
sem eat = 0

bool bread = false
bool tomato = false
bool egg = false

co

// F process
while (true):
    p(make)
    add_ingredients() // function that sets two random ingredients to true.
    v(eat)

||

// p1 with beard
while (true):
    if (tomato = true and egg = true):
        p(eat)
        take_ingredients() // set ingredients to false.
        make_and_eat()
        v(make)

||

// p2 with tomato
```

```
while (true):
    if (bread = true and egg = true):
        p(eat)
        take_ingredients() // set ingredients to false.
        make_and_eat()
        v(make)

||

// p3 with egg
while (true):
    if (bread = true and egg = tomato):
        p(eat)
        take_ingredients() // set ingredients to false.
        make_and_eat()
        v(make)

oc




advantages of split binary semaphores in this example:
- i can know that the F process will not run at the same
  time as the P processes.
- it prevents race conditions for the ingredient booleans.
- it garuantees mutual exclution between the P processes.

```

Maks poeng: 25

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

8 0 7 8 4 7 1

5 Task 5

Consider the following variant of the Readers/Writers problem: reader processes query a database and writer processes examine and modify it. Readers may access the database concurrently, but writers require exclusive access. Although the database is shared, we cannot encapsulate it by a monitor, because readers could not then access it concurrently since all code within a monitor executes with mutual exclusion. Instead, we use a monitor merely to arbitrate access to the database. The database itself is global to the readers and writers.

The *arbitration monitor* grants permission to access the database. To do so, it requires that processes inform it when they want access and when they have finished. There are two kinds of processes and two actions per process, so the monitor has four procedures: **request_read**, **request_write**, **release_read**, **release_write**. These procedures are used in the obvious ways. For example, a reader calls **request_read** before reading the database and calls **release_read** after reading the database.

To synchronize access to the database, we need to record how many processes are reading and how many processes are writing. In the implementation below, **nr** is the number of readers, and **nw** is the number of writers; both of them are initially 0. Each variable is incremented in the appropriate request procedure and decremented in the appropriate release procedure.

A software developer has started on the implementation of this monitor. Your task is to fill in the missing parts.

Your solution does not need to arbitrate between readers and writers.

monitor RW_Controller {
  int nr = 0;
  int nw = 0;
  cond OK_to_write; *// signalled when nr == 0 and nw == 0*
  cond OK_to_read; *// signalled when nw == 0*

  procedure request_read() {

while (nw>0) ✔ (while (nw<0), while (nr==0), while (nw==0), while (nr<0), while (nr>0 || nw>0), while (nw>0), if (nw==0), if (nr==0), if (nr<0 || nw<0), if (nw<0), while (nr>0), /* conditional or loop is not needed here */, while (nr<0 || nw<0), if (nr>0), if (nr<0), if (nr>0 || nw>0), if (nw>0)) {

wait(OK_to_read) ✔ (signal_all(OK_to_write), signal(OK_to_write), wait(OK_to_read), // nothing is needed here, wait(OK_to_write), signal_all(OK_to_read), signal(OK_to_read))
    };
    nr = nr + 1;

signal_all(OK_to_read) ✖ (signal(OK_to_write), signal(OK_to_read), wait(OK_to_read), wait(OK_to_write), // nothing is needed here, signal_all(OK_to_write), signal_all(OK_to_read))
  }

procedure request_write() {

while (nr>0 || nw>0) (/* conditional or loop is not needed here */, while (nw<0), while (nw==0), while (nw>0), if (nw>0), while (nr<0 || nw<0), while (nr==0), if (nr>0 || nw>0), if (nw==0), if (nw<0), while (nr>0 || nw>0), while (nr<0), if (nr>0), if (nr==0), while (nr>0), if (nr<0 || nw<0), if (nr<0))

wait(OK_to_write) (// nothing is needed here, signal_all(OK_to_read), wait(OK_to_read), signal_all(OK_to_write), signal(OK_to_write), wait(OK_to_write), signal(OK_to_read))

}

nw = nw + 1;

// nothing is needed here (signal_all(OK_to_read), signal(OK_to_write), signal(OK_to_read), signal_all(OK_to_write), wait(OK_to_write), // nothing is needed here, wait(OK_to_read))

}

procedure release_read() {

// nothing is needed here (wait(OK_to_write), signal(OK_to_write), signal_all(OK_to_read), signal_all(OK_to_write), signal(OK_to_read), wait(OK_to_read), // nothing is needed here)

nr = nr - 1;

if (nr==0) (while (nw<0), /* conditional or loop is not needed here */, if (nr>0 || nw>0), while (nr==0), if (nw==0), if (nr<0 || nw<0), while (nr<0 || nw<0), while (nw==0), if (nw<0), while (nr<0), while (nr>0), if (nr==0), if (nw>0), while (nr>0 || nw>0), while (nw>0), if (nr<0), if (nr>0)) {

signal(OK_to_write) (signal_all(OK_to_read), wait(OK_to_read), signal(OK_to_write), signal(OK_to_read), signal_all(OK_to_write), wait(OK_to_write), // nothing is needed here)

}

}

procedure release_write() {

// nothing is needed here (wait(OK_to_read), signal(OK_to_read), wait(OK_to_write), // nothing is needed here, signal_all(OK_to_read), signal(OK_to_write), signal_all(OK_to_write))

nw = nw - 1;
signal(OK_to_write);

signal_all(OK_to_read) (wait(OK_to_write), signal_all(OK_to_write), // nothing is needed here, wait(OK_to_read), signal_all(OK_to_read), signal(OK_to_write), signal(OK_to_read))
}
}

Maks poeng: 11

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

2 8 0 4 1 9 9

6 Task 6

A savings account is shared by several people (processes). Each person may deposit or withdraw funds from the account. The current balance in the account is the sum of all deposits to date minus the sum of all withdrawals to date. The balance must never become negative. A deposit never has to delay (except for mutual exclusion), but a withdrawal has to wait until there are sufficient funds.

A software developer was asked to implement a monitor to solve this problem, using Signal-and-Continue discipline. Below is the code the developer has written so far. Help the developer to finish the implementation.

monitor Account {
  int balance = 0;
  cond cv;

  procedure deposit(**int** amount) {

// nothing is needed here (signal(cv), // nothing is needed here, signal_all(cv), if (balance > 0) wait(cv), if (balance < 0) signal(cv), if (balance < 0) wait(cv), wait(cv), if (balance > 0) signal(cv));
    balance = balance + amount;

signal(cv) (wait(cv), // nothing is needed here, signal_all(cv), signal(cv), if (empty(cv)) wait(cv));
  }

  procedure withdraw(**int** amount) {

while (amount > balance) wait(cv) (while(empty(cv)) signal(cv);, while(empty(cv)) wait(cv);, signal_all(cv), // nothing is needed here, while (amount > balance) signal(cv), while (balance > amount) wait(cv), while (amount > balance) wait(cv), wait(cv), signal(cv), while (balance > amount) signal(cv));
    balance = balance - amount;

// nothing is needed here (signal(cv), signal_all(cv), wait(cv), // nothing is needed here, empty(cv));
  }
}

Maks poeng: 8

Knytte håndtegninger til denne oppgaven?

9385985

Bruk følgende kode:

7 Task 7

Using Modern CSP, specify behaviour of a traffic light that repeatedly turns green, then yellow, and then red.

Fill in your answer here

```
light -> green -> yellow -> red -> light


```

Maks poeng: 5

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

3 3 1 4 2 8 4

8 Task 8

What will be printed when the following JavaScript code is executed?

function* foo(x) {
 var y = x * (**yield** false);
 return y;
}
var it = foo(100);
var res = it.next(2);

console.log(res.value); *// what will be printed here? Answer:* *false*

res = it.next(3);

console.log(res.value); *// what will be printed here? Answer:* *300*

Maks poeng: 5

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

8 5 3 3 6 0 3

9 Task 9

```
line
number

1    var a = promisify({});
2    var b = a.onResolve(x => x + 1);
3    var c = a.onResolve(y => y - 1);
4    a.resolve(100);
```

Consider the JavaScript code on the image.

Note the syntax here is a blend of JavaScript and λ_p, which uses:

- **promisify** *to create a promise,*
- **onResolve** *to register a resolve reaction*

Draw a promise graph for this code.

Remember to use the names of nodes in that graph that represent the "type" of node:
v for value
f for function
p for promise

with a subscript that represents the **line number** where that particular value/function/promise has been **declared / where it appears first**.

For example, the value 100 on line 4 will be denoted by $\mathbf{v_4}$ in the promise graph.

Please draw the promise graph on a piece of paper that you will get during the exam.
DO NOT WRITE ANYTHING IN THIS TEXT FIELD.

Ord: 0

Maks poeng: 10

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

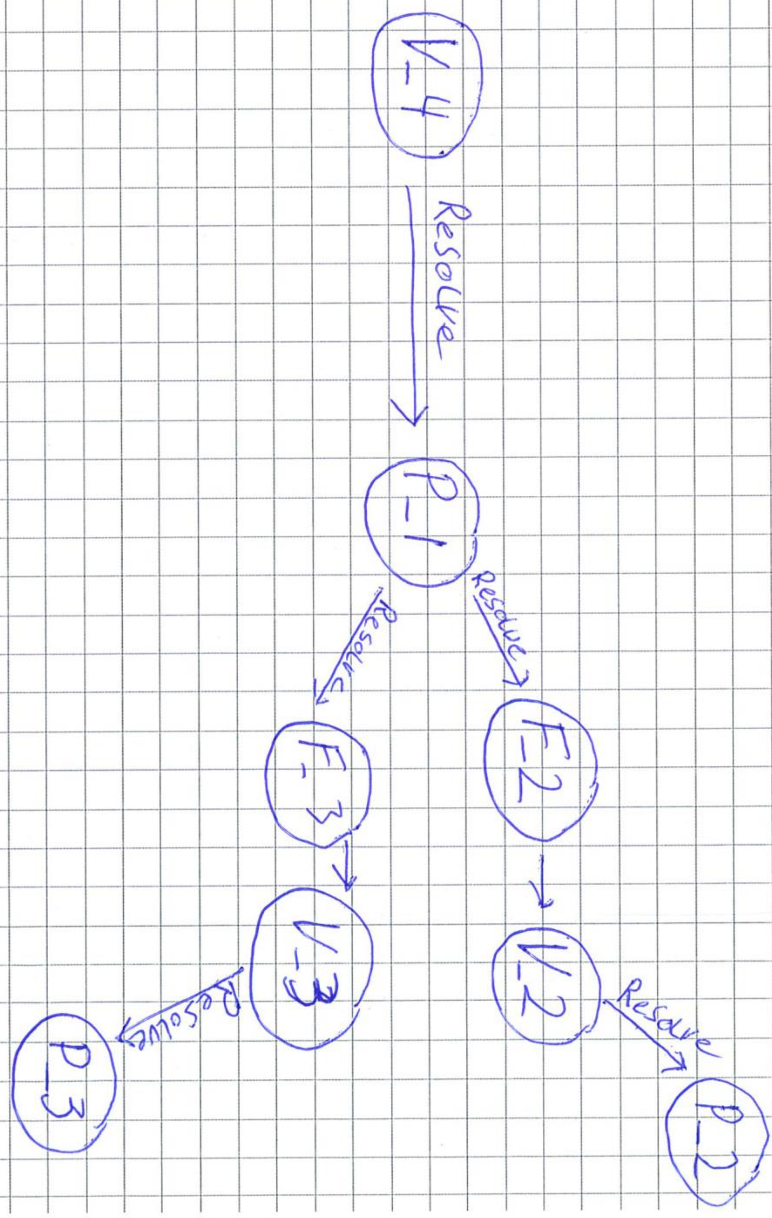
2 9 3 4 9 1 9

Håndtegning 1 av 1

Fyll inn oppgavekode og emneinformasjon på alle skissearkene — Fill out question code and test information on every sheet

Oppgavekode / Question code	Dato / Date	Emnekode / Subject code	Kandidatnummer / Candidate number	Oppgavenummer / Question number	Sidetall / Page number
2934919	21.11.23	INF 214	149	9	1 av/of 1

Tegneområde — Drawing area



10 Task 10

Consider the following HTML/JavaScript attached in the PDF file to this question.

- This code runs on a computer of a super-user, who clicks the button **myButton** 7 (seven) milliseconds after the execution starts.
- This user does **not** click the button **myOtherButton** at all.

What happens at particular time points?

Write an integer number in each of the text boxes. **If something mentioned in the left column on the table does not happen, then write "-1" (negative one) in the corresponding right column.**

what happens	at what time
\``clickHandler`\` starts	18 ✓ milliseconds
\``clickHandler`\` finishes	28 ✓ milliseconds
interval fires (for the first time)	10 ✓ milliseconds
interval fires (for the second time)	40 ✗ (20) milliseconds
interval fires (for the third time)	50 ✗ (30) milliseconds
\``intervalHandler`\` starts (for the first time)	38 ✓ milliseconds
\``intervalHandler`\` finishes (for the first time)	46 ✓ milliseconds
mainline execution starts	*0* milliseconds
mainline execution finishes	18 ✓ milliseconds
\``otherClickHandler`\` starts	-1 ✓ milliseconds
\``otherClickHandler`\` finishes	-1 ✓ milliseconds
promise handler starts	28 ✓ milliseconds
promise handler finishes	32 ✓ milliseconds
promise resolved	a tiny bit after 32 ✗ (18) milliseconds
\``timeoutHandler`\` starts (for the first time)	32 ✓ milliseconds
\``timeoutHandler`\` finishes (for the first time)	38 ✓ milliseconds

what happens	at what time
`**timeoutHandler**` starts (for the second time)	-1 milliseconds
`**timeoutHandler**` finishes (for the second time)	-1 milliseconds
timer fires (for the first time)	10 milliseconds
timer fires (for the second time)	-1 milliseconds
timer fires (for the third time)	-1 milliseconds
user clicks the button	*7* milliseconds

Maks poeng: 9

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

8 2 1 2 3 6 0

11 Task 11.1

There are four rules shown in the PDF attached to this question.

Which of the rules **registers a fulfill reaction on a pending promise**?

Answer:

- (selected) Rule 1
- Rule 2
- Rule 3
- Rule 4

Maks poeng: 1

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

7 7 1 0 8 9 0

12 Task 11.2

There are four rules shown in the PDF attached to this question.

Which of the rules **turns an address into a promise**?

Answer:

- Rule 1
- Rule 2
- Rule 3
- Rule 4

Maks poeng: 1

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

0 5 2 2 8 6 8

13 Task 11.3

$$\frac{a \in \mathit{Addr} \qquad a \in \mathsf{dom}(\sigma) \qquad \psi(a) \in \{\mathsf{F}(v'),\ \mathsf{R}(v')\}}{\langle \sigma, \psi, f, r, \pi, E[a.\mathtt{resolve}(v)] \rangle \rightarrow \langle \sigma, \psi, f, r, \pi, E[\mathtt{undef}] \rangle}$$

What does this rule describe?
Select one alternative:

- This rule handles the case when a pending promise is resolved.
- This rule handles the case when a fulfill reaction is registered on a promise that is already resolved.
- This rule turns an address into a promise
- This rule states that resolving a settled promise has no effect. ✔

Maks poeng: 1

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

3070391

14 Task 11.4

$$\frac{a_1 \in \mathit{Addr} \qquad a_1 \in \mathrm{dom}(\sigma) \qquad a_2 \in \mathit{Addr} \qquad a_2 \in \mathrm{dom}(\sigma) \qquad \psi(a_1) = \mathsf{F}(v) \qquad \pi' = \pi ::: (\mathsf{F}(v), \mathrm{default}, a_2)}{\langle \sigma, \psi, f, r, \pi, E[a_1.\mathtt{link}(a_2)] \rangle \rightarrow \langle \sigma, \psi, f, r, \pi', E[\mathsf{undef}] \rangle}$$

What does this rule describe?

Select one alternative:

- This rule causes a pending promise to be "linked" to another.
- This rule causes a promise to be "linked" to another, with no regards to the state of that original promise.
- This rule causes a non-settled promise to be "linked" to another.
- This rule causes an already settled promise to be "linked" to another. ✔

Maks poeng: 1

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

7413877

15 Task 12

What kind of bugs can be detected by what kind of situations in a promise graph?
In the table, the rows describe various conditions on nodes and edges in a promise graph, and the columns are names of bugs.
Please note that there are 4 rows and 5 columns in the table, so one of the columns is irrelevant.
Please match:

	Bug "Double Resolve/Reject"	Bug "Missing Resolve/Reject Reaction"	another type of bug	Bug "Dead Promise"	Bug "Missing Exceptional Reject Reaction"
a promise with a reject edge, but no reject registration edge					
a promise that has no outgoing registration edges					
a promise with no resolve or reject edges nor any link edge					
multiple resolve (or reject) edges leading to the same promise					

Maks poeng: 2

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

6 0 8 8 7 8 2