UNIVERSITETET I BERGEN

KANDIDAT

131

PRØVE

INF214 0 Multiprogrammering

Emnekode	INF214
Vurderingsform	Skriftlig eksamen
Starttid	01.03.2023 08:00
Sluttid	01.03.2023 11:00
Sensurfrist	--
PDF opprettet	07.11.2023 12:16

Exam structure

Oppgave	Tittel	Oppgavetype
i	Exam structure	Informasjon eller ressurser

Theoretical questions

Oppgave	Tittel	Oppgavetype
1	1	Langsvar
2	2	Langsvar

Semaphores

Oppgave	Tittel	Oppgavetype
3	3	Programmering

Monitors

Oppgave	Tittel	Oppgavetype
4	4	Programmering
5	5	Paring

"Modern" CSP (1985)

Oppgave	Tittel	Oppgavetype
6	6	Programmering

JavaScript generators and iterators

Oppgave	Tittel	Oppgavetype
7	7	Langsvar

JavaScript promises

1 1

Consider the following program:

```
int x = 1;
int y = 1;
co
  < x = x + y; >
||
  y = 0;
||
  x = x - y;
oc
```

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

This don't follow the At-Most-Once-Property because in the first line we are using the variable Y to write to the letter X, then later in the program we are writing the variable Y. Thus it does not follow AMOP.

The possible final values for X and Y is:

X = 2, Y = 0
X = 1, Y = 0
X = 0, Y = 0

Y will always be 0 because the second line will always run, and we do not write Y any other places.
X will vary depending on what gets executed first.

Ord: 96

Maks poeng: 7

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

7 2 4 1 2 9 0

2 2

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

Fill in your answer here

Ord: 0

Maks poeng: 10

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

4 7 4 3 4 7 5

3 3

Three persons *A*, *B*, and *C*, who like hot chocolate very much, have gathered to play the following game at the home of their friend *D*.

To drink a portion of hot chocolate, each of the persons *A*,*B*,*C* needs three "ingredients": the hot chocolate powder, some warmed milk, and a mug. Person *A* has the the chocolate powder, *B* has warmed milk, and *C* has mugs. We assume that persons *A*,*B*, and *C* each has an unlimited supply of these ingredients (i.e., chocolate powder, warmed milk, mugs), respectively.
The host of the party (person *D*) also has an unlimited supply of all the ingredients.

The situation unfolds as follows: *D* puts two random ingredients on the table. The person who has the third ingredient picks up the other two, makes the drink (i.e., puts chocolate powder into a mug and adds warmed milk), and then drinks it.
Person *D* waits for that person to finish.
This "cycle" is then repeated.

Write code in the AWAIT language that simulates this situation.
Represent the persons *A*,*B*,*C*,*D* as processes.

- **Use SPLIT BINARY SEMAPHORES 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 mutex = 1;

//powder
person_A(ingredients){
    P(mutex)
    ingredient = powder
    ingredients.add(ingredient)
    drink() // clears the lsit
    V(mutex)
}

//milk
person_B(ingredients){
    P(mutex)
    ingredient = milk
    ingredients.add(ingredient)
    drink() // clears the lsit
    V(mutex)
}

//mugs
person_C(ingredients){
    P(mutex)
    ingredient = mug
    ingredients.add(ingredient)
    drink() // clears the lsit
    V(mutex)
}

person_D(){
    while(true){
        ingredients[] = twoRandomIngredient() // this metod return two diffrent ranc
        if(!ingredients.contains(powder)){ // if it dosent contain powder choose per
```

```
            person_A(ingredients)
        }
        else if(!ingredients.contains(milk)) { // // if it dosent contain milk choos
            person_B(ingredients)
        }
        else{ // if it dosent contain mugs choose person C
            person_C(ingredients)
        }
    }
}
```

Maks poeng: 25

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

3 7 5 6 9 4 9

4 4

Recall 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.

In the Readers/Writers problem, 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 beginner software developer has implemented this code, but has unfortunately missed a lot of details related to synchronization. Help the beginner developer fix this code.

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

```
// TODO: fix mistakes in this code
monitor ReadersWriters_Controller {
  int nr = 0;
  int nw = 0;
  cond OK_to_write; // signalled when nr == 0 and nw == 0

  procedure request_read() {
    nr = nr + 1;
  }

  procedure request_write() {
    nw = nw + 1;
  }

  procedure release_read() {
    nr = nr - 1;
  }

  procedure release_write() {
    nw = nw - 1;
    signal(OK_to_write);
  }
}
```

Fill in your answer here

```
// Since it isnt possible to check if any processes are waiting the last procedure j
    , then if no readers are waiting the process requesting to write will proced


monitor ReadersWriters_Controller {

   int nr = 0;
   int nw = 0;

   cond OK_to_write;
   cond OK_to_read;


    procedure request_read() {
        if(nw = 1){
            wait(OK_to_read);
        }
        nr = nr + 1;
    }



    procedure request_write() {
        if(nr > 0){
            wait(OK_to_write);
        }
        nw = nw + 1;

    }



    procedure release_read() {
        if(nr = 1){
            signal(OK_to_wirte);
        }
        nr = nr - 1;
    }



    procedure release_write() {
        signal_all(OK_to_read)
        nw = nw - 1;
    }


}
```

Maks poeng: 20

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

3 4 6 7 0 3 9

5 5

There is a duality between monitors and message passing. What is that duality exactly?

In the table, the rows represent notions about monitors, and the columns represent notions about message passing.

Click the circle in a cell to represent that a notion about monitors is dual to a notion about message passing.

Please match the values:

	`send request(); receive reply()`	save pending request	retrieve and process pending request	`send reply()`	local server variables	`receive request()`	arms of case statement on operation kind
procedure call	○	●	○	○	○	○	○
procedure return	○	○	●	○	○	○	○
`wait`	○	○	○	○	○	●	○
`signal`	○	○	○	●	○	○	○
procedure identifiers	○	○	○	○	○	○	○
permanent variables	○	○	○	○	●	○	○
monitor entry	●	○	○	○	○	○	○
procedure bodies	○	○	○	○	○	○	●

Maks poeng: 6

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

9 5 3 4 5 9 6

6 6

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

Fill in your answer here

```
1
```

Maks poeng: 5

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

0 2 6 5 8 4 0

7 7

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

```
function* foo(x) {
  var y = x * (yield false);
  return y;
}
var it = foo(404);
var res = it.next();
console.log(res.value); // what will be printed here?
res = it.next(2);
console.log(res.value); // what will be printed here?
```

Write in your answer just the values (no explanations needed).

Fill in your answer here

false
NaN

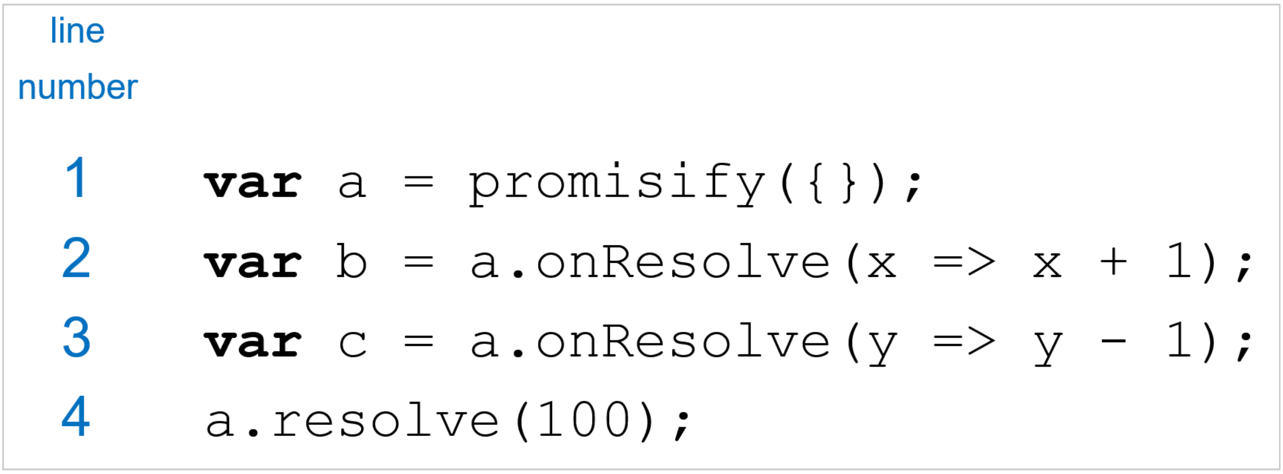
Ord: 2

Maks poeng: 4

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

6 6 9 1 5 2 0

8 8



```
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.

Please draw the promise graph on a piece of paper that you will get during the exam.

Ord: 0

Maks poeng: 10

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

5 0 2 5 5 4 4

9 9

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.

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**` finishes	milliseconds
`**clickHandler**` starts	0 milliseconds
interval fires for the first time	milliseconds
interval fires for the second time	milliseconds
interval fires for the third time	milliseconds
interval fires for the fourth time	milliseconds
`**intervalHandler**` starts	milliseconds
`**intervalHandler**` finishes	milliseconds
mainline execution starts	***0*** milliseconds
mainline execution finishes	milliseconds
promise handler starts	milliseconds
promise handler finishes	milliseconds
promise resolved	a tiny bit after milliseconds
`**timeoutHandler**` starts	milliseconds
`**timeoutHandler**` finishes	milliseconds
timer fires for the first time	milliseconds
timer fires for the second time	milliseconds

what happens	at what time	
timer fires for the third time		milliseconds
timer fires for the fourth time	-1	milliseconds
user clicks the button	**7** milliseconds	

Maks poeng: 9

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

4 1 3 9 7 8 9

10 10.1

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:

- ○ Rule 1
- ○ Rule 2
- ○ Rule 3
- [✓ Rule 4]

Maks poeng: 1

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

4 6 0 2 4 0 8

11 10.2

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:

8 8 5 7 3 6 4

12 10.3

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

What does this rule describe?
Select one alternative:

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

Maks poeng: 1

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

7 7 3 5 6 7 8

13 10.4

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

What does this rule describe?

Select one alternative:

- This rule causes a non-settled promise to be "linked" to another.
- This rule causes an already settled 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 pending promise to be "linked" to another.

Maks poeng: 1

Knytte håndtegninger til denne oppgaven?
Bruk følgende kode:

1 5 2 2 6 3 1