**INF214 Eksamen H2020**

**Task 1:**

1. Yes
2. No
3. Yes
4. Yes
5. Yes
6. Yes
7. Yes
8. ?
9. Yes
10. No
11. Yes
12. No
13. No
14. Yes
15. Yes
16. Yes
17. No
18. No
19. No
20. Yes
21. Yes
22. No
23. No
24. No
25. ?

**Task 2:**

2.1) The volatile keyword is used on variables that values are written by threads. Used to make the program thread-safe and tells the program that the variable may be changed by multiple threads.

2.2) PoisonPill is used when you want to shut down the program at a given condition or time. Sentinel value.

2.3) The waiting time of a thread is increased and the performance of the program is affected.

2.4) Actor programming is vastly used in concurrency programming. The way it works is that a actor has control of its own state and can receive messages. If multiple messages are received it will be placed in a mailbox. The actor performs actions based on what the message was. Actors cannot change the state of other actors.

2.5) Channels lets us do asynchronous message passing. Channels acts as a server for communication. We can use commands such as send, receive and empty.

**Task 3:**

3.1) The semaphores value is changed by the P and V operations. P increments the semaphore by 1, telling the program that the semaphore is currently being used to enter a critical section. The V operation decreases the value by 1, telling the program that the semaphore has left the critical section and can now be passed on.

3.2) The purpose of a barrier is to have all threads currently working, wait for the last thread to finish when themselves are finished, before continuing.

3.3) In synchronous message passing, the sender blocks until the receiver picks up the message. In asynchronous message passing the sender does not block, it just leaves the message and continue working.

**Task 4:**

// global variables

int buf;

bool buf\_full = false;

const int N = 100, Ps = 10, Cs = 10;

process Producer[i = 1 to Ps] {

int a[N];

// ... here we would have some code to fill array ‘a‘ with data

// (we don’t care how exactly -- you don’t have to provide this code here!)

int p = 0;

while (p < N) {

<await (buf\_full == false); buf = a[p]; p = p + 1; buf\_full = true;>

}

}

process Consumer[i = 1 to Cs] {

int b[N];

int c = 0;

while (c < N) {

<await (buf\_full == true); b[c] = buf; c = c + 1; buf\_full = false>

}

**Task 5:**

5.1)

int x = 2;

int y = 3;

co

< x = x + y; >

||

< y = x \* y; >

oc

Possible values:

x = 5

y = 15

y = 6

x = 8

5.2)

{x>=4}

< x = x - 4; >

x >= 0

{x >= 0}

< x = x + 5; >

x >= 5

No interference.

**Task 6:**

int portions = F

sem eat = 1

sem dish\_empty = 0

process baby[i = 1 to N] {

while(true) {

P(eat)

portions = portions - 1

if (portions == 0) {

V(dish\_empty)

}

else {

V(eat)

}

}

}

process parent {

while(true) {

P(dish\_empty)

portions = F

V(eat)

}

}