NTNU
Norwegian University of
Science and Technology

Institutt for teknisk kybernetikk
Fakultet for informasjonsteknologi
matematikk og elektroteknikk

Eksamen i TTK4145 Sanntidsprogrammering 8. June 2017 9.00-13.00 Sensorveiledning

Generally:

Be short and clear. Enumerate items and arguments, use figures where appropriate. You may be punished for text without substance. All questions have similar weights except possibly where marked with (2x), (3x) etc.

Feel free to give your answers in English or Norwegian (...or any reasonable mix of these...)

Be certain your handwriting is readable.

Permitted examination aids:

D: No printed or hand-written support material is allowed. A specific basic calculator is allowed.

Contact person during the exam:

Sverre Hendseth

Telephone 98443807

1 Acceptance tests

1-1) Testing can never prove that a complex system is fault-free since the list of tests might not be complete. Also handling all known errors does not imply that there are not unknown faults left in the system that might cause the system to malfunction.

On this background: How does “acceptance tests” contribute towards fault tolerance?

Expectations: Fault tolerance does not require that the system is fault free. We use testing for detecting whether /anything/ is wrong, not to identify any fault.

1-2) Give examples of what one can test for when making acceptance tests. (Hint: this is the 'learn-by-heart' list; 7 items).

Expectations: This is the “learn-by-heart” list from the book.

- Replication Checks
- Timing Checks
- Reversal Checks
- Coding Checks
- Reasonableness Checks
- Structural Checks
- Dynamic Reasonableness Checks

In a huge skyscraper there are N similar lifts controlled by one control program. The lifts can be configured for different purposes like express lifts going to few designated floors, some serving only the residential floors, some are maintenance lifts with restricted access, occasionally a lift may be disabled etc.

The configuration resides in a text file on a disk somewhere. The format of this file a fixed sequence of key=value pairs of different types, one on each line.

The building management have access to this configuration file and may change lift configuration at need.

You are the programmer that gets the specification of the format of this configuration file, and is to make the module reading this file.

We will be discussing the readConfiguration(char * fileName,TConfigData * configData); function.

1-3) Suggest a complete list of ways you think this function should be allowed to fail/-malfunction/crash.

Expectations: Only one way: /Fail/. Returning this error status, so that the caller can react to it is probably best.

Considerations on how to prepare for treatment, bugfixing etc. is a sidetrack from the question, and assuming any testing on specific things that can go wrong, is bad answer.

With a minimum of added code complexity you can add the feature that the lines of the configuration file can be in any order, giving the building management a bit of flexibility when relating to the configuration file format.

1-4) Give some arguments both for and against doing this.

Expectations: For:

- If this simplifies maintenance, prepares for future changes in spec, or makes such changes easier to do it is a good thing.
- The code enforcing the fixed order may be as complex as the code allowing random order.

Against:

- As soon as this 'creeping feature' is added, one can not go back, closing the door or any additions to the spec requiring fixed order.
- Hotel management might just as well find this a feature to abuse and cause problems.
- Even though this looks like some kind of fault tolerance, it is really more on the side of 'tolerating sloppiness'.

Another alternative way of handling a configuration file with the lines in the wrong order could be to fix it: Detect that the sorting is bad and write back a correct configuration.

1-5) Give some arguments against doing this.

Expectations: This is partially solving one problem, while introducing a lot of complex functionalities and sources for errors, and not preventing that the original problem must be solved in another way anyway. What a stupid idea.

1-6) How would an acceptance test for the 'readConfiguration' functionality look? Suggest a number of /specific/ things to test for, relating to this skyscraper project.

Expectations: It would check that the TConfigData data structure was reasonable. The students need not come up with any 'complete' list, just enough point to show that they can make such a test. Even one test that is an acceptance test rather than an 'error test'. is very good.

Thinks like these could be checked:

- The number of lifts (in the data structure) should be correct.
- One should always be able to call at least one lift to the floor that you are in.
- All residential floors should be available by at least one lift all the time.
- All lifts that are not disabled should serve between 2 and all the floors.
- All floors should be served at least some of the time.
- All active lifts should serve at least one floor with en exit from the building.
- etc...

and more on the structural side:

- Each lift, should be either disabled or have a configuration.
- All fields of the datastructure should be set to something.
- ...

1-7) What would be a reasonable consequence of this acceptance test failing?

Expectations: I would propose something like this:

All lifts are disabled (since the reason for the reconfiguration in the first place might be that one lift needed to be disabled), but can be started in a 'single-lift mode' serving all floors, with a 'secret code' on one of its panels.

A lot of reasonable suggestions might be good answers.

2 Real-Time Programming

2-1) Why are "priorities" so important in real-time programming? — Which features are we hoping to gain in our system by assigning priorities to tasks?

Expectations: To be able to reason on timing/execution behaviour of our program and/or to make schedulability proofs, we need predictablility. Priorities is what make us able to predict which thread is running in any given situation.

2-2) When making a traditional real-time system we need to know an upper bound of the execution time of a calculation. Comment shortly on how this can be achieved/-found.

Expectations: Both how to program (on the design side) and how to estimate should be touched in the answer.

Design: No recursion, no algorithms with undeterminable execution times. No unbounded-size data structures (or dynamic datastructures at all). No unbounded input data dependencies. More?

Estimation 1: We have sequences of predictable-time assembly instructions enclosed by looping with known max bounds. We can multiply...

Estimation 2: We can find the theoretical worstcase input data, measure time of executuin, and then multiply by whatever factor brings us safely to a 'upper bound' side (taking cache/memory, branch prediction etc. uncertainties into consideration)

One of the two last (or another reasonable strategy) are more than enough.

Note: This is not a question about making a /tight/ upper bound of execution. A student knowing that this is extremely difficult is nice but irrelevant.

2-3)

While the functional behavior of a program is given by its code, the same is not true about the exact timing behaviour of the program.

What consequences are there to not beeing able to tell the timing from code, from a maintenance perspective?

Expectations: This is terrible.

Changing the code will necessitate a re-analysis of the whole system. And now with multicore systems even an performance improvment of the code or an upgrade to the hw might lead to worse timing behaviour.

Any mature answer...

3 Left and Right

You are to solve this synchronization problem: Any number of threads perform either the 'Left' operation or the 'Right' operation. You are to ensure that these two operations are released i pairs, one Left and one Right, running concurrently as they progress.

If there are no waiting 'Lefts', then all arriving 'Rights' are blocked and vice versa.

As long as one or both of the previous pair are executing, all other threads must be blocked from executing their Left and Right operations.

There is no synchronization at the end; A thread that has finished its Left or Rigth processing calls 'Finished()', which should not block.

When the last one of the processing pair leaves (calls 'Finished()'), if there is a waiting pair, they should be unblocked and allowed to continue.

Assume each thread has this structure:

```
void
t1(){
  while(1){
    // Work
    if(some condition) Left(); else Right();
      // Work, being certain that only one other thread
      // does Left or Right processing, and that it does
      // the complementary operation to me.
    Finished();
  }
}
```

You are to show how the 'Left()', 'Right()' and 'Finished()' functions can be implemented.

3-1) Implement (pseudocode) the three functions using semaphores as the synchronization mechanism. Skip 'Right()' if its structure is like 'Left()'. Use nonproportional font in your answer.

Expectations:

My solution... Exiting to wait for the student detecting that this does not work :-) Took me some time to set up also (-20 minutes!)

```
Semaphore Mutex(1), Lefts(0), Rights(0);
```

```
int activeThreads = 0;
int waitingLefts = 0;
int waitingRights = 0;

void
Left(){
  wait(mutex);

  // When must I do something more than wait(Left)?
  // Answer: only when nobody is executing already and there is a
  // waiting Right

  if(activeThreads == 0 && waitingRights > 0){
    // There is a Right waiting and, with me, at least one Left.
    activeThreads += 2;
    signal(Right);
    signal(Left);
  }

  waitingLefts++;
  signal(Mutex);

  // Here I can be interupted, and some other Left might get to run, ok
  wait(Left);

  wait(mutex);
  waitingLefts--;
  signal(Mutex);
}

void
Finished(){
  wait(Mutex);
  activeThreads--;

  // When must I do something?
  // Answer: When I am the last active thread leaving, and there are
  // both a waiting Left and Right

  if(activeThreads == 0 && waitingLefts > 0 && waitingRights > 0){
    activeThreads += 2;
    signal(Right);
    signal(Left);
  }

  signal(Mutex);
}
```

3-2) Implement (pseudocode) the three functions using Java (synchronized, wait, notify, notifyAll) (or POSIX - mutexes and condition variables - if you prefer).

Expectations:

```
int activeThreads = 0;
int waitingLefts = 0;
int waitingRights = 0;

synchronized void
Left(){
  // I cannot run when there still are active threads, or when

  while(activeThreads > 0 && waitingRights > 0){
    waitingLefts++;
    wait();
    waitingLefts--;
  }

  activeThreads++
}
```

3-3) Implement (pseudocode) the three functions using Adas protected objects (functions, procedures, entries with guards.)

Expectations: ...

Assume that the “some condition” in the if-statment of the code over is some random function that yields on average as many trues as falses.

The system will deadlock after a time.

3-4) What is a deadlock?

Expectations: The standard definition is something like "a number of participants hindered from progressing by a circular dependency".

This is an interesting context to ask the question, because the "circular dependency"is not so easy to spot, when all threads wants to do Left, and none Right :-)

3-5) Why does the system deadlock? What happens?

Expectations: The system deadlocks when this random function decides that all threads should do 'Left'.

We can easily also imagine a condition in the if statement that will not lead to deadlock in the same manner (like alternating between true and false). This gives yet another indication that 'deadlock analysis' is hard; we would have to go into the semantics of the condition in addition to the thread interactions...

3-6) A tool like Ltsa, which you know from the exercises, can be enormously useful. Think carefully: How would you model this /if/ statement (in the given code) in the FSP modelling language?

Assume that you build a complete and correct model of your implementation using this way of modelling the 'if': What would the result be, of Ltsa checking you model for deadlocks?

Expectations: Part 1: '|' - Both things kan happen.

Trying to use FSP's 'if' or guards ('when') would be futile and must be considered a sidetrack, unless the answer is very mature.

Part 2: Ltsa would report a deadlock, even in the non-deadlock case. (this is good).

3-7) The main problem deadlock analysis is however scalability; Explain why this is a problem.

Expectations: Deadlock analysis is a global analysis, and in principle every new semaphore in the system /multiplies/ the number of states to check by 2.

At N (25 :-)) or so semaphores in the system, it becomes impossible to analyze.

3-8) While Sverre made this exam set, pondering whether to replace "some condition" in the example with real code, he found this statement in the manual pages for the rand() function:

```
  The function rand() is not reentrant or thread-safe, [...]
```

What might this mean?

Expectations: The rand function cannot be called from more threads: There might be 'exercise 1' type errors here. Typically that the rand function delivers the same number or repeats sequences of 'random' numbers.

3-9) What is starvation?

Expectations: One thread does not get the resources it needs due to unfortunate scheduling.

3-10) Readers/writers locks is an interesting case for discussing starvation. Why?

Expectations: The need for readers/writers locks is motivated by a lot of readers. If there are a lot of readers, they may overlap in execution, starving any writers.

3-11) Back to the Left and Right context: Let us assume that all the different threads have different priorities, and that we are in an overloaded situation, where there are always a lot of threads waiting to run, both in the Left and Right queue.

We are worried that the lowest priority threads *never* will get to run. Would you say that this is a problem? Why/Why not?

Expectations: Most probably this is not a problem since who is awakened by signal (or notify...) is not the thread with the highest priority, but the one that have been waiting the longest.

4 Some hard questions

These are questions on the fringes of the curriculum; No depth is required in the answers here; Answer shorty; Just indicate that you know the problem area.

4-1) Discussing shared variable synchronization in Java, the 'inheritance anomaly' comes up. What is the inheritance anomaly?

Expectations: Integrating the 'Object' of object oriented programming with the in-many-sences similar contruct the 'Monitor' sounds like a good idea.

However when it comes to classes and inheritance synchronization makes really not much meaning. We can not 'inherit' synchronization behaviour and achieve any kind of 'hiding the details of the base class'.

4-2) The shared variable synchronization contruct, "Monitors", have a problem with composition; We must be very aware when calling one monitor from the inside of another. What is the problem?

Expectations: Locking the inner monitor /must/ also lock the outer monitor (this has to do with 'releasing control in safe places'). Getting the outer monitor locked in odd places leads to all the same problems that we originally had with semaphores, just on one call/abstraction-level higher.

4-3) Mention one problem that can be solved by the 'setjmp and longjmp' calls in C.

Expectations: From the top op my head:

- Transforming a 'signal', which is 'resumption mode' into 'termination mode' (ATC, which we want)
- Zero-overhead error handling / C exception handling.
- ... Any example, also outside of curriculum :-)

4-4) What is 'optimistic concurrency control'?

Expectations: We assume that interleaving threads under preemptive scheduling does no damage; then use fault tolerance to handle it when it happens anyway.

4-5) 'Cancelling', 'destroying' or 'killing' threads is generally seen as a thing to avoid; Why?

Expectations: We have no idea of the state of the thread when it is killed. It may have allocated memory of semaphores etc. We may deadlock or leak memory++.

But all these problems may be controlled/handled with the principles handled in this course...

4-6) Why can Adas guards only test on the protected object's private variables?

Expectations: If it were not for this restriction we would not be able to know when to re-evaluate the guards and wake up sleeping processes.

4-7) Operations for locking resources are always assumed to be atomic. Why is this so important?

Expectations: Locking is often an integral part of the infrastructure allowing error handling (like in an AA). We would like to avoid that the lock manager needs to get involved in error handling together with the action participants. (this would increase the complexity of the error handling, and possibly demand knowledge in the lock manager of the Action.)

5 Misc

5-1) The following code is Sverres network module interface (C header file) made to simplify the networking aspect of the project.

```
#ifndef SVERRESNETWORK_H
#define SVERRESNETWORK_H

```

```
// Note that the callback functions you provide may be called by the
// threads created by this module. Synchronize the access to any
// resources.

typedef void (*TMessageCallback)(const char * ip, char * data, int datalength
    );
typedef void (*TTcpConnectionCallback)(const char * ip, int created);


// Misc
char * getMyIpAddress(char * interface); // NB: allocates the return string!

// UDP & Broadcast
void udp_startReceiving(int port,TMessageCallback callBack);
void udp_send(char * address,int port,char * data, int dataLength);
void udp_broadcast(int port,char * data, int dataLength);


// TCP
void tcp_init(TMessageCallback messageCallback, TTcpConnectionCallback
    connectionCallback);
void tcp_startConnectionListening(int port);
void tcp_openConnection(char * ip,int port);
void tcp_send(char * ip,char * data, int datalength);


#endif
```

Criticize the interface; point out both strengths and weaknesses.

Expectations:

Sverres own comments:

Bad:

- Is the fact that the module spawns threads clear enough?
- Obviously this is one module providing more segments of services. - Not ′do one thing and do it well...′
- This ′NB: allocates the return string!′ comment is clear, but it is still bad form to leave responsibility of an allocation you have done to your user.
- ′sverresnetwork′ is a bad name.
- More?

Good:

Any mature (that is, relating to real code quality criteria - the code complete checklists) praise is good.

5-2) Backward error recovery may, when generalizing to multi-thread systems give the domino effect. Explain. How can we avoid the domino effect?

Expectations: Everything that looks like the "domino effect figure" is great for the first part of the question. Coordinating recovery points is the solution to the second.

5-3) When doing forward error recovery in a multi thread setting, the need arises for the different threads to get to know about errors that happen in other threads. List mechanisms that can be used to convey such information between threads.

Expectations: Avstemmingen i etterkant av en AA. En kan polle feilstatus variable, eller hvis systemet er meldingsbasert kan en sende feilmeldinger. Ellers er asynchronous transfer of control det som er mest behandlet i boken: select then abort i Ada, AsynchrouneslyIterruptedExceptions i Java, og setjump/longjump-trikset eller pthread_cancel i C/POSIX.

5-4) What is an Atomic Action? Which problem(s) is Atomic Actions meant to solve?

Expectations: More other good answers exist for the first part here. Pointing out the three boundaries (side, start and end), possibly with standard mechanisms to achieve them (locking, explicit membership, and two-phase commit protocol) is reasonable enough.

The problem to be solved: If more participants cooperates on something, they must possibly also cooperate in handling errors. AA provides the framework for achieving this, containing the errors and avoiding the domino effect.

"A mechanism/infrastructure for error containment when we have cooperating threads." basicly covers both questions quite well.

5-5) What do we achieve by using process pairs?

Expectations: Fault tolerance by (dynamic) redundancy. High availability since the switching between replicas happens immediately.

5-6) Explain shortly how process pairs work.

Expectations: Two programs, the primary and the backup are run at the same time. The primary does the side effects (like "send answer to the client") and sends the program state/checkpoints to the backup (though in the opposite order!) along with IAmAlive messages. The backup broadcasts IAmMaster when enough IAmAlive messages have been missed - and continues from the last checkpoint.