Norwegian University of Science and Technology

Department of Engineering Cybernetics

Exam in course TTK41415
Real-time Programming

Friday 8th of June 2018

Time: 09:00 - 13:00

Contact:	Torleif Anstensrud Phone: 95808760
Examination aids:	No printed or hand-written support material is allowed. A specific basic calculator is allowed.
Language:	English
Number of pages:	8

Task 1. - Fault Tolerance

a) In the context of fault tolerance, how are the terms **fault**, **error** and **failure** related to each other?

b) What is meant by **merging of error modes** in the context of simplifying error detection and handling?

c) When is a system **failfast**, and how does this contribute to **error containment**?

d) Why is **fault tolerance** considered a better approach than **fault prevention** when designing reliable systems?

e) When designing **acceptance tests**, what are the 7 different types of checks that are often implemented?

f) What are the two main strategies used for **error recovery**, and how do they differ?

g) When is **Asynchronous Transfer of Control** between threads desirable, and what tools are available in C/POSIX, Java and Ada to implement it?

h) The **domino effect** can occur when recovery points are used within multithreaded systems. What triggers this effect, and what is the worst-case result of it? Illustrate your answer with a figure.

i) What are **transactions**, and how do they prevent the domino effect?

j) What is the **Two-Phase Commit** protocol? Illustrate its use with a simple figure or sequence diagram.

k) Gray discusses three cases in low-level design for fault tolerance by **redundancy**. List these three cases and discuss briefly techniques for how **redundancy** can be implemented to handle failures in each of the cases.

Task 2. - Shared Variable Synchronization

a) What are the standard problems that are associated with sharing memory between threads?

b) A common synchronization problem requires two threads to both reach a certain point in their execution before continuing, referred to as a rendezvous. Using the incomplete code below, implement a simple rendezvous using semaphores that synchronizes the threads at the indicated spot.

```
void thread1(){
  doWorkT1();

  /* Wait here for thread 2 to complete its work before
    exiting*/

  return;
}

void thread2(){
  doWorkT2();

  /* Wait here for thread 1 to complete its work before
    exiting*/

  return;
}
```

c) The following system consists of two threads, `stoker` and `stacker`, which are responsible for adding wood to a fire and replenishing the wood pile, respectively. When the stoker runs out of wood, it should wait for the stacker to replenish the woodpile. Additionally, the 3 global variables `woodPileEmpty`, `woodPileBusy` and `woodInPile` keep track of the status of the woodpile.

The system as written contains race conditions, what is a race condition, where are they in the code below and what it the worst case result of these race conditions?

```
bool woodPileEmpty = false;
bool woodPileBusy = false;
int amountWoodInPile = 10;

void stoker(){
  while(1){
    if(!woodPileBusy){
      woodPileBusy = true;
      if(amountWoodInPile > 0){
        amountWoodInPile -= 1; // Adding wood to fire
```

```
        } else {
          woodPileEmpty = true;
          woodPileBusy = false;
          suspend();
        }
        woodPileBusy = false;
      }
    }
}

void stacker(){
  while(1){
    if(!woodPileBusy){
      woodPileBusy = true;
      if(woodPileEmpty){
        amountWoodInPile = 10; // Refilling woodpile
        resume(stoker);
        woodPileEmpty = false;
      }
      woodPileBusy = false;
    }
  }
}
```

d) Reimplement the stoker and stacker problem from the previous task using semaphores, and show how this solution avoids the race conditions from the previous implementation.

e) The roller coaster problem is a thread synchronization problem that can be stated as follows

> Suppose there are n passenger threads and a car thread. The passengers repeatedly wait to take rides in the car, which can hold C passengers, where $C < n$. The car can go around the tracks only when it is full.

Additionally, the passenger threads should queue up and not board until given the signal from the car thread. The car should then wait until C passenger threads have boarded before departing. Once the car has finished the ride, the passengers should disembark following a signal from the car thread.

Implement a solution to the roller coaster synchronization problem using semaphores. Your solution should consist of the the following two thread functions:

```
void car_thread(){
  /* Synchronization code for car thread goes here, the car
     thread should allow for C passenger threads to board (
     run) simultaenously */
}

void passenger_thread(){
  /* Synchronization code for passenger thread goes here,
     your system should support n of these threads */
}
```

To simplify the problem, the following semaphores and variables are defined for you:

```
mutex = Semaphore(1)
mutex2 = Semaphore(1)
int boarders = 0
int unboarders = 0
boardQueue = Semaphore(0) /* Passengers wait here before
   boarding car*/
unboardQueue = Semaphore(0) /* Passengers wait here before
   leaving the car*/
allAboard = Semaphore(0) /* Indicates the car is full */
allAshore = Semaphore(0) /* Indicates all passengers have
   left the car */
```

f) Show how you can solve the roller coaster problem with Java using `synchronized`, `wait` and `notify`/`notifyAll`.

g) Show how you can solve the roller coaster problem with Ada using `protected` objects containing `functions`, `procedures` and/or `entries` with guards.

Task 3. - Scheduling

a) A common approach to designing hard real-time systems is to assign each deadline to a thread, and switch between them using preemptive scheduling. Alternatively, the deadlines can be implemented as a sequence of tasks inside a big while loop, known as a **Cyclic Executive**. What are some benefits and drawbacks of this approach?

b) Given the following task set:

Task	Period T	Computation Time C	Priority P
a	16	8	1
b	8	2	2
c	4	1	3

Perform an utilization test on the task system using the formula

$$U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N \left(2^{\frac{1}{N}} - 1\right), \tag{1}$$

and comment on whether the system is schedulable.

c) Again, consider the task set:

Task	Period T	Computation Time C	Priority P
a	16	8	1
b	8	2	2
c	4	1	3

Draw a time-line for the task set, and indicate if any deadlines are missed. Does this result conflict with your result from the utilization test, if so why? Assume that all tasks are released at time 0, and that $P = 1$ is the lowest priority task.

d) Systems running on **Fixed-Priority Scheduling** can be affected by **Priority Inversion**. What is the cause of this problem, and what are the potential consequences?

e) What is the common approach to solving the **Priority Inversion** problem?

Task 4. - Code Quality

a) The following code represents parts of an interface for a module that controls the walking speed of a bipedal robot.

```
1 #ifndef WALKING.H
2 #define WALKING.H
3
4 typedef struct {
```

```
  int & gyroStatus;
  int & accStatus;
  double walkingSpeed;
} TStatus;

void StartLocomotion(void);
void Exit(void);

void SetWalkingSpeed(double speed);
float GetWalkingSpeed(int & gyroStatus);
void SetLeftMotor(char commandByte);
void SetRightMotor(double pulseFrequency);

void PrintStatus(TStatus currentStatus);
TStatus GetStatus(void); // NB: Uses global semaphore
    statusLock

#endif
```

Criticize the interface from a code quality aspect by pointing out any weaknesses.

b) Criticize the following routine from a code quality aspect, using the checklists from Code Complete as a guide:

```
void HandleStuff( CORP_DATA & inputRec, int crntQtr,
    EMP_DATA empRec, double & estimRevenue, double
    ytdRevenue, int screenX, int screenY, COLOR_TYPE &
    newColor, COLOR_TYPE & prevColor, StatusType & status,
    int expenseType )
{
  int i;
  for ( i = 0; i < 100; i++ ) {
    inputRec.revenue[i] = 0;
    inputRec.expense[i] = corpExpense[ crntQtr ][ i ];
  }
  UpdateCorpDatabase( empRec );
  estimRevenue = ytdRevenue * 4.0 / (double) crntQtr;
  newColor = prevColor;
  status = SUCCESS;
  if ( expenseType == 1 ) {
    for ( i = 0; i < 12; i++ )
      profit[i] = revenue[i] - expense.type1[i];
  }
  else if ( expenseType == 2 ) {
    profit[i] = revenue[i] - expense.type2[i];
  }
  else if ( expenseType == 3 ) {
    profit[i] = revenue[i] - expense.type3[i];
```

```
21     }
22 }
```