TECHNISCHE UNIVERSITEIT EINDHOVEN
Faculteit Wiskunde en Informatica

Examination Operating Systems (2XN05)
on November 1, 2010, 14.00h-17.00h.

The exam consists of two parts that are handed in separately. Part I consists of knowledge-questions that must be answered in maximally 30 minutes, *on a separate sheet.* Extensive explanations are not required in part I, compact answers will be appreciated, answers will be judged correct/wrong only. After you have handed in part I, possibly much sooner than this 30 minutes, you may use the course book and the slides from the lectures for part II. (**Note**: just the lecture slides, no slides concerning exercises or previous exams or any other notes.)

Work clearly. Read the parts before you start. Scores for exercises are indicated between parentheses. The score sums to 10 points. There are 3 pages in total.

PART I
(4 points)

---

1. Give at least three motivations for the existence of operating systems.
    a. *abstractie*, van onderliggende diversiteit
    b. *virtualisatie*, algemene concepten (lineair geheugen, file systeem, processor) beschikbaar voor applicaties als virtuele machine
    c. *sharing,* van functionaliteit die alle applicaties nodig hebben
    d. *resource management*
    e. *concurrency,* gelijktijdig gebruik kunnen maken van resources
    f. *programma portabiliteit*, d.m.v. gestandaardizeerde interfaces

2. What are the steps in processing a system call?
    a. 1-3: pushing parameters
    b. 4: call library function
    c. 5: put code for *read* in reg.
    d. 6: trap: switch mode and call particular handler
    e. 7: handler calls read function handler
    f. 8: handler performs read actions (a.o. store data at address)
    g. 9-11: control back to caller

3. Under what circumstances is busy waiting acceptable? And where is it applied?
    a. When the waiting time is guaranteed to be short. Typical in sychronisation between processors in critical sections.
    b. When it is the only thing that may happen *on that processor*. Typical in special-purpose hardware for devices.

4. Mention the four correctness concerns in concurrent programs or systems.
    a. functional correctness, minimal waiting, absence of deadlock, fairness

5. Wat are the two principles of condition synchronization?
   At places where the truth of a condition is required: check and block
   At places where a condition *may have become true*: signal waiters

6. Give 3 rules of thumb to avoid deadlock in action synchronization.
   Let critical sections terminate; avoid greediness; avoid cycles in semaphore calls.

7. What is *interference* of concurrent programs?
   The truth of an assertion that on local reasoning would be true is falsified.

8. In Posix, the value of a semaphore can be inspected. Is this useful in a program? If yes, explain with an example.
   a. You can use it to avoid blocking of a process, for example by inspecting the semaphore from time to time doing something useful in between. Disadvantage is that thev value is not stable: you cannot assert that the value is 0 after you have tested that.

PART II
(6 points)

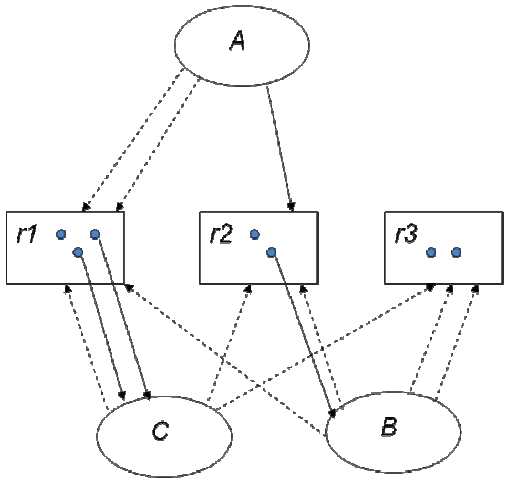
1. Given is the following taskset.

Task	R1	R2	R3
A	2	1	0
B	1	2	2
C	3	1	1

Columns *R1, R2* and *R3* represent maximum numbers of resources of type *R1, R2* and *R2* needed by the tasks.

a) (1.0) Consider greedy allocation (each request is honored as soon as possible). Give values for numbers of available resources of the three types such that no deadlock is possible and their sum is minimal. Motivate your answer.
Answer. A deadlock with a maximal number of resources is constructed if all three tasks are blocked and are short of 1 resource. This gives a total of 10 resources, e.g. R1:5, R2:3, R3:2 with A one short of R1, B one short of R2 and C one short of R3. Adding one resource to either of R1 or R2 makes it impossible for the three tasks to block and, consequently, the other two tasks cannot block either.

b) (0.5) Now assume the following numbers of resources are available: R1: 3, R2: 2, R3: 2. The bankers algorithm is used for allocating resources. Is it possible that *A* is blocked, waiting for R2 while there is in fact an R2 resource available? If yes, give a corresponding full claim graph.
Answer: yes, this is possible according to the graph below.

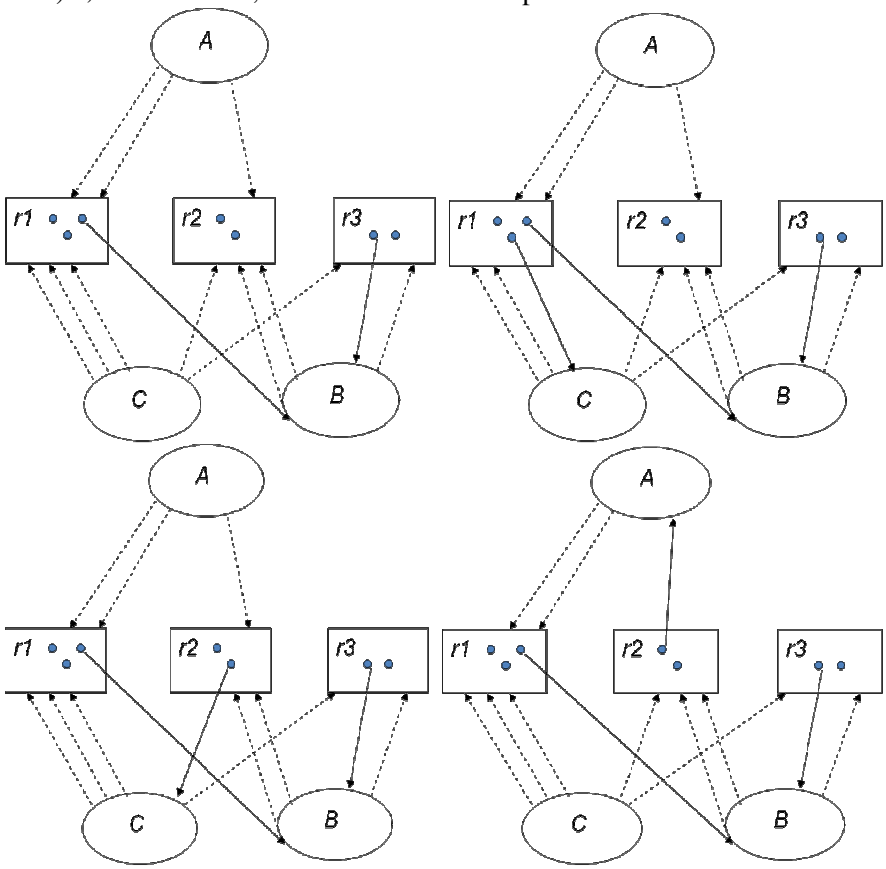


c) (0.75) Consider the situation after *B: req(R1, 1); B: acq(R1,1); B: req(R3, 1): B: acq(R3,1).* Will the bankers algorithm grant the following requests? Motivate your answer with a diagram.

i. C: req(R1, 1)
ii. C: req(R2, 1)

iii. A: req(R2,1)

Answer. The next four graph gives respectively: the situation after the two given acquisitions of *B,* the situation if *i,ii and iii* were to be allowed (note: they are not incremental but independent; otherwise it would not be consistent with the table). *i,iii:* is allowed, *ii*: is not allowed: no open task.

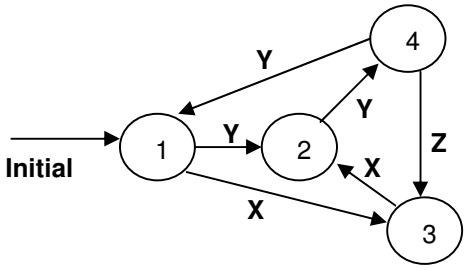


2. There are three groups of threads, corresponding to the following three procedures.

var *st: int { initially 1 }*

proc *A* = **|[while** *true* **do** *X* **od]|;**
proc *B* = **|[while** *true* **do** *Y* **od]|;**
proc *C* = **|[while** *true* **do** *Z* **od]|;**

Execution of these threads must be constrained according to the following state transition diagram. Variable *st* records the state and must be modified together with the execution of the actions *X, Y* and *Z.*



a) (1.5 pt) Synchronize this system using condition variables according to this diagram. Add the modifications of variable *st* and make sure that proper exclusion is guaranteed.
Answer. Introduce condition variables *CX, CY* and *CZ* corresponding to conditions guarding execution of the three functions. Semaphore *m* with initial value 1 is used for exclusion. The call to *X* is embedded in a conditional critical section as follows.

P(m);
while *st*<>1 **and** *st*<>3 **do** *Wait (m, CX)* **od**;
X;
if *st=1* **then** *st:*=3; *Signal(CX)* **else** *st:*=2; *Signal(CY)* **fi**;
V(m)

Similarly, for *Y* and *Z.*

P(m);
while *st*=3 **do** *Wait (m, CY)* **od**;
Y;
if *st=1* **then** *st:*=2; *Signal(CY)*
else if *st=2* **then** *st:*=4; *Signal(CY)*; *Signal(CZ)*
else *st:*=1; *Signal(CX); Signal(CY)*
fi; fi;
V(m)

P(m);
while *st*<>4 **do** *Wait (m, CZ)* **od**;
Z;
st:=3; *Signal(CX);*
V(m)

b) (1.0) Discuss fairness of your solution; if it is not fair modify it such that it becomes fair.
Answer. We may have sequences that avoid both *X* and *Z* as well as sequences that avoid just *Z.* With our given threads this means that one or two of the threads may be starved. Each sequence, however, visits state 4. If in this state we give preference to *Z* the system is fair in the sense that there is a bound on the number of actions of other threads on which thread *C* has to wait. Such preference is added in the standard way by using the waiters on *CZ.*Similarly, in order to avoid starvation of *A* when no *C* is present we give preference to *X* in state 1.

P(m);
while *st*=3 **or** *(st=4* **and not** *Empty(CZ))* **or** *(st=1* **and not** *Empty(CX))*
do *Wait (m, CY)*
od;
Y;
if *st=1* **then** *st:*=2; *Signal(CY)*
else if *st=2* **then** *st:*=4; *Signal(CY)*; *Signal(CZ)*
else *st:*=1; *Signal(CX); Signal(CY)*
fi; fi;
V(m)

c) (0.25) Is it required that accesses to *st* are performed inside a critical section only? Is it required that the actions *X, Y* and *Z* occur only within critical sections? Explain your answer.

Answer. *st* is a shared variable and interference between inspection and assignment – as in the if statements above – would yields erroneous results. In addition, according to the diagram execution of the action should be combined atomically with the execution of the state adaptation. If the latter would be done non-atomically before or after the action, the state adaptation could run ahead of the actual execution of the actions. Hence, both are required to occur in a single critical section.

3. (1 pt) Consider the following procedure.

var *x: int;* { *x* is initialized to 0 }
proc *A* = |[**var** *r: int;* **while** *x* <> *10* **do** *r:=x; x := r+1* **od**]|

Assume that two threads both execute *A,* hence, two instances of *A* run concurrently. Is it possible for variable *x* to obtain a value larger than 10? If yes, give a trace; if no, give an argument.

Answer. yes, *x* can become larger than 10. Name the two instances *A0* and *A1.* Consider the following trace:

A0: (x<>10)(r:=x; x := r+1) 9 times
A0: (x<>10)(r:=x)
A1: (x<>10)(r:=x); x:= r+1)(x=10)
A0: (x := r+1)(x<>10) ….. (infinite repetition)