Computer Systems

Exercise 6

REMINDER: Exam question

vs

My email: ferraric@ethz.ch

https://hanslodge.com/clipart/8Txr4xdoc.htm
http://clipartmag.com/angel-cartoon-clipart

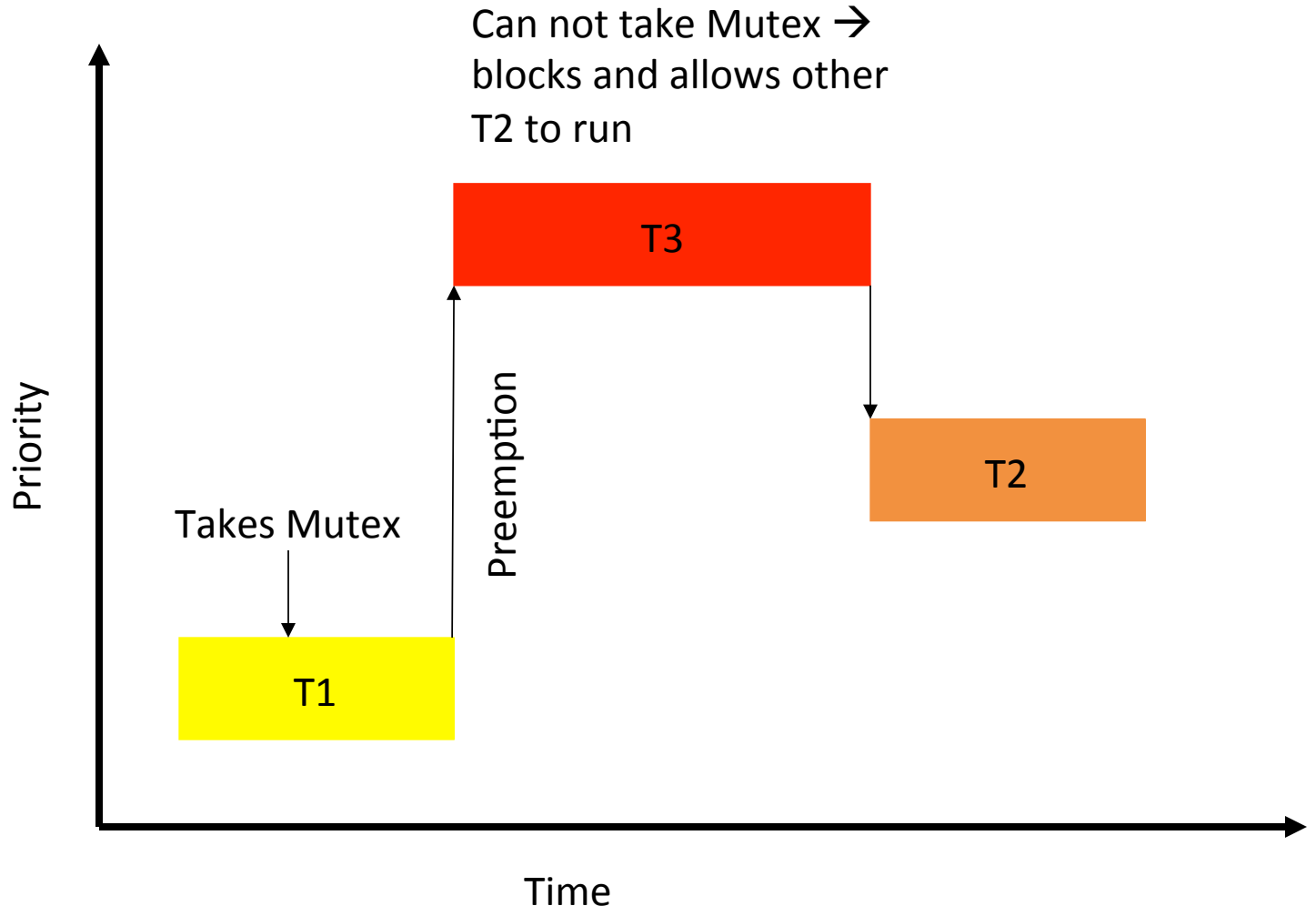
Last Exercise

- What is the problem with shortest job first (SJF)?
  - *Answer: Long jobs are potentially never scheduled*
- What is the advantage of shortest job first (SJF)?
  - *Answer: It minimizes the average time a task is waiting to be scheduled (and therefore the average turnaround time)*
- What is the benefit of round robin(RR)?
  - *Answer: It has a good response time. (And is easy to implement/understand/analyze)*
- What is the big conceptual difference between earliest deadline first (EDF) and round robin (RR)?
  - *Answer: EDF is for realtime workloads, so tasks have priorities. RR treats all tasks the same.*

Last exercise

- Why do hard realtime systems often don't have dynamic scheduling?
  - *Answer: In a dynamic setup, the feasibility of a correct schedule is often not guaranteed -> we don't want that for hard realtime systems.*
- What algorithm would be good for interactive workloads?
  - *Round robin (RR), because it has the best response time.*
- What is priority inversion?
  - *Answer: A lower priority task is running and holding a lock for A, it gets preempted by a higher priority task which can't proceed because the lower priority thread is still holding the lock and gives up the scheduler to a medium priority thread.*

Priority Inversion



Last exercise

- What is the problem with priority inversion?
  - *Answer: High-priority tasks cannot proceed because low-priority tasks are running.*
- What preconditions must hold to achieve priority inversion?
  - *Answer: There must be at least three runnable tasks of different priorities. The lowest one must hold a lock that the highest one needs to proceed.*
- How can this problem be solved?
  - *Answer: This can be solved by the priority inheritance scheme. The task holding the lock temporarily inherits the priority of the task which wants to acquire the lock.*

Last exercise

- How many levels of priority inheritance do you need?
  - *You need as many levels as possible priority levels.*
- Why?
  - *Because the low priority thread has to inherit the priority of the high priority thread that wants to acquire the priority.*
- How could you implement that?
  - *Problem: A low priority thread holding multiple locks that multiple high priority tasks want. So after releasing one lock and one high priority task running, the priority shouldn't be reset to the original level, but to the one of the other thread needing the lock.*
  - *Solution: A linked list of priorities that the thread will get reset to.*

Last exercise

- State three advantages/disadvantages of placing functionality in the device controller (hardware), rather than in the kernel (software)?
  - *Advantages:*
    - *Bugs are less likely to cause an operating system crash*
    - *Performance can be improved by utilizing dedicated hardware and hard-coded algorithms*
    - *The kernel is simplified by moving algorithms out of it*
  - *Disadvantages*
    - *Bugs are harder to fix- a new firmware version or new hardware is needed*
    - *Improving algorithms likewise require hardware update rather than just a kernel or device driver update*
    - *Embedded algorithms could conflict with application's use of the device, causing decreased performance*

Last exercise

- Why might a system use interrupt-driven I/O to manage a single serial port(character device), but polling I/O to manage a front-end processor, such a terminal concentrator?
  - *Polling can be more efficient than interrupt-driven I/O when I/O is frequent and of short duration.*
- Describe a hybrid strategy that combines polling, sleeping and interrupts for I/O device service. For each of these three strategies (polling, interrupts, hybrid), describe a computing environment in which that strategy is more efficient than is either of the others.
  - *Hybrid approach: Switch between polling and interrupts depending on the length of the I/O operation wait. Eg. Loop N times and if device is still busy at n+1, go to sleep. This approach is good for very long or very short busy times*
  - *Pure polling is best with very short wait times*
  - *Pure interrupts is best with long wait times*

Last exercise

- How does DMA increase system concurrency? How does it complicate the hardware design?
  - *DMA increases system concurrency by allowing the CPU to perform tasks while the DMA system transfers data via the system and memory buses.*
  - *Hardware design is complicated because the DMA controller must be integrated into the system and the system must allow the DMA controller to be a bus master.*
- Although DMA does not use the CPU, the maximum transfer rate is still limited. Consider reading a block from disk. Name three factors that might ultimately limit the file transfer.
  - *Limited speed of the I/O device*
  - *Limited speed of the bus*
  - *A disk controller with no internal buffers or too small buffer size could also limit the performance of the read file operation*

Last exercise

- A DMA controller has multiple channels that can be used by device drivers to request a DMA transfer. The controller itself is capable of requesting a 32 bit word ever 100 nsec. A response takes equally long. How fast does the bus have to be to avoid being a bottleneck?
  - *Every bus transaction has a request and a response each taking 100 nsec, or 200 nsec per bus transaction. This gives 5 million bus transactions/sec. If each one is four bytes, the bus should be able to handle 20MB/sec. The fact that these transmissions may be distributed over multiple I/O devices in round-robin fashion is irrelevant. A bus transaction takes 200 nsec, regardless of whether consecutive requests are to the same device or different device, so the number of channels the DMA controller has does not matter.*

Byzantine nodes

- Node which has arbitrary behavior
- So it can:
  - Decide not to send messages
  - Sending different messages to different nodes
  - Sending wrong messages
  - Lie about input value
- If an algorithm works with f byzantine nodes, it is f-resilient

Different Validities

- Any-input validity:
  - The decision value must be input of any node
  - That includes byzantine nodes, might not make sense
- Correct-input validity:
  - The decision value must be input of a correct node
  - Difficult because byzantine node could behave like normal one just with different value
- All-same validity:
  - if all correct nodes start with the same value, the decision must be that value
- Median validity:
  - If input values are orderable, byzantine outliers can be prevented by agreeing on a value close to the median value of the correct nodes

Byzantine agreement in the synchronous model

- Assumption: nodes operate in synchronous rounds. In each round, each node may send a message to each other node, receive the message by other nodes and do some computation.
  - -> runtime is easy, since it is only the number of rounds

King Algorithm (synchronous byzantine agreement)

Idea:
If not all correct input nodes have the same value, decide on value of one correct input node. Ensure this by doing f+1 rounds, since there must be at least one correct input node.

Algorithm 11.14 King Algorithm (for $f < n/3$)

```
1: x = my input value
2: for phase = 1 to f + 1 do
     Round 1
3:   Broadcast value(x)
     Round 2
4:   if some value(y) received at least n − f times then
5:     Broadcast propose(y)
6:   end if
7:   if some propose(z) received more than f times then
8:     x = z
9:   end if
     Round 3
10:  Let node v_i be the predefined king of this phase i
11:  The king v_i broadcasts its current value w
12:  if received strictly less than n − f propose(y) then
13:    x = w
14:  end if
15: end for
```

Do until at least one correct input node

Send out own value

If some value received from all nodes but byzantine ones (or at least ((n – f)- f) correct ones), propose that value

If some value proposed by at least one correct node, set your value to that value

King of this phase broadcasts its value

If didn't get propose from all nodes but byzantine ones (or at least ((n – f)- f) correct ones), set your value to value of king

King Algorithm (synchronous byzantine agreement)

Why f+1?

- Because there are f byzantine nodes, at least one of the kings will be a correct node

Algorithm 11.14 King Algorithm (for $f < n/3$)

```
1:  x = my input value
2:  for phase = 1 to f + 1 do
        Round 1
3:      Broadcast value(x)
        Round 2
4:      if some value(y) received at least n − f times then
5:          Broadcast propose(y)
6:      end if
7:      if some propose(z) received more than f times then
8:          x = z
9:      end if
        Round 3
10:     Let node v_i be the predefined king of this phase i
11:     The king v_i broadcasts its current value w
12:     if received strictly less than n − f propose(y) then
13:         x = w
14:     end if
15: end for
```

King Algorithm (synchronous byzantine agreement)

Algorithm 11.14 King Algorithm (for $f < n/3$)

1: $x =$ my input value
2: **for** phase $= 1$ to $f + 1$ **do**

Round 1

3: Broadcast `value`(x)

Round 2

4: **if** some `value`(y) received at least $n - f$ times **then**
5: Broadcast `propose`(y)
6: **end if**
7: **if** some `propose`(z) received more than f times **then**
8: $x = z$
9: **end if**

Round 3

10: Let node v_i be the predefined king of this phase i
11: The king v_i broadcasts its current value w
12: **if** received strictly less than $n - f$ `propose`(y) **then**
13: $x = w$
14: **end if**
15: **end for**

King Algorithm (synchronous byzantine agreement)

Why n-f?

- Because if there are n-f correct nodes, so we can't wait for more. If we wait for less than f + 1 nodes, all the input values could be fake. Because 3f < n, n – f > f.
- Ensures only one proposal: If one node sees n-f values v, then every other node sees at least n-2f times v. Because n-(n-2f) = 2f < n-f, there can be no proposal for another value.
- All same validity ensured here!

Algorithm 11.14 King Algorithm (for $f < n/3$)

```
1:  x = my input value
2:  for phase = 1 to f + 1 do
        Round 1
3:      Broadcast value(x)
        Round 2
4:      if some value(y) received at least n − f times then
5:          Broadcast propose(y)
6:      end if
7:      if some propose(z) received more than f times then
8:          x = z
9:      end if
        Round 3
10:     Let node v_i be the predefined king of this phase i
11:     The king v_i broadcasts its current value w
12:     if received strictly less than n − f propose(y) then
13:         x = w
14:     end if
15: end for
```

King Algorithm (synchronous byzantine agreement)

Why more than f?

- If we just waited for <= f propose messages, they all could be byzantine.

Algorithm 11.14 King Algorithm (for $f < n/3$)

```
1:  x = my input value
2:  for phase = 1 to f + 1 do
        Round 1
3:      Broadcast value(x)
        Round 2
4:      if some value(y) received at least n − f times then
5:          Broadcast propose(y)
6:      end if
7:      if some propose(z) received more than f times then
8:          x = z
9:      end if
        Round 3
10:     Let node v_i be the predefined king of this phase i
11:     The king v_i broadcasts its current value w
12:     if received strictly less than n − f propose(y) then
13:         x = w
14:     end if
15: end for
```

King Algorithm (synchronous byzantine agreement)

Why n-f propose messages?

- Similar as for n-f broadcast messages. We can wait for at most n-f ones because those are the correct nodes, and we have to wait for at least f+1 ones.

After a correct king, the correct nodes will not change their values anymore! Why?

- If all of them have less than n-f propose messages, all correct nodes will have the king value and then “all same validity” holds. If one does not adapt, this means that it got n-f propose messages. This means, every other message got at least n-f-f > f propose messages, so it adapted its value to the propose. So the king also adapted it’s value and again all nodes have the same value.

Algorithm 11.14 King Algorithm (for $f < n/3$)

1: x = my input value
2: **for** phase = 1 to $f + 1$ **do**

Round 1

3: Broadcast `value`(x)

Round 2

4: **if** some `value`(y) received at least $n - f$ times **then**
5: Broadcast `propose`(y)
6: **end if**
7: **if** some `propose`(z) received more than f times **then**
8: $x = z$
9: **end if**

Round 3

10: Let node v_i be the predefined king of this phase i
11: The king v_i broadcasts its current value w
12: **if** received strictly less than $n - f$ `propose`(y) **then**
13: $x = w$
14: **end if**
15: **end for**

King Algorithm (synchronous byzantine agreement)

- Does it solve byzantine agreement?
  - Validity: All same validity!
  - Agreement: They agree at least after the first correct king.
  - Termination: After (f+1)*3 rounds

Algorithm 11.14 King Algorithm (for $f < n/3$)

```
1:  x = my input value
2:  for phase = 1 to f + 1 do
        Round 1
3:      Broadcast value(x)
        Round 2
4:      if some value(y) received at least n − f times then
5:          Broadcast propose(y)
6:      end if
7:      if some propose(z) received more than f times then
8:          x = z
9:      end if
        Round 3
10:     Let node v_i be the predefined king of this phase i
11:     The king v_i broadcasts its current value w
12:     if received strictly less than n − f propose(y) then
13:         x = w
14:     end if
15: end for
```

Asynchronous Byzantine Agreement

- Assumption: Messages do not need to arrive at the same time anymore. They have variable delays.

-> Also works, but is a lot more complicated.

-> Algorithm in script is proof of concept, so don't worry about it too much.

->Asynchronity changes messages you have to wait for, but not principle

- Problem: slow! (exponential runtime)

Algorithm 11.21 Asynchronous Byzantine Agreement (Ben-Or, for $f < n/10$)

1: $x_i \in \{0,1\}$ ◁ input bit
2: r = 1 ◁ round
3
4 Broadcast own value
5 Do until converged
6 Wait for enough messages
7
8 If big enough amount agrees, decide and terminate
9
10 If some agree, adapt your value but don't decide yet
11
12 If no popular value, decide randomly
13
14 Broadcast own value
15
16: **until** decided (see Line 8)
17: decision = x_i

Asynchronous Byzantine Agreement with oracle

- Now, if no popular value, all correct nodes will decide on same oracle value.
- Constant runtime
- Problem: oracle does not exist

Algorithm 11.21 Asynchronous Byzantine Agreement (Ben-Or, for $f < n/10$)

1: $x_i \in \{0,1\}$ ◁ input bit
2: r = 1 ◁ round
3
4 Broadcast own value
5 Do until converged
6 Wait for enough messages
7
8 If big enough amount agrees, decide and terminate
9
10 If some agree, adapt your value but don't decide yet
11
12 If no popular value, ask oracle
13
14 Broadcast own value
15
16: **until** decided (see Line 8)
17: decision = x_i

Asynchronous Byzantine Agreement with random bitstring

- New idea: generate a random bitstring and take next value of bitstring instead of asking oracle
- Problem: byzantine nodes know "random" value and can adapt their behavior

Algorithm 11.21 Asynchronous Byzantine Agreement (Ben-Or, for $f < n/10$)

1: $x_i \in \{0,1\}$ ◁ input bit
2: r $= 1$ ◁ round
3
4 Broadcast own value
5 Do until converged
6 Wait for enough messages
7
8 If big enough amount agrees, decide and terminate
9
10 If some agree, adapt your value but don't decide yet
11
12 If no popular value, ask look at bitstring
13
14 Broadcast own value
15
16: **until** decided (see Line 8)
17: decision $= x_i$

Asynchronous Byzantine Agreement with blackboard

- Back to the roots! – shared coin
- Implement it by writing values to a public blackboard, after seeing a certain amount of values nodes decide on coin value
- Constant probability that value is the same for all
- Similar to shared coin but works asynchronously
- Byzantine nodes don't know value of shared coin in advance

Algorithm 11.21 Asynchronous Byzantine Agreement (Ben-Or, for $f < n/10$)

1: $x_i \in \{0,1\}$ ◁ input bit
2: r $= 1$ ◁ round
3:
4: Broadcast own value
5: Do until converged
6: Wait for enough messages
7:
8: If big enough amount agrees, decide and terminate
9:
10: If some agree, adapt your value but don't decide yet
11:
12: If no popular value, generate shared coin
13:
14: Broadcast own value
15:
16: **until** decided (see Line 8)
17: decision $= x_i$

Reliable Broadcast

- **Best effort broadcast**
  - Best effort broadcast ensures that a message that is sent from a correct node v to another correct node w will be received and accepted by w
- **Reliable broadcast**
  - Reliable broadcast ensures that the nodes eventually agree on all accepted messages. That is, if a correct node v considers message m as accepted, then every other node will eventually consider message m as accepted.
- **FIFO (reliable) broadcast**
  - The FIFO (reliable) broadcast defines an order in which the messages are accepted in the system. If a node u broadcasts message m1 before m2, then any node v will accept the message m1 first.
- **Atomic broadcast**
  - Atomic broadcast makes sure that all messages are always received in the same order. So for two random nodes u1 and u2 and two random messages m1 and m2, if u1 sees m1 first, u2 will also see m1 first.

Reliable Broadcast

Algorithm 4.15 Asynchronous Reliable Broadcast (code for node u)

1:	Broadcast own value
2:	If message received from node directly, broadcast it together with your own name
3:	
4:	
5:	If you do not get message from node directly, but from a reasonable amount of others also broadcast with own name
6:	
7:	
8:	If you get enough forwarded messages, accept message
9:	
10:	

Reliable Broadcast

Guarantees:

- If a node broadcasts a value reliably, all correct nodes will eventually accept that value
- If a correct node has not broadcast a value, it will not be accepted by any other correct node
- If a correct node accepts a message from a (byzantine) node, it will be eventually accepted by every correct node

Problem:

- Does only tolerate <= n/5 byzantine nodes
  - This is better if we use the FIFO assumption

Quiz

- Can byzantine nodes collaborate?
  - *Yes*
- Can byzantine nodes forge a sender address?
  - *No, otherwise one could impersonate all correct ones.*
- In all-same validity, is the decision value restricted if not all nodes start with the same value?
  - *No*

Quiz

1.1 Synchronous Consensus in a Grid

In the lecture you learned how to reach consensus in a fully connected network where every process can communicate directly with every other process. Now consider a network that is organized as a 2-dimensional grid such that every process has up to 4 neighbors. The width of the grid is w, the height is h. Width and height are defined in terms of edges: A 2×2 grid contains 9 nodes! The grid is big, meaning that $w + h$ is much smaller than $w \cdot h$. We use the synchronous time model; i.e., in every round every process may send a message to each of its neighbors, and the size of the message is not limited.

a) Assume every node knows w and h. Write a short protocol to reach consensus.

b) From now on the nodes do not know the size of the grid. Write a protocol to reach consensus and optimize it according to runtime.

c) How many rounds does your protocol from **b)** require?

Assume there are Byzantine nodes and that you are the adversary who can select which nodes are Byzantine.

d) What is the smallest number of Byzantine nodes that you need to prevent the system from reaching agreement, and where would you place them?

2.1 What is the Average?

Assume that we are given 7 nodes with input values $\{-3, -2, -1, 0, 1, 2, 3\}$. The task of the nodes is to establish agreement on the average of these values. As always, our system might be faulty - nodes could crash or even be byzantine.

a) Show that in the presence of even one failure (crash or byzantine), the nodes cannot agree on the average of all input values.

Since we cannot establish agreement on the exact value, it would be great to understand how close we can get to the average value. Let us begin by only considering crash failures in the system. Assume that at most 2 of the 7 given nodes can crash.

b) In which range do you expect the consensus value to be?

From now on, we will consider byzantine failures as well. Assume that we have 9 nodes in total. 7 of these nodes are correct and have the input values specified above. The remaining two nodes are byzantine. We will start with a synchronous system.

c) Show that the consensus values can be basically anything now.

d) Suggest a rule that a node could use to locally choose a value as an approximation to the average.

e) What is the range of all possible local approximations of the average?

f) Suggest a validity condition that can be used to determine a consensus value.

Now assume that the system is asynchronous. Keep in mind that the scheduling is worst-case.

g) How does the range of all possible local approximations of the average change in this case?

h) Suggest a new validity condition that can be used to determine a consensus value.