CHALMERS
UNIVERSITY OF TECHNOLOGY

Distributed Computing and Systems
Chalmers university of technology

Distributed Systems

Lab Introduction
Part 2

Charalampos Stylianopoulos

Agenda

- Lab 2 introduction
- Solution costs for Lab 1
- Concurrency in Repy

Notes about the labs

- Lab rooms:
  - Prefer rooms 3354 and 3358
  - That's where TAs will be.
- Check ping pong, lab pages for news, FAQs etc.
  - E.g. How to use curl

```
for i in `seq 1 20`; do
curl -d 'entry=t'${i} -X 'POST' 'http://ip:port/entries'
done
```

This will post entry=t1, entry=t2, ..., entry=t20 to http://ip:port/entries

Notes about the labs(2)

- We expect you to use the RESTful API for browser –vessel communication.
- You also want to extend it and use it for vessel-vessel communication
  - It will make your life easier when the code becomes more complex

Functions	API	Parameters	Returns
Add a new entry	POST /entries	entry : text	Status
Modify an entry	PUT /entries/ entryID	entry : text	Status
Delete an entry	DELETE /entries/ entryID	None	Status
Modify or Delete an entry	*POST /entries/ entryID*	entry : text *delete: logical*	Status
Add entry to neighbor	POST /neighbourID/ entries	*entry:text* *neighbourID:text*	Status

Lab 2 Introduction

RELIABLE CENTRALIZED BLACKBOARD

Distributed Blackboard

- We have a simple working version so far…
  Let's make it better!

- Reliable and consistent
  – Every board shows messages in the same order
  – No message gets lost

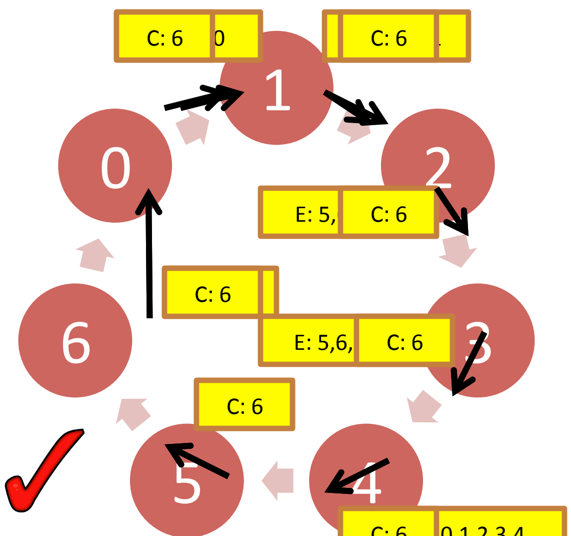
- How? Centralized version!

Distributed Blackboard – Centralized

- Each post is sent to the leader which distributes it to the network
- The leader should be able to handle correctly multiple posts from different nodes…
- But who is the leader?

Leader election on a ring

1. Node 5 initiates the leader election process. It sends ID to its next node in an "Election" (**E**) message.
2. When node 6 receives the message, it appends its ID and forwards the message
3. When the message gets back to the process that started the election:
   i. it elects process with highest ID as coordinator, and
   ii. changes the message type to "Coordination" message (**C**) and circulates it in the ring



Leader Election

- Use the Ring-based Election Algorithm (see Lecture slides on Monday) when starting the board in order to decide the leader
  - Define a ring topology
  - Every node should send **only** to their next neighbor
  - Use a **locally generated random number** as a criterion for selecting the leader (e.g. highest wins)
- Every node acts as an initiator in the beginning .
  - n elections running **concurrently**.
  - Eventually they all agree on the same leader.
- The protocol stars running as soon as the nodes are up .
  - you might have to wait a bit to make sure everyone has booted .
  - How? Use a timer (see slide later on).
- Simplifications (but feel free to impress us):
  - Not dynamic – only run election in the initialization of the protocol
  - Assume that communication between neighbors is reliable

After the election

- After the election, nodes send messages directly to the leader.
- The leader can serve as centralized sequencer:
  - He decides the correct, global order of all messages.
  - Everybody else follows that order.
  - Hint: use the sequence number field. It will now be issued by the leader and it will be unique.

What could go wrong?

Ensure that the ordering of the blackboard entries is the same on every vessel, **even in corner cases.**

Pitfalls:

- **`waitforconn(ip,port,function)`**, spawns a new thread to run **`function`**
- if **`function`** accesses shared variables or data structures, you need to take care of any synchronization needed (i.e. no thread safety when accessing shared data)

What could go wrong?(2)

Possible race conditions:

- On the Non-Leaders: Multiple browsers operating on the same blackboard.
- On the leader: Messages from different blackboards will be handled by different threads.

Hint: Use a lock every time you read/write to a shared resource.

Task 1
Leader Election

- Explain your leader election algorithm
- Use a field in the webpage to show who the leader is and what its random number is
- Discuss the *solution cost* of the leader election algorithm that you use*

*= slide 21

Task 2
Blackboard (centralized)

- Show that concurrent **submissions** do not lead to problems anymore
  - with multiple browsers submitting in the same vessel concurrently, and
  - with multiple vessels submitting concurrently
- Explain where you use locks and why (in case you do use locks)
- Demonstrate the cost of your solution (i.e. cost of a post delivered to all nodes)
- Briefly discuss pros + cons of this design

Optional Tasks

- Note: completely optional
  - We still give you up to 10 points even without this extension
- Handle dynamic networks:
  - What happens if the leader fails while the program is running?
  - What happens if a node during the election cannot reach its next neighbor?
- Concurrently delete/modify entries in the blackboard.

Summary

- ✓ Leader election protocol to decide a leader among the blackboards.
- ✓ Blackboard must now be consistent always.

Optional:

  - Dynamic leader election.
  - Delete/Modify.

- ✓ Deliverables: code + Video(or report).

Deadline: November 24

Agenda

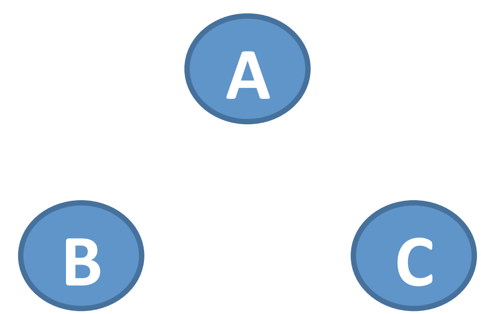
- Lab 2 introduction
- Solution costs for Lab 1
- Concurrency in Repy

Solution cost

- We can measure the cost of a solution (e.g. in Lab 1) in terms of
  - **number of nodes** to which a new post is propagated
  - **payload**: number of blackboard entries per message

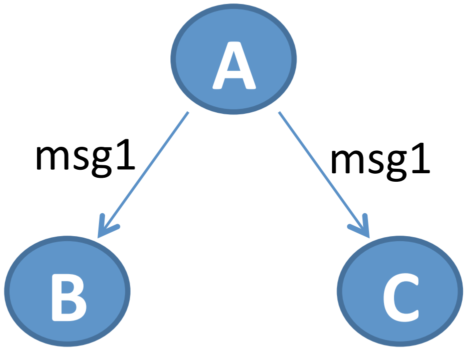
- Overall cost per post = (number_of_nodes - 1) · (payload)

- For example, consider the case of three vessels A, B and C, and the following events:
  - Event 1: User posts “msg1” to vessel A.
  - Event 2: User posts “msg2” to vessel A.

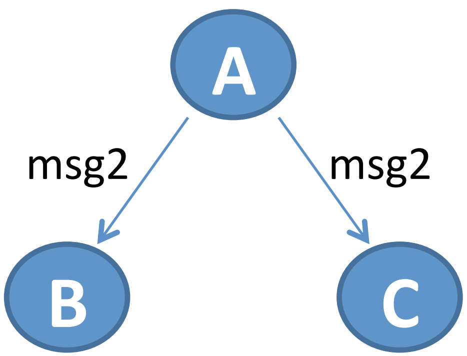


Example: a good scenario

- Upon Event 1,
  Vessel A sends “msg1” to vessels B and C
  - Payload for each message = 1
  - Overall cost = 2



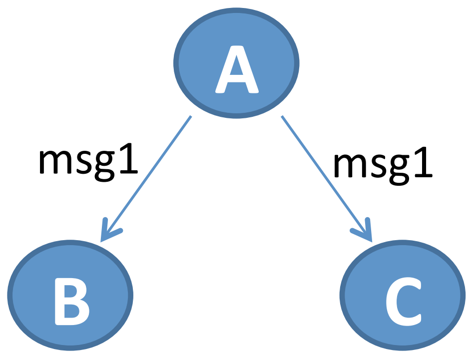
- Upon Event 2,
  Vessel A sends “msg2” to vessels B and C
  - Payload for each message = 1
  - Overall cost = 2



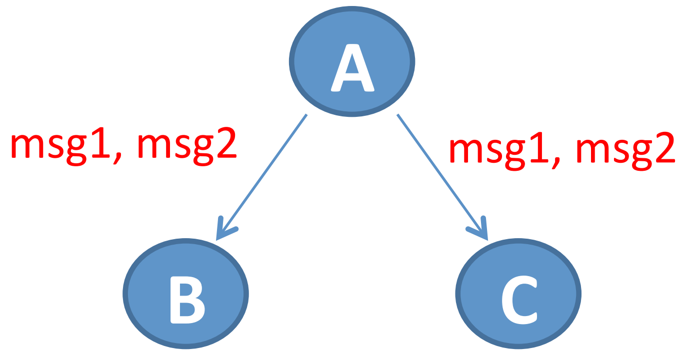
- Overall: *m(n-1)*

Example: a costly scenario

- Upon Event 1, Vessel A sends “msg1” to vessels B and C
  – Payload for each message = 1
  – Overall cost = 2



- Upon Event 2, Vessel A sends “msg1, msg2” to vessels B and C
  – Payload for each message = 2
  – Overall cost = 4



- Overall: *bm(n-1)*
  - board size * msg * (#nodes-1)

Cost of Lab2

- We want you do the same simple communication cost analysis for Lab2:
  - First for the leader election part.
  - Then for the blackboard.

Agenda

- Lab 2 introduction
- Solution costs for Lab 1
- Concurrency in Repy

Concurrency in Repy

- Locks
- Example 1
- Timers
- Example 2

Locks

- **getlock()**
  Returns a lock object that can be used for mutual exclusion and critical section protection.
- **lock.acquire(blocking=1)**
  Blocks until the lock is available, then takes it (lock is an object obtained by calling getlock()).
  - If the optional "blocking" argument is False, the method returns False immediately instead of waiting to acquire the lock; if the lock is available it takes it and returns True, as if it were called with no argument.
- **lock.release()**
  Releases the lock. Do not call it if the lock is unlocked.

Lock example

Write to a log file on each request, and make sure that the file is only handled by **one thread at a time.**

```
def on_request(ip, port, socket, handle, listener):
  mycontext['lock'].acquire()

  # Here's the critical section:
  log_file = open('log.txt', 'a')
  log_file.write('got a request from ' + ip + '\n')
  log_file.close()

  mycontext['lock'].release()

if callfunc == 'initialize':
  mycontext['lock'] = getlock()
  waitforconn(getmyip(), 63153, on_request)
```

Lock example

Write to a log file on each request, and make sure that the file is only handled by **one thread at a time.**

```
def on_request(ip, port, socket, handle, listener):
  mycontext['lock'].acquire()

  # Here's the critical section:
  log_file = open('log.txt', 'a')
  log_file.write('got a request from ' + ip + '\n')
  log_file.close()

  mycontext['lock'].release()

if callfunc == 'initialize':
  mycontext['lock'] = getlock()
  waitforconn(getmyip(), 63153, on_request)
```

Critical Section

Lock example

Write to a log file on each request, and make sure that the file is only handled by **one thread at a time.**

```
def on_request(ip, port, socket, handle, listener):
  mycontext['lock'].acquire()

  # Here's the critical section:
  log_file = open('log.txt', 'a')
  log_file.write('got a request from ' + ip + '\n')
  log_file.close()

  mycontext['lock'].release()

if callfunc == 'initialize':
  mycontext['lock'] = getlock()
  waitforconn(getmyip(), 63153, on_request)
```

Lock initialization.

Lock example

Write to a log file on each request, and make sure that the file is only handled by **one thread at a time.**

```
def on_request(ip, port, socket, handle, listener):
  mycontext['lock'].acquire()

  # Here's the critical section:
  log_file = open('log.txt', 'a')
  log_file.write('got a request from ' + ip + '\n')
  log_file.close()

  mycontext['lock'].release()

if callfunc == 'initialize':
  mycontext['lock'] = getlock()
  waitforconn(getmyip(), 63153, on_request)
```

Will spawn a new thread on a new connection...

Lock example

Write to a log file on each request, and make sure that the file is only handled by **one thread at a time.**

```
def on_request(ip, port, socket, handle, listener):
  mycontext['lock'].acquire()

  # Here's the critical section:
  log_file = open('log.txt', 'a')
  log_file.write('got a request from ' + ip + '\n')
  log_file.close()

  mycontext['lock'].release()

if callfunc == 'initialize':
  mycontext['lock'] = getlock()
  waitforconn(getmyip(), 63153, on_request)
```

..and every new thread will call this method

Will spawn a new thread on a new connection…

Lock example

Write to a log file on each request, and make sure that the file is only handled by **one thread at a time.**

```
def on_request(ip, port, socket, handle, listener):
  mycontext['lock'].acquire()

  # Here's the critical section:
  log_file = open('log.txt', 'a')
  log_file.write('got a request from ' + ip + '\n')
  log_file.close()

  mycontext['lock'].release()

if callfunc == 'initialize':
  mycontext['lock'] = getlock()
  waitforconn(getmyip(), 63153, on_request)
```

Acquire and release the lock to protect access

Timers

- **settimer(waittime, function, args)**
  Sets a timer that when it expires will start a new thread to call a function with a set of arguments.
- **canceltimer(timerhandle)**
  Tries to cancel a timer handle that has not started a thread.

Timer example

Count the number of requests received, and print the result when no requests have been received for 10 seconds.

```
def on_request(ip, port, socket, handle, listener):
  mycontext['counter'] += 1
  canceltimer(mycontext['timer'])
  mycontext['timer'] = settimer(10, stop, [listener])

# This is the function that will run in a separate thread:
def stop(listener):
  print 'got ' + str(mycontext['counter']) + ' requests'
  stopcomm(listener)

if callfunc == 'initialize':
  mycontext['counter'] = 0
  listener = waitforconn(getmyip(), 63153, on_request)
  mycontext['timer'] = settimer(10, stop, [listener])
```

Timer example

Count the number of requests received, and print the result when no requests have been received for 10 seconds.

```
def on_request(ip, port, socket, handle, listener):
  mycontext['counter'] += 1
  canceltimer(mycontext['timer'])
  mycontext['timer'] = settimer(10, stop, [listener])

# This is the function that will run in a separate thread:
def stop(listener):
  print 'got ' + str(mycontext['counter']) + ' requests'
  stopcomm(listener)

if callfunc == 'initialize':
  mycontext['counter'] = 0
  listener = waitforconn(getmyip(), 63153, on_request)
  mycontext['timer'] = settimer(10, stop, [listener])
```

Start a timer, that expires in 10s.

Timer example

Count the number of requests received, and print the result when no requests have been received for 10 seconds.

```
def on_request(ip, port, socket, handle, listener):
  mycontext['counter'] += 1
  canceltimer(mycontext['timer'])
  mycontext['timer'] = settimer(10, stop, [listener])

# This is the function that will run in a separate thread:
def stop(listener):
  print 'got ' + str(mycontext['counter']) + ' requests'
  stopcomm(listener)

if callfunc == 'initialize':
  mycontext['counter'] = 0
  listener = waitforconn(getmyip(), 63153, on_request)
  mycontext['timer'] = settimer(10, stop, [listener])
```

If it expires, it spawns a thread that calls "stop"

Start a timer, that expires in 10s.

Timer example

Count the number of requests received, and print the result when no requests have been received for 10 seconds.

```
def on_request(ip, port, socket, handle, listener):
  mycontext['counter'] += 1
  canceltimer(mycontext['timer'])
  mycontext['timer'] = settimer(10, stop, [listener])

# This is the function that will run in a separate thread:
def stop(listener):
  print 'got ' + str(mycontext['counter']) + ' requests'
  stopcomm(listener)

if callfunc == 'initialize':
  mycontext['counter'] = 0
  listener = waitforconn(getmyip(), 63153, on_request)
  mycontext['timer'] = settimer(10, stop, [listener])
```

A new connection will try to reset the timer

Timer example

Count the number of requests received, and print the result when no requests have been received for 10 seconds.

```
def on_request(ip, port, socket, handle, listener):
  mycontext['counter'] += 1
  canceltimer(mycontext['timer'])
  mycontext['timer'] = settimer(10, stop, [listener])

# This is the function that will run in a separate thread:
def stop(listener):
  print 'got ' + str(mycontext['counter']) + ' requests'
  stopcomm(listener)

if callfunc == 'initialize':
  mycontext['counter'] = 0
  listener = waitforconn(getmyip(), 63153, on_request)
  mycontext['timer'] = settimer(10, stop, [listener])
```

But not thread safe
Use locks for that

More Examples

- https://seattle.poly.edu/wiki/RepyTutorial#RacesSleepandLocksexample1.6