HDU-ITMO Joint Institute
杭州电子科技大学 圣光机联合学院

Distributed Computing
Laboratory work #3
Lamport's logical clocks

Michael Kosyakov
Associate Professor

Denis Tarakanov
Assistant Lecturer

dstarakanov@itmo.ru
ifmo.distributedclass.bot@gmail.com

Classes plan

1. Lamport’s clocks in banking system
2. Communication framework API
3. Task for laboratory work 3
4. Control questions

Lamport's clocks in banking system

- Model of banking system from laboratory work #2
- Problem of calculating the total amount of money at specific time:
  - Incomplete operations
  - Clock of branches can be out of sync
- Real transfers are **not** instantaneous!
- Clocks can drift!

Lamport's clocks in banking system

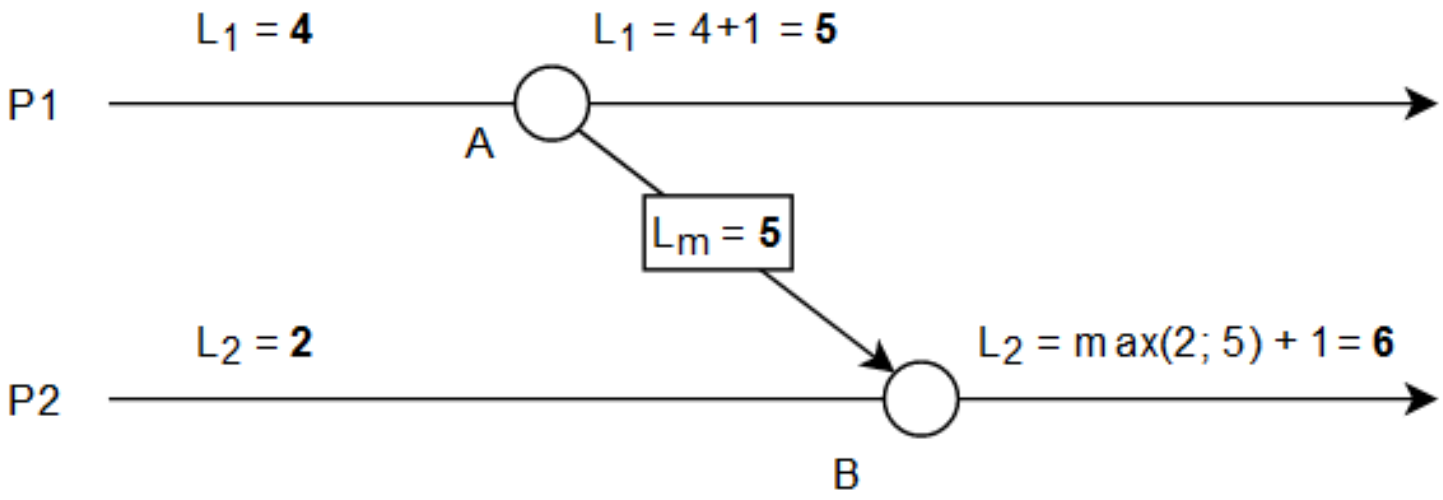
- Logical clocks
  - No physical clocks
  - Counts number of events occurred
  - Relationship "happened-before"
  - Each process has its own clock
  - Lamport's scalar clocks

Lamport's clocks in banking system

- Lamport's scalar clocks rules:
  - **R1.** Each process P_i increments L_i between any two successive events (send, receive, internal)
    - $L_i = L_i + d$, $d > 0$ (typically $d = 1$, L_i initialized with 0)
  - **R2.** Each message piggybacks L_{msg} of its sender at sending time, so that receiving process P_j could update its view of global time. When P_j receives a message with timestamp L_{msg}, it executes the actions:
    - $L_j = \max(L_i, L_{msg})$
    - Executes R1
    - Delivers the message

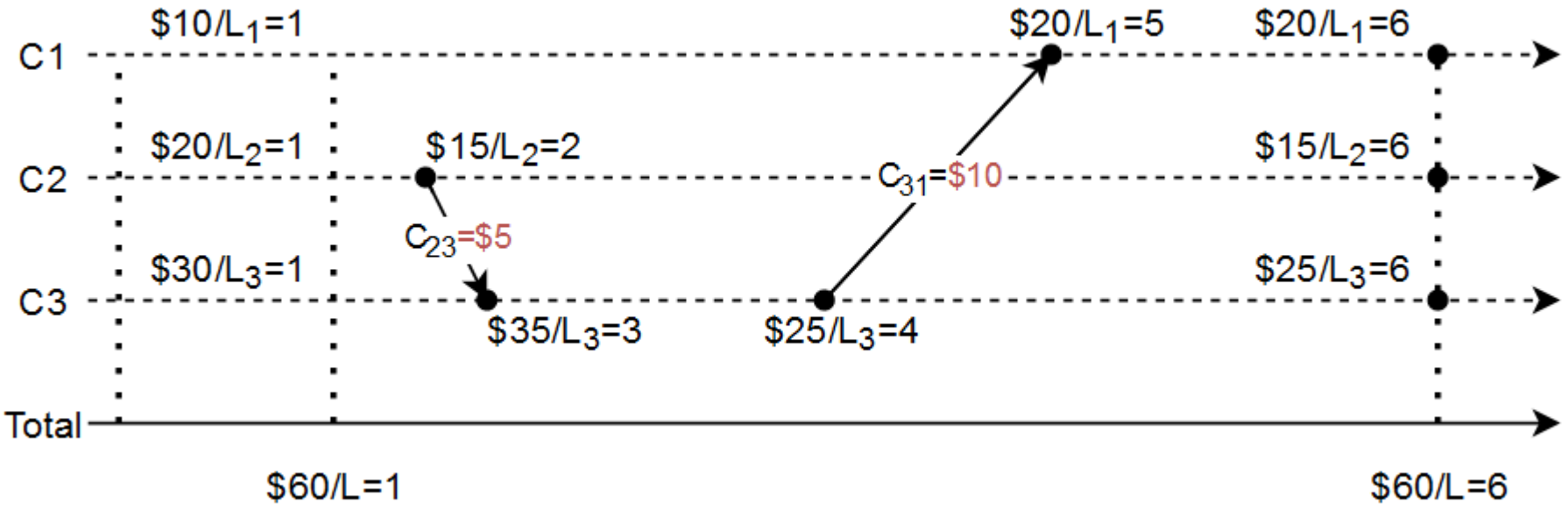
Lamport's clocks in banking system

- Example of rules usage



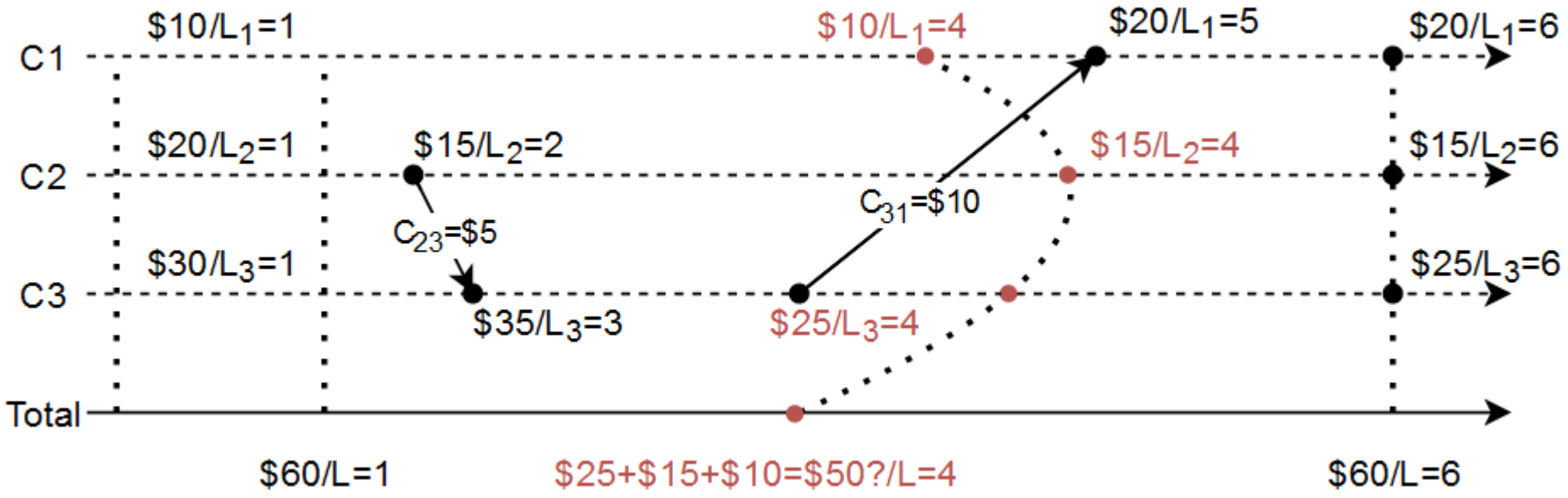
Lamport's clocks in banking system

- Problem of calculating the total amount of money
  - Total amount of money is the same at any point of time



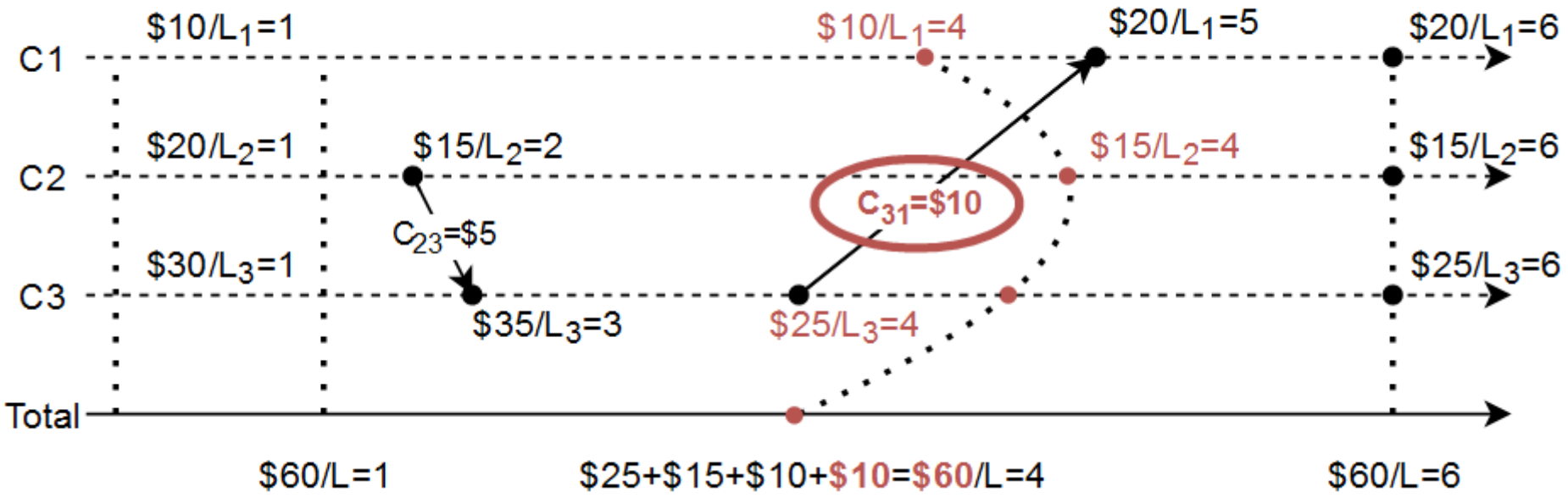
Lamport's clocks in banking system

- Problem of calculating the total amount of money
  - Total amount of money is the same at any point of time?



Lamport's clocks in banking system

- Problem of calculating the total amount of money
  - Pending money
  - Some money still inside the channel!



Communication framework API

- Presented in header files from labs_headers
- Documentation for functions and structures is available as commentaries in header files

- message.h, banking.h, log.h
  - Changes in usage of structures' fields and log formats

Header file "message.h"

```
typedef struct {
    uint16_t      s_magic;
    uint16_t      s_payload_len;
    int16_t       s_type;
    // Usage of field is different!
    timestamp_t   s_local_time;
} __attribute__((packed)) MessageHeader;
```

- Defines a format of message header
- New field to use:
  - s_local_time – time of message sending event, in this laboratory work should be filled with timestamp of **Lamport's clock**
    Must be used by receiver to calculate its clock value!

Header file "banking.h"

- `timestamp_t get_physical_time();`
  - This function **shouldn't** be used in this laboratory work!

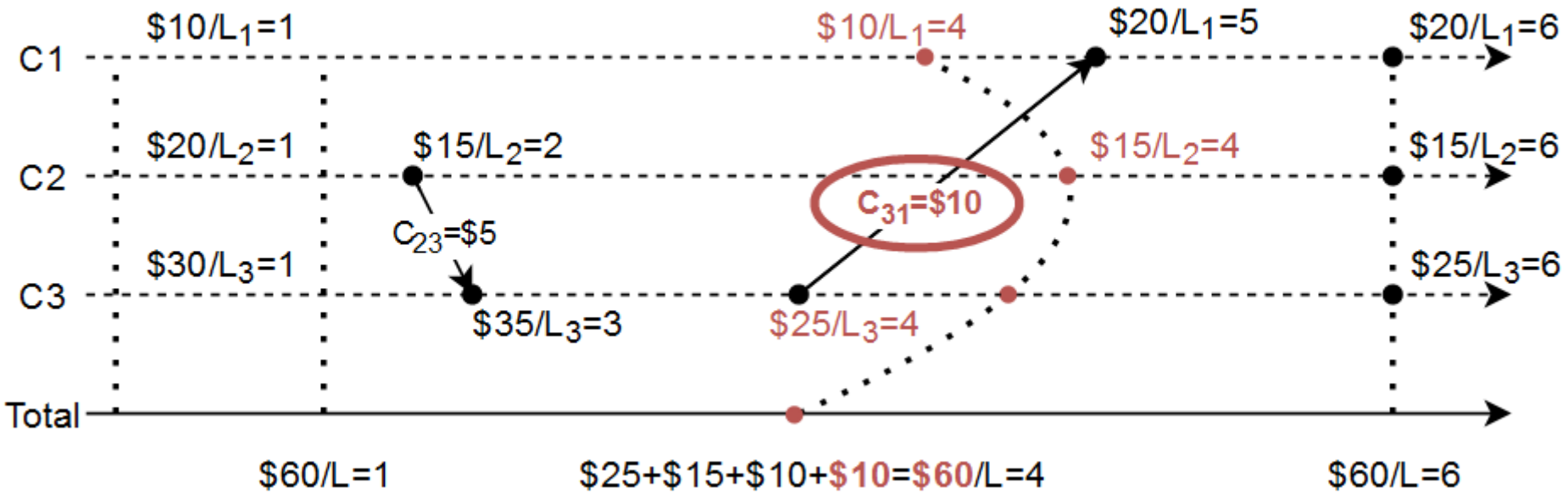
Header file "banking.h"

```
typedef struct {
        balance_t   s_balance;
        timestamp_t s_time;
        // New field to use
        balance_t   s_balance_pending_in;
} __attribute__((packed)) BalanceState;
```

- Defines a format for storing information of process's balance at specific time
- Fields :
  - s_balance – balance of process at time s_time
  - s_time – specified time for this balance (**value of Lamport's clock**)
  - s_balance_pending_in – field is used to present pending money inside channels

Header file "banking.h"

- Pending money
- BalanceHistory should be updated for each moment of time



Header file "banking.h"

- Pending money
- BalanceHistory should be updated for each moment of time

- Process C1 receives $10 from C3
  - $L_1 = 1$; $L_m = 4$

current L_1

Time	0	1	2	3	4	5
C1 balance (pending) $	10 (0)	10 (0)				
C2 balance (pending) $	20 (0)	20 (0)	15 (0)	15 (0)	15 (0)	15 (0)
C3 balance (pending) $	30 (0)	30 (0)	30 (5)	35 (0)	25 (0)	25 (0)

L_m, when money were sent

Header file "banking.h"

- Pending money
- BalanceHistory should be updated for each moment of time

- Process C1 receives $10 from C3
  - $L_1 = 1$; $L_m = 4$
  - receive $L_1 = \max(L_1; L_m) + 1 = 5$

receive L_1

Time	0	1	2	3	4	5
C1 balance (pending) $	10 (0)	10 (0)	?? (?)	?? (?)	?? (?)	20 (0)
C2 balance (pending) $	20 (0)	20 (0)	15 (0)	15 (0)	15 (0)	15 (0)
C3 balance (pending) $	30 (0)	30 (0)	30 (5)	35 (0)	25 (0)	25 (0)

Header file "banking.h"

- Pending money
- BalanceHistory should be updated for each moment of time

- Process C1 receives $10 from C3
  - $L_1 = 1$; $L_m = 4$
  - receive $L_1 = \max(L_1; L_m) + 1 = 5$

receive L_1

Time	0	1	2	3	4	5
C1 balance (pending) $	10 (0)	10 (0)	10 (?)	10 (?)	10 (?)	20 (0)
C2 balance (pending) $	20 (0)	20 (0)	15 (0)	15 (0)	15 (0)	15 (0)
C3 balance (pending) $	30 (0)	30 (0)	30 (5)	35 (0)	25 (0)	25 (0)
Total $	60	60	60	60	**50?**	60

Header file "banking.h"

- Pending money
- BalanceHistory should be updated for each moment of time

- Process C1 receives $10 from C3
  - $L_1 = 1$; $L_m = 4$
  - receive $L_1 = \max(L_1; L_m) + 1 = 5$
  - Update pending, for $t \in [L_m, \ receive\ L_1)$

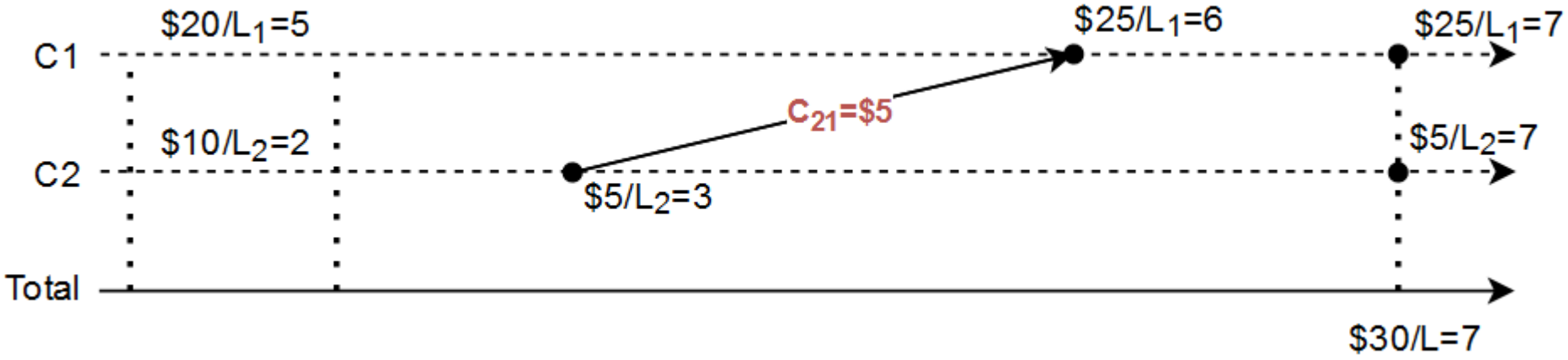
receive L_1 

Time	0	1	2	3	4	5
C1 balance (pending) $	10 (0)	10 (0)	10 (0)	10 (0)	10 (**10**)	20 (0)
C2 balance (pending) $	20 (0)	20 (0)	15 (0)	15 (0)	15 (0)	15 (0)
C3 balance (pending) $	30 (0)	30 (0)	30 (5)	35 (0)	25 (0)	25 (0)
Total $	60	60	60	60	**60**	60

Header file "banking.h"

- Pending money
- BalanceHistory should be updated for each moment of time

- Process C1 receives $5 from C2
  - $L_1 = 5$; $L_m = 3$ => receive $L_1 = \max(L_1; L_m) + 1 = 6$



Header file "banking.h"

- Pending money
- BalanceHistory should be updated for each moment of time

- Process C1 receives $5 from C2
  - $L_1 = 5$; $L_m = 3$ => receive $L_1 = \max(L_1; L_m) + 1 = 6$
  - Update pending, for $t \in [L_m, \ receive\ L_1)$

receive L_1

Time	2	3	4	5	6
C1 balance (pending) $	20 (0)	20 (0)	20 (0)	20 (0)	25 (0)
C2 balance (pending) $	10 (0)	5 (0)	5 (0)	5 (0)	5 (0)
Total $	30	25?	25?	25?	30

Header file "banking.h"

- Pending money
- BalanceHistory should be updated for each moment of time

- Process C1 receives $5 from C2
  - $L_1 = 5$; $L_m = 3$ => receive $L_1 = \max(L_1; L_m) + 1 = 6$
  - Update pending, for $t \in [L_m, \ receive\ L_1)$

receive L_1

Time	2	3	4	5	6
C1 balance (pending) $	20 (0)	20 (5)	20 (5)	20 (5)	25 (0)
C2 balance (pending) $	10 (0)	5 (0)	5 (0)	5 (0)	5 (0)
Total $	30	30	30	30	30

Header file "log.h"

- Important changes to log formats
  - First parameter of formats should be current logical time of the process

Task for laboratory work

- Package for preparing laboratory work has following parts:
  - libdistributedmodel.so – communication framework, presenting model of distributed system
  - labs_headers/ – directory with headers, which are presented API of communication framework
  - Makefile – file of build automation system for laboratory work to simplify local testing
  - **lab.c – template of source code file, which should be implemented by students**
  - template.docx – template for report with control questions

Task for laboratory work

- Goal: implement banking system
- Task: students should implement functions parent_work(), child_work() and transfer()
- Students should implement Lamport's clocks by themselves

- Workflows of parent and child processes are the same as in laboratory work #2
- For each purpose as time should be used timestamps of Lamport's clocks (instead of physical clocks)
- Don't forget to fix BalanceHistories and fill s_balance_pending_in field!

Task for laboratory work

- Lamport's clocks implementation
  - Each process (including parent) has its own clocks
    You can define and use variable for clocks as global variable, globally defined variables in lab.c will have different values for each process
  - Timestamp of Lamport's clocks should be used in log formats, as timestamp in each message and as timestamp in BalanceHistories
  - Clocks should be initialized as **0**
  - Process increases its clock by **1**
  - Processes **don't have internal events** from point of clocks
  - Processes should modify their Lamport's clock on each send and receive events (**before** handling received message in case of receive event)
  - In case of sending multicast messages (STARTED, DONE, STOP) clocks are increased **only by 1** regardless of processes count

Compilation

- Makefile is provided with laboratory work to simplify its compilation
- Default target for make will compile "lab" binary executable file with program
- No compilation warnings are allowed
- C99

- libdistributedmodel.so supports only x64 Linux-based operating systems!
- Recommended environment for compilation:
  - Ubuntu 16.04 or newer (or other Linux-based operating system)
    - Virtual machine can be used (VMware Player, Virtual Box or other)
  - clang3.8 or newer

Execution

- To execute program, you must set environment variable LD_LIBRARY_PATH, e.g.:

```
export LD_LIBRARY_PATH=
        "$LD_LIBRARY_PATH:/path/to/directory/with/libdistributedmodel"
```

- To execute program use following command:

```
LD_PRELOAD=/path/to/lib/libdistributedmodel.so
                                ./lab -l 3 -p X B1 B2 <..> BX
```

- Where "l" key – for number of laboratory work (work 3)
- And "p" key – for number of child processes [1..9]
- B1, B2, <…> BX – is starting balances of child processes, for example, for 5 child processes:

```
./lab -l 3 -p 5 10 20 30 40 90
```

- LD_PRELOAD environment variable should be set in mentioned way to be applied only for execution of lab binary

Bot usage

1. Pack source code to archive with the name labN.tar.gz, where N – is num of laboratory work; this archive should contain single file **lab.c**
   - Example of command, to prepare archive:
     tar czf **lab3.tar.gz** lab.c
   - For MacOS COPYFILE_DISABLE variable should be set:
     COPYFILE_DISABLE=1 tar czf **lab3.tar.gz** lab.c

2. Send mail to address ifmo.distributedclass.bot@gmail.com
   - Archive should be added to mail as attachment
   - Subject of letter should have following format (only Latin symbols):

HDU Laboratory work #N Student_Name Student_HDU_ID

Bot usage

3. If bot answers with message “Failed” – you should find the reason, fix issue and send code again
   - Also, bot can send the logs of execution with instructions, how to open them; it can be useful for debug
   - If bot doesn’t answer for 10 minutes, feel free to send message again

4. If bot answers with message “Passed” – you can prepare a report and answer on control questions

Control questions

Control questions should be answered in the report:

1. Lamport's clocks. What does it mean, that these clocks aren't strongly consistent?
2. What will happened, if parent process's messages (e.g., TRANSFERs from parent) aren't marked with Lamport's clock timestamp?
   Modify your model and explain results. Attach resulted table to the report.
3. Make an experiment. What will happened if we will not use s_balance_pending_field?
   Modify your model and explain results. Attach resulted table to the report.

Task and report

For this laboratory work students should:

1. Prepare source code for the task
2. Get reply “Passed” from the bot
3. Answer on control questions
4. Fill the report (see template.docx, all fields must be filled) and send it to dstarakanov@itmo.ru before deadline

Thank you!