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

Distributed Computing Laboratory work #4

Distributed mutual exclusion algorithm

Michael Kosyakov
Associate Professor

Denis Tarakanov
Assistant Lecturer

dstarakanov@itmo.ru

ifmo.distributedclass.bot@gmail.com

Classes plan

1. Distributed mutual exclusion algorithm
2. Communication framework API
3. Task for laboratory work 4
4. Control questions

Distributed mutual exclusion algorithm

- Critical section (CS) is a part of a program with access to shared resources, that should be used exclusively

- Mutual exclusion is a requirement that while a process is executing CS, no other process is executing the same CS

- Problems of mutual exclusion in distributed systems:
  - Lack of shared memory
  - Incomplete knowledge of system state

Distributed mutual exclusion algorithm

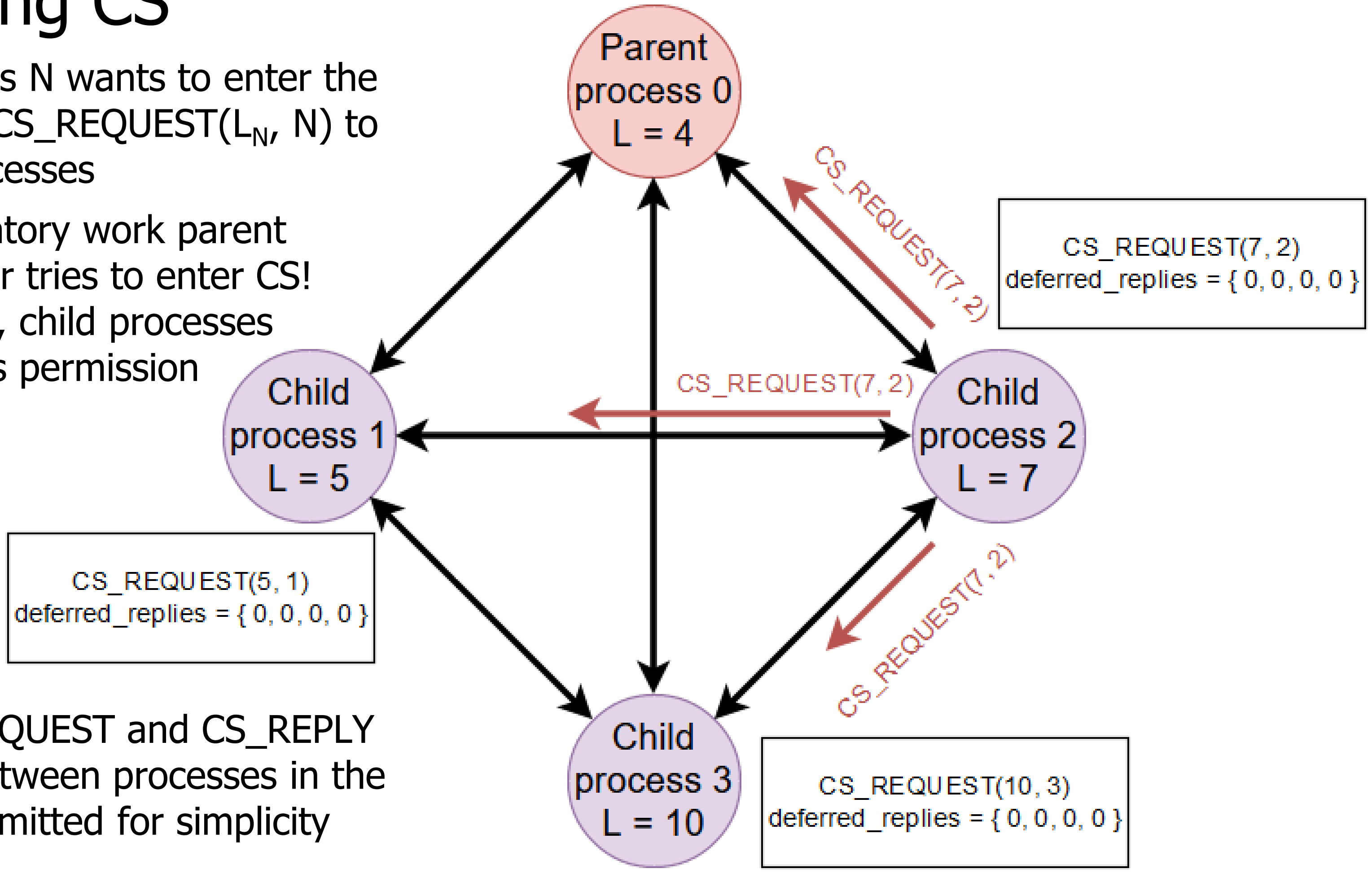
- Ricart–Agrawala algorithm
- Model of distributed system from laboratory work #1
- Lamport’s clock implementation from laboratory work #3

Distributed mutual exclusion algorithm

- Ricart–Agrawala algorithm
  - Two types of messages: CS_REQUEST and CS_REPLY
  - The process needs to collect permissions from all other processes in order to enter the CS
  - Deferred replies
  - Timestamps of Lamport's clocks are used to resolve conflict between processes
  - The algorithm requires 2(N−1) messages per CS invocation
  - No FIFO requirements for channels

Distributed mutual exclusion algorithm

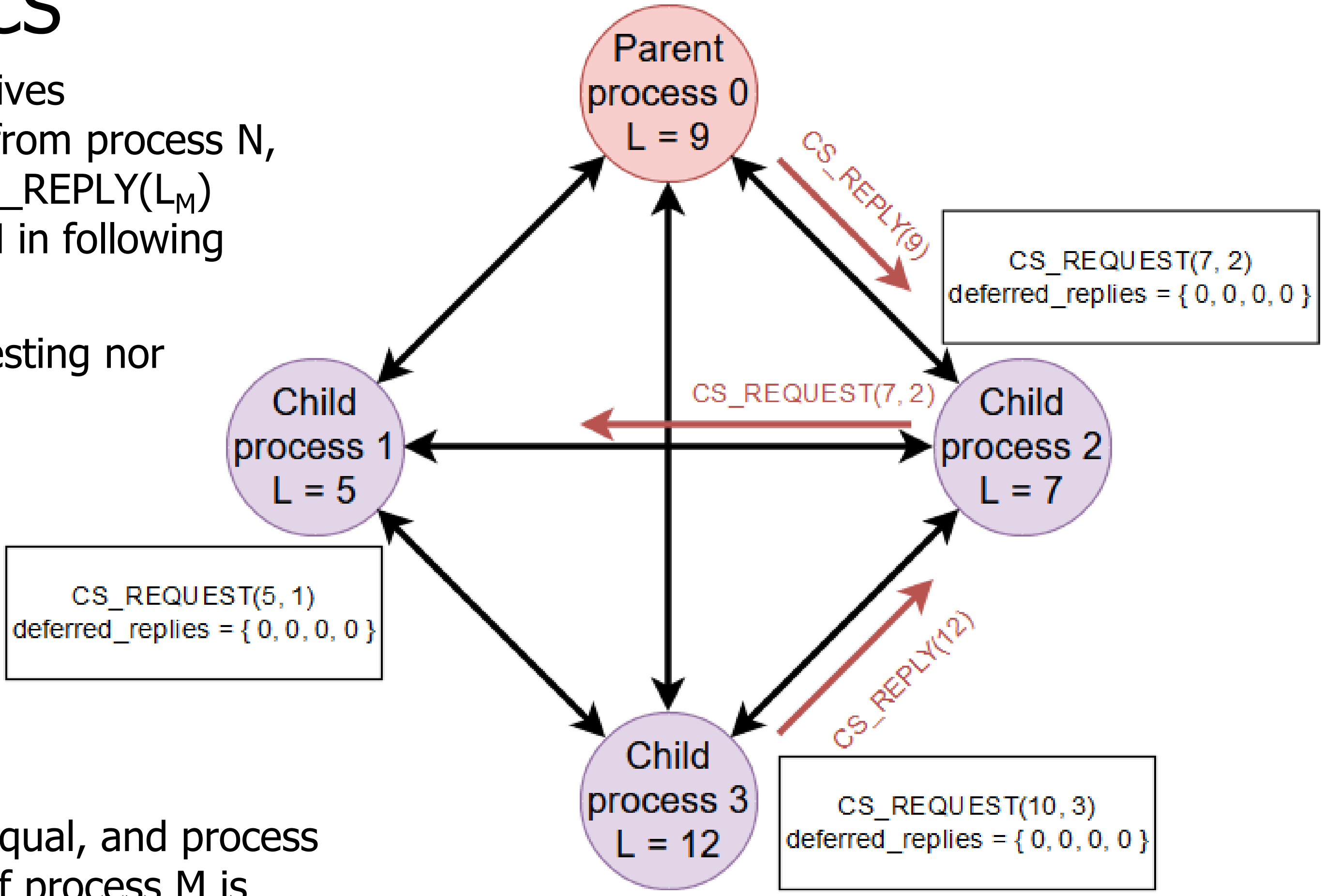
- **Requesting CS**
  - When process N wants to enter the CS, it sends CS_REQUEST(L_N, N) to all other processes
  - In this laboratory work parent process never tries to enter CS! Nevertheless, child processes should get its permission
  - Other CS_REQUEST and CS_REPLY messages between processes in the system are omitted for simplicity


Parent process 0 L = 4
Child process 1 L = 5
Child process 2 L = 7
Child process 3 L = 10
CS_REQUEST(7, 2)
CS_REQUEST(7, 2)
CS_REQUEST(7, 2)
CS_REQUEST(7, 2) deferred_replies = { 0, 0, 0, 0 }
CS_REQUEST(5, 1) deferred_replies = { 0, 0, 0, 0 }
CS_REQUEST(10, 3) deferred_replies = { 0, 0, 0, 0 }

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

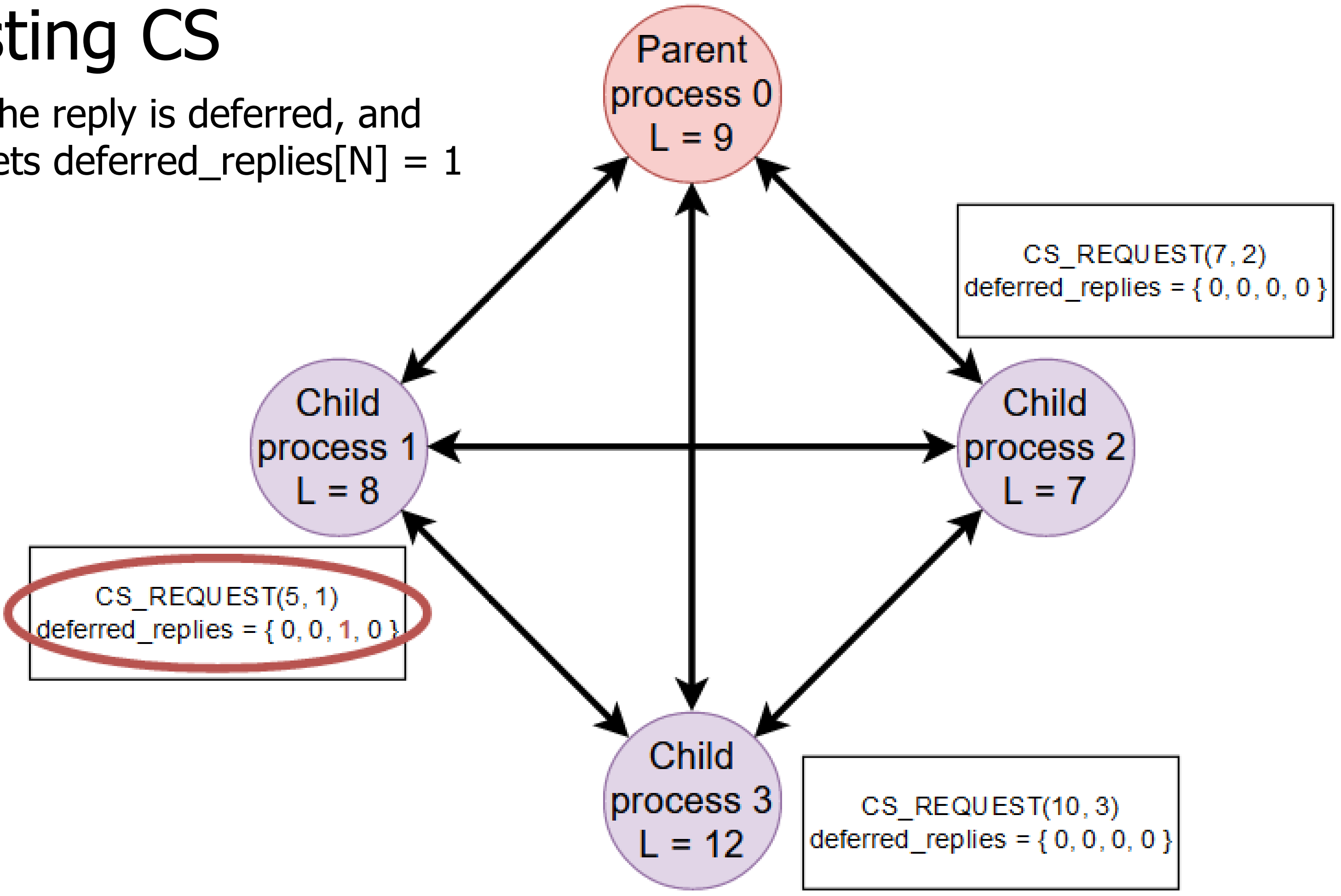
Distributed mutual exclusion algorithm

- Requesting CS
  - When process M receives CS_REQUEST(L_N, N) from process N, process M sends a CS_REPLY(L_M) message to process N in following cases:
    - If it is neither requesting nor executing the CS
    - If it is requesting the CS and the timestamp of its own CS_REQUEST is larger than the timestamp of process N request
    - If timestamps are equal, and process identifier (self_id) of process M is greater then process identifier of process N



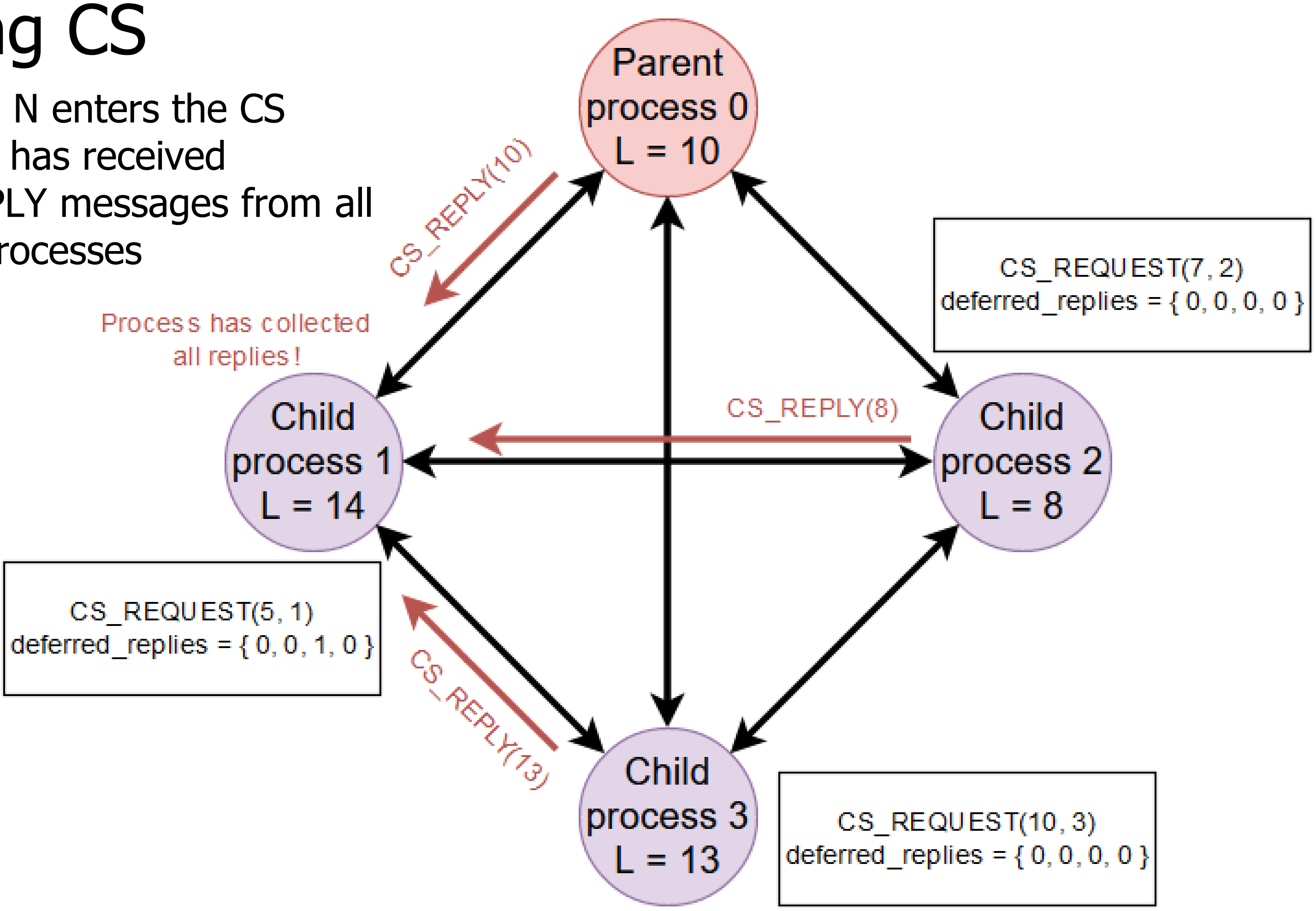
Distributed mutual exclusion algorithm

- Requesting CS
  - Otherwise, the reply is deferred, and process M sets deferred_replies[N] = 1


Parent process 0 L = 9
Child process 1 L = 8
Child process 2 L = 7
Child process 3 L = 12
CS_REQUEST(7, 2) deferred_replies = { 0, 0, 0, 0 }
CS_REQUEST(5, 1) deferred_replies = { 0, 0, 1, 0 }
CS_REQUEST(10, 3) deferred_replies = { 0, 0, 0, 0 }

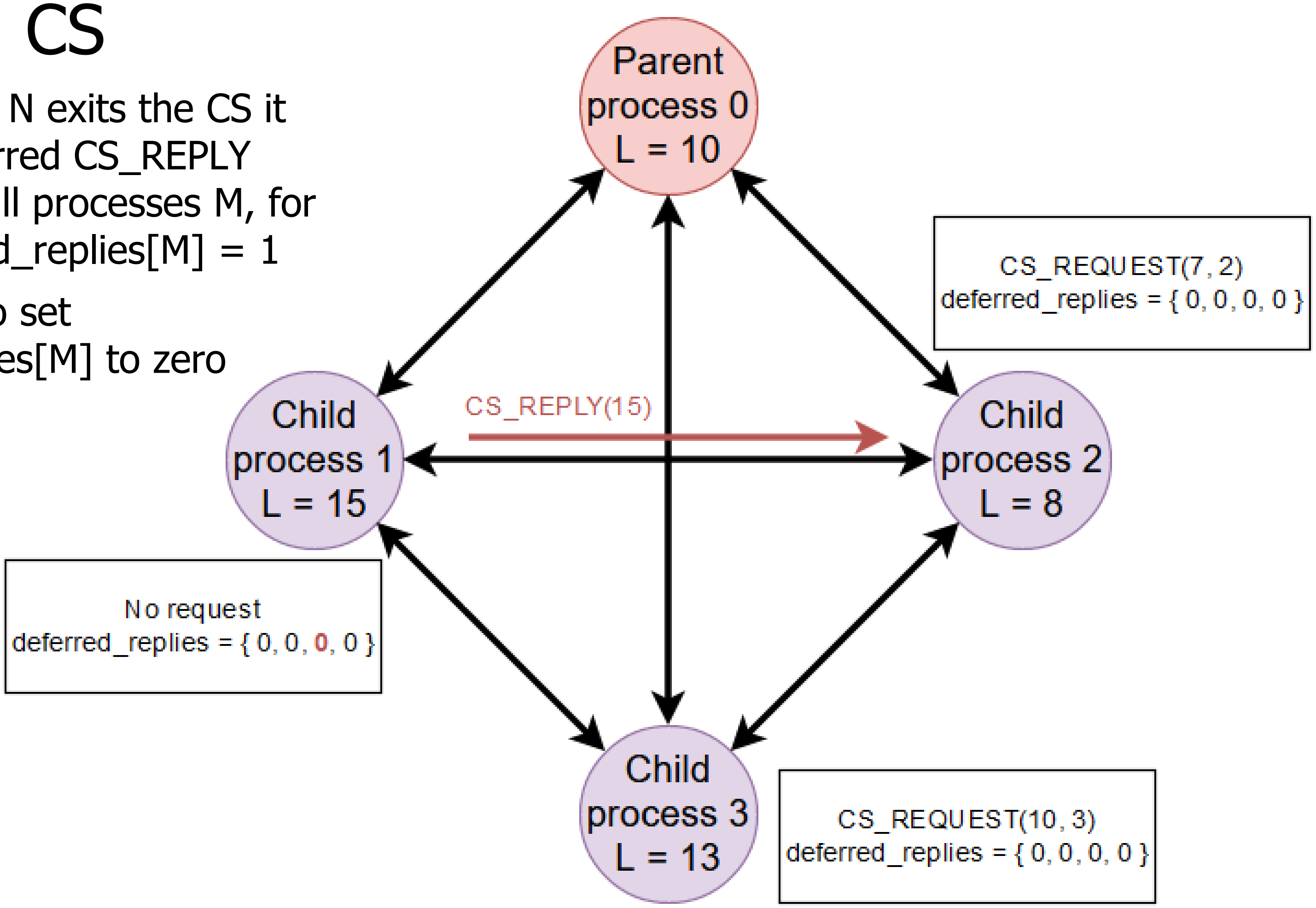
Distributed mutual exclusion algorithm

- Entering CS
  - Process N enters the CS when it has received CS_REPLY messages from all other processes


Parent process 0 L = 10
Child process 1 L = 14
Child process 2 L = 8
Child process 3 L = 13
CS_REPLY(10)
CS_REPLY(8)
CS_REPLY(13)
Process has collected all replies!
CS_REQUEST(7, 2) deferred_replies = { 0, 0, 0, 0 }
CS_REQUEST(5, 1) deferred_replies = { 0, 0, 1, 0 }
CS_REQUEST(10, 3) deferred_replies = { 0, 0, 0, 0 }

Distributed mutual exclusion algorithm

- Releasing CS
  - When process N exits the CS it sends all deferred CS_REPLY messages to all processes M, for which deferred_replies[M] = 1
  - Don’t forget to set deferred_replies[M] to zero


Parent process 0 L = 10
Child process 1 L = 15
Child process 2 L = 8
Child process 3 L = 13
CS_REPLY(15)
CS_REQUEST(7, 2) deferred_replies = { 0, 0, 0, 0 }
No request deferred_replies = { 0, 0, 0, 0 }
CS_REQUEST(10, 3) deferred_replies = { 0, 0, 0, 0 }

Communication framework API

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

- process.h, message.h, log.h
  - New types, fields, functions and log formats
- banking.h
  - This header file isn't used in this laboratory work

Header file “process.h”

```
struct child_arguments {
    local_id self_id;
    int count_nodes;
    uint8_t balance;     // Zero in this work
    bool mutex_usage;    // New field to use
};
```

- Defines required information about child process
- New field to use:
  - mutex_usage – if field value is true – mutual exclusion algorithm should be used to access CS
    Check slide 22 with example of usage

Header file “message.h”

- New messages are available for this laboratory work!
- CS_REQUEST message
  - Should be sent from child process N to all other processes, when process N wants to enter CS
  - Payload should be empty; payload length should be 0
- CS_REPLY message
  - Should be sent by process to other child process, meaning depends on mutual exclusion algorithm
  - Payload should be empty; payload length should be 0

Header file “message.h”

- `int receive_any(Message * msg);`
  - Receive a message from any of the processes
  - Function receives the first message, which it finds in communication channels of this process
  - Arguments:
    - msg – pointer to message structure, allocated by user, where received message should be stored
  - Return value:
    - **Function returns id of process, which sent the message**

Header file “log.h”

- `void print(const char * s);`
  - Presents shared resource, this function should be executed in CS in this laboratory work
  - Prints string with required special format
  - s should have format, described on the next slide
  - For all other logs shared_logger() function should be used
  - **It is forbidden to call print() multiple times during one CS invocation!**

  - Arguments:
    - s – string to be printed in CS

Header file “log.h”

-
```
char * log_loop_operation_fmt =
    "process %1d is doing %d iteration out of %d\n";
```

  - New log format, which should be used with print() function
  - First parameter of format should be process identifier
  - Second parameter is current number of times, the process has already called print() function (should start from 1)
  - Third parameter – is number of how many times process should call print() function in total

  - Meaning of parameters will be explained in the task

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 Ricart–Agrawala algorithm
- Task: students should implement functions parent_work() and child_work()
- Implementation of Lamport's clock should be taken from laboratory work #3

- Parent and child processes have different workflow
- For each purpose as time should be used timestamps of Lamport’s clocks

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
  - 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, **CS_REQUEST**) clocks are increased **only by 1** regardless of processes count
  - **CS_REPLY** should be marked with timestamp of Lamport’s clock as usual

Task for laboratory work

- Phase 1 of parent process
  - Process must receive all STARTED messages from child processes
- Phase 2 and 3 of parent process
  - Process must receive all DONE messages from child processes – after that parent process must finish its execution
  - If parent process has received CS_REQUEST message – it should answer with immediate CS_REPLY message
  - Parent process never tries to enter CS
- No logs are printed by parent process

Task for laboratory work

- Phase 1 of child process same as in previous laboratory works
  - Sending STARTED message and collecting all STARTED messages
- Phase 2 of child process
  - Useful work of child process is mutual exclusive access to print() function
  - Each child process should enter critical section exactly (self_id * 5) times and call print() function with log, mentioned on slide 15, as argument
  - For this purpose, process should send CS_REQUEST message each time it should call print() and handle CS_REQUEST and CS_REPLY messages of other child processes
  - It is forbidden to call print() multiple times during one CS invocation
  - After process calls print() exactly (self_id * 5) times – it can proceed to phase 3
  - Don’t forget, that during Phase 2 child process can receive DONE messages

Example

- Example of phase 2 main loop

```
bool mutex_usage = args.mutex_usage;
int count_prints = self_id * 5;
<…>
for (int i = 1; i <= count_prints; i++) {
    if (mutex_usage) cs_request(/* args */);

    snprintf(buf, BUF_SIZE,
        log_loop_operation_fmt, self_id, i,
        count_prints);
    print(buf);

    if (mutex_usage) cs_release(/* args */);
}
```

Task for laboratory work

- Phase 3 of child process
  - Process sends a message of type DONE to all other processes, including the parent and prints log of format `log_done_fmt`
  - Process waits for DONE messages from all other child processes and prints log of format `log_received_all_done_fmt` after all DONE messages were received
  - During this phase process should continue to handle CS_REQUEST messages from other child processes with immediate CS_REPLY response

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

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

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 4 -p X -m
```

- Where “l” key – for number of laboratory work (work 4)
- “p” key – for number of child processes [1..9]
- And “m” key – if this key is set, mutex_usage will have value true
- 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 **lab4.tar.gz** lab.c
   - For MacOS COPYFILE_DISABLE variable should be set:
     COPYFILE_DISABLE=1 tar czf **lab4.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

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

Control questions

Control questions should be answered in the report:

1. Ricart–Agrawala algorithm. Is your implementation satisfying properties of safety, liveness and fairness? Explain why.

2. Why child process during phase 3 should continue to handle CS_REQUEST messages? What will happened otherwise?

3. Make an experiment. What will happened if execute program without key “m”? Add results of execution to the report and explain them.

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

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!