02335 Operating Systems

OS Challenge – Part 2

Document version 2023-02

Novermber 1, 2023

1 Introduction

This is the second part of the **mandatory** group project for the course 02335 Operating Systems. The project has the form of a challenge for which this document specifies the objective and rules. Note that this document may change in which case updates will be announced on DTU Learn.

1.1 Important Dates and General Information

Here, you can find a summary of the deadlines and milestones for quick reference.

- **Challenge Kick-off: Wednesday August 30, 2023**
- **Group Information Submission: Wednesday September 6, 2023**
- **Milestone Submission: Friday October 6, 2023**
- **Final Submission: Friday November 24, 2023**

The learning objectives, the project specification, details about the execution environment and requirements for the milestone submission have been given in Part 1.

This document focuses on the requirements and delivery information for the final submission.

1.2 Document Structure

- **Section 1** summarises the deadlines and document structure.
- **Section 2** describes the requirements for the final submission.
- **Section 3** gives a few amendments to the specifications in Part 1.
- **Section 4** specifies the deliverables to be handed in for the final submission.
- **Section 5** lists the criteria against which the challenge will be evaluted.
- **Section 6** answers any frequently asked questions about Part 2.
- **Section 7** summarises this document's version history. If a new version is released, a summary of changes will be available in this section.

2 Requirements for final submission

In the final submission, the reverse hashing server should be optimized in various ways in order to improve its performance. By performance is here ment the `score` provided by the client program taking both request response times and priorities into account.

For the final submission, some parts must be done as collective group work, whereas others must be done individually.

The various parts must be documented by a project report and accompanying server versions as described in the following.

2.1 Group work: Extended Server [mandatory]

In order to prepare for various optimization experiments, the group must refactor the basic server from the milestone such that the handling of connections is separated from the reverse hashing work.

In particular, the server process should feature two threads:

- One thread repeatedly accepting connections from clients and transferring them to a FIFO request queue.
- One thread repeatedly receiving requests from the request queue, doing the reverse hashing and sending back the response on the corresponding connection.

The request queue between the two threads will be a shared state component and must be properly protected by synchronization.

The implementation of the threads and the request queue must be described.

Also, you should explain how your `Makefile` is constructed and how it builds your server.

As part of your design, you will have decide in which thread the request decoding should take place and argue for this.

It should discussed whether this refactoring will have any effect on the server performance and one or more experiments should be set up testing this. The results of the tests should be presented and explained in the report.

The extended server should be used as a baseline for evaluating the individual optimizations.

2.2 Individual Experiments [mandatory]

Based on the extended server, each group member should be responsible for carrying out and documenting a particular performance optimization experiment. The group members may select among the following given optimization options:

A **Concurrent execution** of different requests

B **Parallelization** of each individual request

C **Caching** of request results

D **Out-of-order request execution** (prioritization)

E Any other server optimization that you may come up with.

No two members of a group may choose the same optimization option.

Each optimization experiment must be documented by a report section with the following structure:

Idea
The overall optimization idea must be described. Why might this improve the performance?

Estimation
An estimate of the performance gain should be made (prior to any coding). This should be done for two client request patterns:

- An **optimistic pattern** in which the effect of the approach is expected to be maximal and
- a **pessimistic pattern** in which the effect might be minimal (perhaps even negative).

Each pattern must be characterized and an estimate of the expected performance gain for each of these must be made and explained.

Implementation
The implementation of the optimization idea must be carefully described. It should focus on the changes made with respect to the extended server. The description should in particular address concurrency related issues related to shared state components.

Tests
The actual tests carried out must be described. What execution environment was used and on which hardware? What client configuration parameters have been used to simulate the optimistic and the pessimistic usage patterns? What are the results for each of these compared to the extended server?

Conclusion

In the conclusion the results should be compared to the estimates. Do they correspond? In case of deviations from the expected results, you should try to explain these. It should also be discussed which other optimzation options this one may work well together with and which not.

Notice that each experiment should focus on one optimization option only.

It is understood that the responsible group member is the main contributor to the experiment and that this is reflected in the commit history. It is, however, allowed to discuss the experiment with other members of the group.

2.3 Group work: Combined optimizations [optional, bonus]

In addition to the individual experiments, the group as such may try to combine two or more optimizations and see, whether the performance can benefit from both/all of these.

If the combination is well done, a bonus will be given to all group members. Note, however, that the main weight will be on the individual experiments. Therefore, it is possible to get at good individual result for the OS challenge without this optional combination part. Also, the combination part cannot contribute negatively to the individual evaluations.

If pursued, the combination must be carried out and documented in the same way at the individual optimization experiments described above.

2.4 Competition participation [optional]

It will be possible for qualified groups to participate in the competition for the best server carried out in course 02159 Operating Systems from where the infrastructure is borrowed.

In order to enter the competition, the group must be approved by the teachers. Further information about this options will be given separately on Learn.

Note that any participation in the competition will have no impact on the evaluation of the OS Challenge.

3 Amendments

3.1 Specification and Rules

The server specification from Part 1 is unchanged. Regarding the rules, however, you are now allowed (and expected) to use the `pthread` library.

3.2 Groups

The 3-4 person groups formed for the milestone are supposed to be continued for the final submission.

Should you have any changes to the groups, please get in contact with the teachers as soon as possible.

4 Deliverables

4.1 Code

The following server versions must be available within your git repository.

- The **extended server** [on default branch]
- The **optimized server** for each optimization experiment [on dedicated branches]
- Possibly a **multi-optimized server** [on default/dedicated branch(es)]

Each individual optimization experiment is expected to be developed on a dedicated branch branching out from the extended server version on the default branch. The name of the branch should be `opt-`X, where X is the option letter, e.g. `opt-B`.

The commit time of all server versions must be within the assignment deadline.

Remember that for each server version submitted, it must be possible to run the optimized server in the execution environment using the procedure described in Section 3.3.1 of Part 1.

4.2 Version info file

The various server versions must be identified by a simple text file `<group-name>-final.txt` that should have the following contents:

- A line with your `<group-name>`
- A line with the link to the GitLab repository
- A line with the `<git hash of the extended server>` (on the default branch)
- A line for each group member of the form:

  ```
  <student no>, <full name>, <branch name>, <git hash of optimized server>
  ```

- A line for each multi-optimzed server (if any) of the form:

  ```
  <descriptive name>, <branch name>, <git hash of multi-optimized server>
  ```

This file must be uploaded to the assignment on DTU Learn.

4.3 Report

A pdf report must be handed in on the assignment on DTU Learn. The report should comprise:

- A front page with information about course, assignment, date, group no., and participants.
- A brief introduction including the distribution of optimization experiments among the group members.
- A section on the extended server as described above.
- A section for each optimization experiment following the scheme described above. The section should be written by the responsible group member.
- Possibly a section on optimization combination following the same scheme.
- A brief conclusion. What have you learned from the challenge?

There is no page requirement, but try to keep the report within 15–20 pages.

5 Evaluation Criteria

The challenge will be evaluted based on

- The quality of the milestone version.
- That the extended server works 100 % correctly and is reasonably efficient.
- That the extended server and the experiments made with it are well described.
- That the individual experiments are well thought out, carefully carried out and succintly described according to the given scheme.
- That all server versions run in the execution environment.
- That all server code is well documented.
- That any combined optimization is well described and experiments with it are well documented.
- That solutions do not suffer from concurrency issues (race conditions, deadlocks, live-locks).

6 Frequently Asked Questions

See also Part 1.

6.1 How do we start with the extended server?

You may find some inspiration in Section 2.4 of *Modern Operating Systems*. [Or 2.3 in the 4th edition.]

6.2 Can I make more than one optimization?

No. For the individual part, each group member must stick to one of the given optimizations and do that well.

However, for multi-optimization work, you may choose any optimization you like.

6.3 Do we need to read the request incrementally?

Yes. In general, on a TCP connection you cannot assume that all bytes sent together will also be received toegether. Therefore, when you read the request from the socket, you must be prepared to receive less bytes than expected and the read the rest afterwards. This should alwayst be your standard way of reading from a socket.

6.4 The server is little endian, right?

Although not so common, big endian machines are still around. Therefore, you should generally not assume that values are stored in memory using a particular endianess. Instead proper conversion functions like `htole64` etc. should be used.

However, your server will indeed be tested on a little endian machine and you may assume that. However, this assumption must be made explicit by checking for little endianness at compile time. This may be done by including the `check_little_endian.h` file provided on DTU Learn.

If this check is not made, we will assume that your solutions is endianness agnostic. If properly done, you will get a little bonus.

7 Document Version History

- Version 2023-01, September 26:
  - Initial version of Part 2
- Version 2023-02, November 1:
  - Second version of Part 2. New FAQ entries.