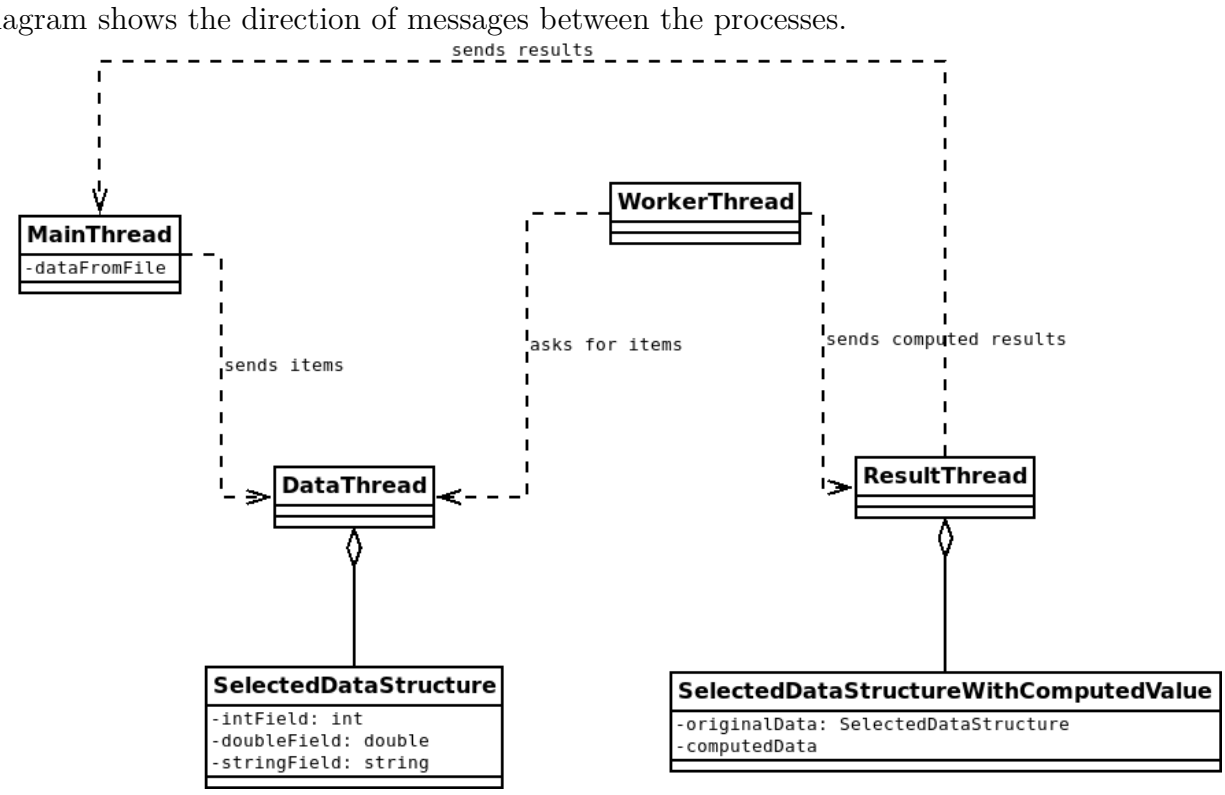
L2. Distributed memory

Task

Modify L1a program so that there is **no shared memory left** and all communication is done using messages instead of calling monitor methods.

Data files are the same as for L1; result files should be the same as for L1.

A generalized of the program is shown in the picture below. There are 4 roles of processes: one main process (main thread can be used for it), one data manager, one result manager and multiple workers. Main process and workers send messages-requests to data thread. The requests are to insert an item or to remove an item from the array owned by the data process. Workers receive items from the data process, calculate a function on the item and if it matches your filter criteria, sends it to a result process that saves received items in a sorted array. When all items are processed, the result process sends all items that it has received to the main process. Data process can only receive requests to modify the internal array, it should not send anything without being asked. Diagram shows the direction of messages between the processes.



Main process works as follows:

1. Reads the data files to any data structure;
2. Spawns processes:
   - a selected amount for worker processes $2 \leq x \leq \frac{n}{4}$ (n — amount of data in your file).
   - one process to control the data array;
   - one process to control the result array.
3. Sends all items from the file to the data process one by one.
4. Receives the results from the result process.

5. Outpus the results to a file as a table.

Worker processes work as follows:

1. Sends a remove request to the data process and receives the item;
2. Calculates your selected function on a received item;
3. If the result matches your filter criteria, sends it to the result process;
4. Repeats until all data are processed.

Data process works as follows:

1. Has an array only visible to itself. The array size cannot be larger than half of your data file;
2. Receives messages that either asks to insert an element or remove it from the array and does whatever action needed.
3. If the data array is full, it **cannot accept messages** from the inserting process;
4. If the data array is empty, it **cannot accept messages** from the removing processes;

Result process works as follows:

1. Has an array only visible to itself and inserts the received elements to it;
2. It must be able to insert an element received from another process and send all items to the main process.

All processes must satisfy these requirements:

- When all data are processed, processes exit by themselves.
- Data are sent between processes using channels (in Go) or messages (in MPI).
- The data array in the data process should be allowed to get full and empty (if it does not happen in your program, it should be possible to achieve it by adding `sleep` calls).

Tools for the lab (choose one):

- Go (channels must be used for all synchronization);
- C++ and MPI (MPI processes and communicator must be used).

L2 grading

- L2a — 6 points
- Test — 4 points

Deadline weeks for lab programs: a) 10, test) 12.

Programs have to be presented during lab lectures not later than the deadline, program (.cpp, .go), data and result files (.txt) have to be uploaded to Moodle before presentation.

Installing MPI

OpenMPI only works in Linux.

Note: Only university server can be used during the test.

For Debian-based Linux installation (Ubuntu and others):

1. Install g++ compiler: `apt install g++`
2. Install OpenMPI: `apt install libopenmpi-dev openmpi-bin`

For Windows 10:

1. Enable „Windows Subsystem for Linux“: Run the command in PowerShell:

   ```
   Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux
   ```

2. Restart your computer;
3. Find and download “Ubuntu 20.04 LTS” from “Microsoft Store”;
4. Run “Ubuntu” app;
5. Create a username and a password in the command prompt;
6. Run all steps from the Linux instruction above;
7. You Windows files should be accessible by Linux terminal from `/mnt` directory.

Connecting to KTU MPI server

1. Create an account for MPI server:
   (a) Open stud.if.ktu.lt.
   (b) Click “Registracija / Paskyros valdymas [mpilab.if.ktu.lt]” (“Registration / account management”)
   (c) Login using KTU credentials
   (d) Click “Kurti paskyrą” (“Create account”)
   (e) Your username and password will be displayed on the screen, you can use them to connect to the server
2. You can login to the server using any SSH client. For example,
   - If you use PowerShell, you can connect using command `ssh username@mpilab.if.ktu.lt`, where `username` is your username.

- If you use PuTTY, provide server address mpilab.if.ktu.lt, port 22, and enter your username and password when logging in.
- If you use a Debian-based system, open terminal and execute `ssh username@mpilab.if.ktu.lt`, where `username` is your username.

3. Files can be uploaded to the server using any SFTP client. Class computers have FileZilla installed. Server address and credentials are the same.