Exercise Sheet 1 for Lecture Parallel Programming

Deadline: 2024-04-21, 23:59

Prof. Dr. Michael Kuhn (michael.kuhn@ovgu.de)
Michael Blesel (michael.blesel@ovgu.de)
Parallel Computing and I/O • Institute for Intelligent Cooperating Systems
Faculty of Computer Science • Otto von Guericke University Magdeburg
https://parcio.ovgu.de

This exercise sheet is an introduction for the Linux CLI (Command Line Interface) and the C programming language. In the following, you are instructed to connect to our cluster, navigate the shell and work on basic programming tasks in C.

1. Forming Exercise Groups (Important!)

For submitting and working on the exercise tasks as a team, every group needs to have a Git repository in the university GitLab. These have to be created manually and assigned to your group. Therefore your first task is to form a group of three students and to send a list of your group members' OVGU account names via e-mail to michael.blesel@ovgu.de.

Important: Everyone needs to have logged into the GitLab (https://code.ovgu.de/) at least once to activate their account before we can assign you a repository.

Please do this as early as possible so that you will receive your repository ahead of the deadline. We will use the first exercise date to form the student groups. Should you for some reason not be able to attend the first exercise you can use the Mattermost to find a group.

2. Cluster Login and Setup (60 Points)

To work on many exercises, you will need access to our cluster. To log in there, connect to the OVGU VPN and then log in to the cluster via SSH. You can then work within the shell. A shell tutorial can be found at https://swcarpentry.github.io/shell-novice/.

On Linux, macOS or the Windows Subsystem for Linux, you can connect to the login node with the following command, where <name> stands for your OVGU account name. Alternatively, on Windows, you can use a graphical SSH client such as PuTTY. You can log in with your usual OVGU account password.

```
$ ssh <name>@ants.cs.ovgu.de
```

To avoid having to specify your user name each time, you can create the following entry in the SSH configuration under ~/.ssh/config.

```
Host ants.cs.ovgu.de
    User <name>
```

Moreover, in order not to have to enter your password every time, you can generate a key pair, which will make logging in easier.

```
$ ssh-keygen
$ ssh-copy-id ants.cs.ovgu.de
```

Data can be copied to and from the cluster using SCP.

```
$ scp local/file ants.cs.ovgu.de:remote/file
$ scp ants.cs.ovgu.de:remote/file local/file
$ scp -r local/directory ants.cs.ovgu.de:remote/directory
$ scp -r ants.cs.ovgu.de:remote/directory local/directory
```

After login, you must run the following command to make the exercise software environment available. Note the space between `.` and `/`. Remember to do this every time you connect to the cluster.

```
$ . /opt/spack/pp-2024/env.sh
```

The software environment contains a current GCC compiler. The cluster is used as a reference system for evaluating the tasks. The exercises are corrected there and must therefore also be executable there.

The software is made available in the form of so-called modules. You can use `module list` to display the currently loaded modules. `module avail` prints an overview of all available modules.

Questions: What software packages are loaded on the cluster by default and which versions are available? What commands would be required to load and then unload a fictitious module `foobar`? Is there a way to unload all loaded modules at once with a single command? How can you display more information about a particular module?

For editing the source code you should also install a suitable editor for the command line (e. g. nano, Vim, Emacs) or your graphical desktop environment (e. g. Atom, gedit). Integrated development environments (e. g. Visual Studio Code, Eclipse) make working even more convenient.[1]

3. Using the Command Line Interface (30 Points)

The following specific tasks are to be done:

1. Navigating the CLI (Command Line Interface)
    a) Get familiar with the usage of manual pages:
    `$ man man`
    b) Display the name of the current working directory:
    `$ man pwd`

[1] Visual Studio Code allows working directly via SSH: `https://code.visualstudio.com/docs/remote/ssh`

c) Display the contents of your home directory:
`$ man ls`

d) Create a new directory with the name `testdir`:
`$ man mkdir`

e) Change your working directory to the new directory:
`$ man cd`

f) Display the current working directory again.

g) Create an empty file with the name `testfile`:
`$ man touch`

h) Rename the new file to `testfile2`:
`$ man mv`

i) Copy the renamed file as `testfile3`:
`$ man cp`

j) Delete the file `testfile2`:
`$ man rm`

Question: With `which` you can display the path of an application. Why does this not work for the `cd` command? (Hint: `man bash`)

2. Creating an archive

a) Create a directory with the name `archive`.

b) In the new directory create a new file with random content:
`$ dd if=/dev/urandom of=archive/random bs=1k count=256`

c) Display the size of the file:
`$ ls -lh archive/random`

d) Display the size of the directory:
`$ ls -ldh archive`

e) Create a `tar` archive that contains the directory:
`$ tar -cf archive.tar archive`

f) Display the size of the archive `archive.tar`

Question: What strikes your attention about the three sizes?

g) Compress the archive:
`$ gzip archive.tar`
The archive is now created. `gzip` has automatically renamed the archive to `archive.tar.gz`.

h) Display the size of the compressed archive `archive.tar.gz`.

Question: Is it possible to create a compressed archive (`archive.tar.gz`) with only the `tar` command? How should this command have looked like?

i) Display the contents of the compressed archive.

You can find a tutorial at `https://swcarpentry.github.io/shell-novice/`.

4. C Basics (60 Points)

This task serves as an introduction to the C language. Basic constructs will be used and practiced. In the materials you will find a file `map.c` with an accompanying `Makefile`.[2] You can compile the application with `make map`.

A small geographical map of the size 3×3 shall be simulated. Declare a global static 3×3 array with the name `map` for this. Additionally define an enumeration data type (`enum`) `cardd` (cardinal direction) with the four cardinal directions N, E, S, W.

Implement the predefined function `set_dir` in a way that it sets the given cardinal direction on the map at the given coordinates `x` and `y`. Invalid inputs have to be handled appropriately.

Implement the display function `show_map` via the function `printf`. Use a `switch` statement to do this. The produced output should be of the following format:

```
0␣␣␣N␣␣␣0
W␣␣␣0␣␣␣E
0␣␣␣S␣␣␣0
```

4.1. Bitwise Operations (30 Bonus Points)

Extend the map with the cardinal directions north-west (NW), north-east (NE), south-east (SE) and south-west (SW). The `enum cardd` can be modified for this but not extended. The new cardinal directions shall be integrated with the use of bitwise operations and displayed in the following format:

```
NW␣␣N␣␣NE
W␣␣␣0␣␣␣E
SW␣␣S␣␣SE
```

A good introduction to C is for example The C Book (`https://publications.gbdirect.co.uk/c_book/`). A list of additional books and tutorials for C can be found at `http://www.iso-9899.info/wiki/Books`.

5. C Pointers (60 Points)

In this task, you will make yourself familiar with the basic concepts of pointers in C. The `pointer.c` file contains multiple functions. In some places the output via `printf` hints at the expected result. In other places the variable names or comments reveal what has to be done.

Your task is to complete the missing entries in a way that the described output is produced. Please consider: You are not allowed to change anything in the program except the parts that are marked with `TODO`. The final program has to compile without errors and warnings and has to produce semantically correct outputs.

[2] A Makefile tutorial is available at `https://swcarpentry.github.io/make-novice/`.

Submission

We will count your last commit on the `main` branch of your repository before the exercise deadline as your submission. In the root directory of the repository, we expect a `PP-2024-Exercise-01-Materials` directory with the following contents:

- A file `group.txt` with your group members (one per line) in the following format:

  ```
  Erika Musterfrau <erika.musterfrau@example.com>

  Max Mustermann <max.mustermann@example.com>
  ```

- A file `answers.txt` with your answers (Tasks 2 and 3)
- The modified code of the `map` application (Task 4)
  - All source code of your application (`map.c`); well documented
  - A `Makefile` such that `make map` and `make clean` create and delete binaries as expected
- The modified code of the `pointer` application (Task 5)
  - All source code of your application (`pointer.c`); well documented
  - A `Makefile` such that `make pointer` and `make clean` create and delete binaries as expected