Exercise Sheet 2 for Lecture Parallel Programming

Deadline: 2026-04-26, 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

---

1. Debugging C Applications (120 Points)

To be able to concentrate on efficient programming, we first want to look into the debugging of (parallel) applications. Therefore, we will work with the GNU debugger *GDB* and the memory checker *memcheck* from the *Valgrind* tool suite in this task. Both of these tools are usable for serial and parallel applications. Memory allocation is very error-prone in C and Valgrind can help detect common errors in memory management.

First Steps

The `simple` directory contains a simple application that can be compiled with `make`.[1] This application will be used for you to familiarize yourself with GDB and Valgrind. It contains four functions, which return a pointer to an array or to a value inside an array and a `main` function that outputs the returned values. Unfortunately, this application contains multiple errors.

- Perform the following small tasks to get familiar with GDB. Document the used `gdb` commands and the output in a text file. A short GDB tutorial can for example be found at `https://www.cs.cmu.edu/~gilpin/tutorial/`.
  - Place a breakpoint on the `mistake1` function, start the application and print out the values of `buf` and `buf[2]`. Continue to the next line and repeat the printing of both values. What is the type of `buf`?
  - Place a breakpoint on the `mistake2` function and continue the application execution. What is the type of `buf`?
  - Continue the application execution. What output do you get now? Display the code around the current line. Which frames are on the stack? Switch to frame 1. Print out the contents of `p`.
  - Call the `mistake3` function in GDB (look up how to call functions directly).
- Firstly, modify the application so that it does not crash anymore. Try to keep the amount of code modifications to a minimum. Use `gdb` to find the errors in the code. The correct application output should look like this:

[1] For an introduction into Makefiles, see `http://swcarpentry.github.io/make-novice/`.

```
1: 1
2: 2
3: 3
4: 4
```

- Now the application is running, but it still contains memory errors that can occur (more or less at random) depending on the environment. Use Valgrind's `memcheck` tool to find the memory management errors and modify the application so that every function allocates memory correctly and that all memory is freed correctly before the termination of the application. Execute the program using `valgrind ./simple`.

  Note: Simply allocating the memory with `static` is *not* allowed. Also, global arrays and variables are not to be used. A short introduction to `valgrind` can for example be found at `https://www.valgrind.org/docs/manual/quick-start.html`.

Document the errors that cause application crashes and memory errors. For every error, state the corresponding lines of code that are erroneous and explain the reasons for the error (for example, multiple freeing of same memory).

2. Debugging of a Complex Application (120 Points)

In the `pde` directory you will find a program for solving partial differential equations. The basic structure of the program is correct but it contains some small errors. To fix the program you do not have to undestand what exactly is being caluclated, but it can help to step through the program with a debugger to get a better understanding of what is happening. Fix all errors in the program and modify only as little code as necessary. Remember to also check for memory errors by using Valgrind.

The program shall be run with the following command `./partdiff 1 1 100 2 5`

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-2026-Exercise-02-Materials` directory with the following contents:

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

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

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

- A text file with the input and output for GDB called `gdb-output.md`, a text file with error descriptions (reasons, code locations) called `simple-error.md` and the modified source code in a directory called `simple` (Task 1)
- A text file `pde-error.md` with error descriptions (reasons, code locations) and the modified source code in a directory called `pde` (Task 2)