Lab Assignment 1

Setup and Introduction to Application Example

Assigned: **09/13/2019**
Due: **09/20/2019 noon**

---

1 Accessing the Lab Machines

The course tools are installed on the Linux machines operated by the Engineering Computing Center of the College of Engineering. The next two subsections describe how to access the lab machines.

1.1 Remote Desktop through VLAB

You can get a Linux remote desktop session using the COE VLAB [1]. Use the **Linux_lab** cluster. This is helpful when using graphical output.

1.2 Remote Access through SSH

Alternatively, you can use these machines remotely through ssh (replace <UserName> with your user name):

```
ssh -X <UserName>@gateway.coe.neu.edu
```

In case you do not have an ssh client on your Windows machine (MAC and Linux have them anyway), try putty `http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html` For future uses, you will also need an X server running on your machine in order to display graphical output on your local Windows machine (MAC and Linux are covered automatically). If you do not have an X server installed, try VcXsrv: `https://sourceforge.net/projects/vcxsrv/` You can also use MobaXterm `https://mobaxterm.mobatek.net/` which combines SSH terminal, file transfer and an XServer.

Please use the Discussion Board on Canvas to discuss any problems. Again, this allows for quick information dissemination.

2 Setup

Use your SSH terminal to connect to one of the servers and make yourself familiar with the available commands in the Linux environment. You may also configure your shell setup to have a convenient working and C/C++ programming environment.

For this course, you will need to be comfortable with editing text and source code files in a text editor (e.g. vi, emacs, or gedit). You will also need to be familiar with the GNU C/C++ compiler chain for building executable programs. If you are not familiar with these tools, study available online tutorials and other resources and practice C/C++ programming in Linux.

Since our modeling example is an image processing application, you will also need tools for handling digital images. There are many command-line tools available in Linux that allow you to convert and manipulate images. Most start with pnm, pbm, or pgm, depending on the file format they process. To enumerate

them, you can type their prefix into your shell followed by a TAB. Documentation is available via the corresponding man pages. In addition, graphical tools are available as well (if you have an X client running), which may be more convenient to use. To view images, you can use eog which supports most image types (including our pgm format). Finally, to manipulate images (e.g. if you want to use your own photos as a test case for our application), you can use gimp, a very powerful image editor in Linux.

3 Application Example

For the embedded system modeling project in this course, we will use a specific image processing application, namely an implementation of the Canny Edge Detector [2] algorithm. The overall project goal is to design a suitable embedded system model of this application and describe it in a System-Level Description Language (SLDL). This embedded specification model will then not only be simulated for functional and timing validation, but also be refined for synthesis and implementation as an embedded System-on-Chip (SoC) suitable for use in a digital camera. The Canny Edge Detector algorithm takes an input image, e.g. a digital photo, and calculates an output image that shows only the edges of the objects in the photo, as illustrated in the figure below. We will assume that this image processing is to be performed in real-time in a digital camera by a SoC that we design and develop. You can read about Canny application on Wikipedia or find other resources (please cite your sources in the report).

4 Instructions

The purpose of this first assignment is for you to become familiar with the Canny algorithm and its application source code.

4.1 Download the application source code

You can download the Canny application in C source code from the web site at:

- `ftp://figment.csee.usf.edu/pub/Edge_Comparison/source_code/canny.src`

We will use this C reference implementation of the Canny edge detection algorithm as the starting point for our embedded design model.

Upload the reference image beachbus.pgm (from assignment) to the same directory.

4.2 Test the given C code

Convert the downloaded canny.src file into a single ANSI-C file canny.c. Few adjustments will be necessary in order to cleanly compile the application on our Linux infrastructure with a modern compiler chain. Then you can compile and test the application, similar to the following:

```
vi canny.c
gcc canny.c -lm -o canny
./canny beachbus.pgm 0.6 0.3 0.8
eog beachbus.pgm_s_0.60_l_0.30_h_0.80.pgm
```

Create a Makefile [3] for compilation. Validate that your code can be cleaned, can compile and run the test.

```
make clean
make
make test
```

4.3 Study and profile application

Before embarking on system-level design, we need to study and understand the application well. For this, examine the source code and its execution. You should be able to answer questions like the following:

- What are the main functions of the algorithm?
- Which functions are used for data input and for data output?
- Where does the actual computation occur?
- How much memory is needed when the algorithm runs?
- Are there any obvious candidates for hardware acceleration?
- What should better be performed in software?

To help answer some of these questions, please profile the application using gprof [4] (tools are installed on COE Linux machines). Include the relevant gprof output and your analysis in the report. Please also include in your report a reference to the tutorial you have used for learning about gprof (you will need to find a suitable tutorial). Warning: using gprof on the example image (beachbus.pgm) will not result anything meaningful. For this convert a higher resolution image to pgm to use as an input. Reflect in the report why beachbus.pgm did not yield meaningful results.

4.4 Write-up Instructions and Lab Assignment

1. Create a brief lab report and include an output image of the canny edge detection.
2. Browse the sources and analyze the source code structure. Draw a block diagram of the function hierarchy and their communication dependencies. Focus on the essential functions that make up the algorithm. Ignore file access and math library functions.
3. Make a coarse grained estimation of how much memory the application takes for execution – how many copies of the image are kept? Please briefly reason about the suitability of hardware implementation.
4. Report on your findings from profiling the application with gprof (see step 3 for details)

Please submit your solutions via Canvas. Submissions should include a single PDF with the writeup and tar.gz archive for the source code. Your code should compile without warnings on the COE Linux systems. Validate the functional correctness of your code by comparing the compressed image against the reference code. Provide a Makefile to compile your modified code.

Please follow the file naming convention:

```
<myneu_id>_<hw|lab>_<Number>.<file_suffix>
```

Example: `h.husky_lab_1.tar.gz`, `h.husky_lab_1.pdf`

References

[1] College of Engineering Computer Center, **VLAB**, http://help.coe.neu.edu/coehelp/index.php/VLAB

[2] Canny, J., **A Computational Approach To Edge Detection**, IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986.

[3] GNU, **Makefile**, https://www.gnu.org/software/make/manual/html_node/Introduction.html

[4] GNU, **GPROF**, http://man7.org/linux/man-pages/man1/gprof.1.html