Data Processing for Mathematicians

Michael A. Saum

University of Tennessee
Department of Mathematics

Overview

- `gcc` Compiler
- Libraries
- Useful Libraries
- Graphics Programs
- `gdb` Debugger
- Profiling
- Choosing the *right* Application
- References

Recall . . .

- The previous seminar covered the very basics of programming in C in UNIX.
- One usually writes a program to help solve a particular problem.
- Algorithm development and coding is usually only part of the solution.
- Other parts include debugging, benchmarking, and analysis of results.
- One goal of this seminar is to provide information on the various tools and applications which can aid in obtaining *complete* solutions.

Compile-Link

- When one compiles a program, a *binary executeable* version of the source code is produced, which can subsequently be run on the machine you are on.
- What is usually hidden from the programmer is the fact that this is actually a two step process.
- The first step (*compile*) is to translate the source code into a format which contains an "intermediate" *machine code* translation called an *object file*.
- The second step (*link*) is to transform the *object code* into a *machine language* or *executeable* file which can then be run on the machine you are on.

Compile-Link, contd.

- Note that object files if saved will usually have the extension `*.o`, and are essentially machine code "almost" ready to run.
- Do not edit object or executeable files, they are binary files!
- The standard UNIX convention is that the executeable is named `a.out` if no name is specified to the compiler for the executeable.
- When a program is run on UNIX, the executeable is *loaded* into memory and machine instructions are executed.

`gcc` compiler

- The `gcc` compiler is the default C, C++, and FORTRAN compiler on most LINUX systems.
- It is considered to be *Open Source* in that there is no fee for using the compiler.
- Other compilers exist but usually cost $. For example, on `agnesi, fubini, fatou, turing` the Intel Compilers exist with limited licences.
- `icc` is the Intel C/C++ compiler.
- `ifc` is the Intel FORTRAN compiler (currently available only on `agnesi`.)

`gcc`, contd.

- `gcc` is used to compile C programs, `g++` is used to compile C++ programs, and `g77` is used to compile FORTRAN programs.
- All three share some basic *command line options* which control various aspects of the compilation process.
- Unless otherwise specified, I will use the term `gcc` to refer to all three compilers `gcc`, `g++`, `g77`.

`gcc` Options

- For the most part, `gcc` compile options can go in any order in the command line with the exception of the file(s) to process, which should go last.
- To just compile `sample.c` and produce an object file `sample.o` with no executeable

```
gcc -c sample.c
```

- To just link object code `sample.o` and produce an executeable file named `sample`

```
gcc -o sample sample.o
```

- To compile and link `sample.c` and produce an executeable file named `sample`

```
gcc -o sample sample.c
```

gcc Options, contd.

The following optimization options can improve program performance:

- **-O0** Do not optimize.
- **-O** or **-O1** Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function. Without -O, the compiler's goal is to reduce the cost of compilation and to make debugging produce the expected results. With -O, the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time.
- **-O2** Optimize even more. GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff. The compiler does not perform loop unrolling or function inlining when you specify -O2. As compared to -O, this option increases both compilation time and the performance of the generated code.
- **-O3** Optimize yet more. -O3 turns on all optimizations specifi ed by -O2 and also turns on the -fi nline-functions and -frename-registers options.
- **-Os** Optimize for size.

gcc Options, contd.

- **-pg** Generate extra code to write profile information suitable for the analysis program "gprof". You must use this option when compiling the source files you want data about, and you must also use it when linking.
- **-g** Produce debugging information in the operating system's native format. GDB can work with this debugging information.
- **-Wall** This enables all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros.
- **-Werror** Make all warnings into errors.
- **-I*dir*** Add the directory *dir* to the head of the list of directories to be searched for header files.
- **-L*dir*** Add directory *dir* to the list of directories to be searched for -l.

`gcc` Options, contd.

-l*library* Search the library named *library* when linking.

- The only difference between using an -l option and specifying a file name is that -l surrounds library with lib and .a and searches several directories.
- The linker searches a standard list of directories for the library, which is actually a file named lib*library*.a. The linker then uses this file as if it had been specified precisely by name.
- It makes a difference where in the command you write this option; the linker searches and processes libraries and object files in the order they are specified. Thus, `foo.o -lz bar.o` searches library z after file foo.o but before bar.o. If bar.o refers to functions in z, those functions may not be loaded.
- The directories searched include several standard system directories plus any that you specify with -L.
- Normally the files found this way are library files—archive files whose members are object files. The linker handles an archive file by scanning through it for members which define symbols that have so far been referenced but not defined. But if the file that is found is an ordinary object file, it is linked in the usual fashion.

Libraries

- The previous slide alluded to the concept of *library files*, which needs clarification.
- A library is a file containing several object files, that can be used as a single entity in a linking phase of a program. Normally the library is indexed, so it is easy to find symbols (functions, variables and so on) in them. For this reason, linking a program whose object files are ordered in libraries is faster than linking a program whose object files are separate on the disk. Also, when using a library, one have fewer files to look for and open, which even further speeds up linking.
- Unix systems (as well as most other modern systems) allow us to create and use two kinds of libraries - static libraries and shared (or dynamic) libraries.
- Static libraries usually have an extension `*.a`
- Shared libraries usually have an extension `*.so`
- Static libraries are just collections of object files that are linked into the program during the linking phase of compilation, and are not relevant during runtime.

Libraries, contd.

- Shared libraries (also called dynamic libraries) are linked into the program in two stages. First, during compile time, the linker verifies that all the symbols (again, functions, variables and the like) required by the program, are either linked into the program, or in one of its shared libraries. However, the object files from the dynamic library are not inserted into the executable file. Instead, when the program is started, a program in the system (called a dynamic loader) checks out which shared libraries were linked with the program, loads them to memory, and attaches them to the copy of the program in memory.
- The basic tool used to create static libraries is a program called `ar`, for 'archiver'.
- This program can be used to create static libraries (which are actually archive files), modify object files in the static library, list the names of object files in the library, and so on.

Libraries, contd.

- In order to create a static library, one can use a command like this:

```
ar rc libutil.a util_file.o util_net.o util_math.o
```

- This command creates a static library named ’libutil.a’ and puts copies of the object files "util_file.o", "util_net.o" and "util_math.o" in it. If the library file already exists, it has the object files added to it, or replaced, if they are newer than those inside the library.
- After an archive is created, or modified, there is a need to index it. This index is later used by the compiler to speed up symbol-lookup inside the library, and to make sure that the order of the symbols in the library won’t matter during compilation.
- The command used to create or update the index is called `ranlib`, and is invoked as follows:

```
ranlib libutil.a
```

- To list the contents (index) of a library, use the command

```
nm -s libutil.a
```

Useful Libraries

- In most cases, one will not have to worry about creating libraries as described above.
- Two excellent source code repositories are GAMS and NETLIB.
- GAMS - Guide to Available Mathematical Software

  `http://gams.nist.gov/`
- NETLIB

  `http://www.netlib.org/`
- There are many other sources of source code on the internet, most come packaged in such a way that when compiled (with a *Makefile*) a library is created automatically.

Useful Libraries, contd.

Available within the Math department on `agnesi, fubini, fatou, turing` are the following collections of functions for which libraries exist:

- BLAS - Basic Linear Algebra System. This set of routines perform Vector, Matrix-Vector, and Matrix-Matrix operations.
- LAPACK - Linear Algebra Package. This set of routines perform Linear Algebra operations such as solving linear systems, matrix factorization, and eigenvalue calculations.
- ATLAS - Automatically Tuned Linear Algebra System. This is a set of optimized (for the machine you are on) BLAS and a subset of LAPACK routines.
- GSL - Gnu Scientifi c Library. This is a set of routines which provides a great number of functions that are of interest to scientifi c calculation.
- GLIB - A part of the GTK+ package, this collection of routines provides many utility routines used in application development on UNIX. Using routines provided in glib one can introduce into one's application data structures such as linked lists (single and double), N-ary trees, and hash tables relatively easily. In addition, if one desires to write an application that has a GUI, routines exist for standard X-window application development.

GSL Library

Complex Numbers	Roots of Polynomials	Special Functions
Vectors and Matrices	Permutations	Combinations
Sorting	BLAS Support	Linear Algebra
Eigensystems	Fast Fourier Transforms	Quadrature
Random Numbers	Quasi-Random Sequences	Random Distributions
Statistics	Histograms	N-Tuples
Monte Carlo Integration	Simulated Annealing	Differential Equations
Interpolation	Numerical Differentiation	Chebyshev Approximations
Series Acceleration	Discrete Hankel Transforms	Root-Finding
Minimization	Least-Squares Fitting	Physical Constants

Example

It is probably best at this time to illustrate how one can implement some of the above concepts with an example.

- The source code and Makefile are attached as supplementary material.
- The program utilizes the `gsl` library for access to some special functions.
- The program then generates data in an output file for later processing and graphical display.

Example, contd.

- `example.c`
  - Line 13 requests to include a file necessary for accesing gsl bessel function routines.
  - Lines 20 and 56-60 define an auxiliary function; put in to obtain timing information from profiling.
  - `example` when run will produce output to stdout (Lines 40, 47, and 50), and will dump *raw* data to the file `example.dat`.
- `Makefile`
  - Line 6 describes additional paths to look for `include` files.
  - Line 7 describes additional paths AND libraries to look for code for functions which are called from your source code.
  - Line 8 describes compiler and linker option flags to be used. In this example, the executeable `example` will be able to be debugged (using `gdb`) as well as generating profile information. In addition, all compiler warnings will be printed and the compilation will stop on all warnings, with basic optimization (level 1) being done.

Graphing

There are a number of options to generating graphs, both 2D and 3D as well as putting together sequences of pictures to produce a movie. The most fundamental rule is to create graphics with minimal amount of work which also meet your analysis needs.

- `matlab` - Integrated GUI platform combining calculations and display of calculations. Can handle just about anything displayed graphically, takes time to get it right. Nice for making movies.
- `xmaple` - Integrated GUI platform combining symbolic math and display of results. As good if not better than `matlab` at displaying 3D and movies.
- `gnuplot` - non GUI, 2D/3D, batch command fi les. One of the quickest ways to import data and graph it.
- `xmgrace` - GUI, 2D, batch command fi les, quick and clean linear and nonlinear regression analysis. Has a non GUI interface and language available (`grace`).
- `R` - non GUI, statistics based graphs are its specialty. Much more than just graphics though.
- `gimp` - GUI, when you have to edit the pixels of an image.

Graphing, contd.

- All of the above applications can read in and process (or parse) the data into different *data sets* which can then be graphed or displayed in a wide variety of ways.
- My philosophy is basically to generate fi les containing multiple columns of data, and then bring them in one of the above graphics programs and tune the graphs to meet my needs.
- `matlab, maple, R` all provide full featured programming environments which means that a separate C program may not have to be written.
- `matlab, maple` may not be the best graphics choice for very large data sets.

gnuplot

- `gnuplot` can be started by just typing `gnuplot` on the command line. Once started, type `help` to obtain more information about the commands.
- This mode is great for exploring gnuplot capabilities and commands that are available.
- Once one has figured out the details of what commands gnuplot needs for your plot, put these commands into a separate file (see `example.gnu` listing in the supplementary materials.)
- To generate plots then,

```
gnuplot example.gnu
```

- Multiple plots can be generated at one time.
- There are a lot of tutorials on the web. See also

`http://www.gnuplot.info/`

xmgrace

- `grace` provides a non GUI interface to `xmgrace`
- `xmgrace` starts up a GUI interface to full featured 2D plotter and data analyzer.
- It is strongly recommended that one view the grace documentation located at:

  `http://plasma-gate.weizmann.ac.il/Grace/`

- Compare the plots printed out using `xmgrace` and `gnuplot` for our Bessel Function example.

ImageMagick

- ImageMagick is a collection of routines which allow one to *convert* one graphics image format to another.
- It is best to embed postscript (.ps) or encapsulated postscript (.eps) files into LaTeXdocuments. Using `convert` one can easily perform this task, no matter what the original graphics format was in.
- `animate` can assemble together a number of still images and make it look like a movie.
- See the `man` pages for more information regarding this set of programs.

gimp

- This is the program if you ever have to edit the detail on an image.
- Many options, similar to Adobe photoshop in that it can allow one to perform a lot of transformations on images, such as filtering, sharpening, etc.
- I use only as a last resort when nothing else works for converting part of an image into a more manageable image size and format.
- For more information, refer to

`http://www.gimp.org/`

R

- R is a full featured system which is basically a cross between SAS and Matlab.
- The best place to find out more information is at:

  `http://www.r-project.org/`

- There is a ton of documentation and some great tutorials available on the internet through the above site.
- This application can be extended in a wide variety of ways to include one's own C libraries, access to large databases, etc.

gdb

- `gdb` is what is called a *debugger*. There are other debuggers such as `dbx`, which may or may not be installed on a system.
- It basically allows one to single step through a program, line by line, and display variables contents so as to track down bugs in programs.
- In order to effectively use `gdb`, one must have compiled the program being debugged with the `-g` compiler option flag.
- For more information, refer to

  `http://www.gnu.org/software/gdb/gdb.html`
- Only use a debugger when manual debugging (i.e., write statements) fails.

gprof

- On applications which require a lot of CPU time to run, it might be worthwhile to analyze how much time an application spends in each routine.
- `gprof` can help in identifying the "CPU Hogs" of your program.
- I order to use `gprof`, one must have compiled and linked the program being profiled with the `-pg` compiler flag.
- More information is available at:

`http://www.cs.utah.edu/dept/old/texinfo/as/gprof_toc.html`

Choosing the RIGHT application

- The most important part of processing data is to utilize the tools available to produce the results one wants with the least amount of hassle.
- For the quick and clean programming integrating with adequate graphics capability, `matlab` or `maple` or `R` are good choices.
- For statistics calculations and display, `R` is very good, although it's interface and language are a bit strange at first.
- There is nothing wrong with writing a C or FORTRAN program to generate data files which are subsequently processed later by another program such as `gnuplot` or `xmgrace`.
- If one has a specialized mathematical computing need, let `google` search for you. You never know, someone may have already developed the code to solve your problem.

References

- MemCheck - Check for Memory Problems

  `http://hald.dnsalias.net/projects/memcheck/`

- ElectricFence - Check for Memory Problems

  `http://linux.maruhn.com/sec/electricfence.html`

- Valgrind - Detailed Memory Analysis tools

  `http://valgrind.kde.org/`

- New applications are being developed all of the time for Linux. Check a very good repository at:

  `http://freshmeat.net/`

- For extracting particular pieces of data from large data files, `perl` is the language of choice, see

  `www.perl.com`