Linux Development Basics

Operating Systems I

Victor Yacovlev (Viktor Iakovlev)

The Programming Languages

- Interpretable: BASH, Python, Perl, Ruby
  - shipped with Linux by default
  - might be started via **#!** line
- Compilable to Native Code: C, C++, Fortran etc.
  - two major compilers: **gcc** and **clang**
  - core libraries are part of Operating System
- Compilable to Intermediate Bytecode: Java, C#

The Compile Stages

```
program.bin: main.o module1.o module2.o
   g++ -o program.bin main.o module1.o module2.o
```

Link all Codes into Executable File

```
main.o: main.cpp
   g++ -c -o main.o main.cpp
```

Translate Text to Native Code

```
module1.o: module1.cpp
   g++ -c -o module1.o module1.cpp
```

Translate Text to Native Code

```
module2.o: module2.c
   gcc -c -o module2.o module2.c
```

Translate Text to Native Code

Library of Codes

- Static Library
  - just an Indexed Archive of object files
  - rarely used today
- Dynamic Library
  - like a program executable file
  - has no entry point
  - has a Symbol Table
    you can see it via **objdump -t** command

Library of Codes

- Linux and many other UNIXes: use **ldd** command
- MacOS X: the **otool** command makes the same
- Windows: there is freeware **Dependency Walker** tool [http://www.dependencywalker.com/]

Library of Codes

- To create use **-shared** and **-fPIC** options passed to compiler
- To use library – link it with **-lNAME** flag
- Consider using **LD_LIBRARY_PATH** environment variable
- When using **-fPIC** option, it is possible to load library at runtime.
  Example: see docs on Python module **ctypes**

Externs

- In C/C++ there is **extern** function visiblity by default for most compilers
- An opposite is **static** visiblity (do not be confused to static variable declaration or class method or field in C++!)
- The symbol table stores just **names** and **addresses**, but not function signatures
- Header files are required for proper signatures handling

Mix of Codes

- Use C++ code in C programs – extern “C”
- Use C code in C++ programs – nothing special
- You can use C, C++, Objective-C, Pascal, Fortran, Ada and Rust code in the same project
- But all the these languages uses their own features not compatible each other
- The Plain C language and its data structures is a common supported subset to make code connections

Example: The String

- **std::string** in C++
  - it is a class
  - stores a pointer to data and actual length
- **String** type in Pascal
  - 0-byte stores the string length
  - the rest 255 bytes are symbols
- **char*** in C
  - just a pointer to the first character
  - the string has special value ‘\0’ as end-of-string marker

Plain-Old-Data types

- Scalar values (ints and floats) are safe to use
- Strings should be represented in C-style
- Arrays should be represented in type* style
- Do not forget on **new/delete/new[]/delete[]/malloc/free** usage
- The library should provide it’s own allocation and deallocation functions

Memory Allocation/Free Problem

```
/* myclass.h */

class MyClass {
public:
  static char* getString();
};
```

Library compiled by MSVC

```
/* myclass.cpp */

char* MyClass::getString()
{
  result=(char*)malloc(10);
  memcpy(result, "Hello");
  return result;
}
```

Program compiled by MSVC

```
/* program.cpp */

void someFunc();
{
  char* r = MyClass::getString();
  printf("%s", r);
  free(r);
}
```

Memory Allocation/Free Problem

```
/* myclass.h */

class MyClass {
public:
  static char* getString();
};
```

Library compiled by MSVC

```
/* myclass.cpp */

char* MyClass::getString()
{
  result=(char*)malloc(10);
  memcpy(result, "Hello");
  return result;
}
```

Program compiled by MinGW

```
/* program.cpp */

void someFunc();
{
  char* r = MyClass::getString();
  printf("%s", r);
  free(r); // Segmentation Fault
}
```

C versus C++

The Input and Output

- Use **printf** and **puts** functions instead of **std::cout**
- Use **scanf** and **fgets** (notice on ‘f’ letter) instead of **std::cin**

Arrays

- Two kind of arrays in C99 and C11:
  - heap-allocated using **calloc** or **malloc**
  - stack-allocated – much faster, but be careful on size

The Tools

The Debugger

- **gdb** command
- has a text user interface **gdb --tui**
- has a graphical tool ddd
- requires a code to be compiled with **-g** and **-O0** options

IDE

- Code Blocks – too legacy
- CLion – the most powerfull but too fat
- QtCreator – has many features