Compiler Laboratory

Prof. P. P. Das and Prof. I. Sengupta

September 10, 2020

How to create library modules in C++ ?

C++ Program using System Call

```
#include <unistd.h>
#define LEN 19
int main()                              // second.cpp
{
        char str[LEN] = ″My second program\n";
        write(1, str, LEN);    // STDOUT_FILENO=1
        _exit(0) ;
}
```

Assembly Code Translation

```
        .file "second.cpp"
        .text
        .globl main
        .type main, @function
main:
.LFB0:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $32, %rsp                  # 32-byte stack-frame
        movq    %fs:40, %rax               # Segment addressing
        movq    %rax, -8(%rbp)             # M[rbp-8] <-- rax
        xorl    %eax, %eax                 # Clear eax
        movl    $1931508045, -32(%rbp)
                # 0111 0011 0010 0000 0111 1001 0100 1101
                # 73 20 79 4D - "s yM"
```

```
movl    $1852793701, -28(%rbp)
        # 0110 1110 0110 1111 0110 0011 0110 0101
        # 6E 6F 63 65 - "noce"
movl    $1919950948, -24(%rbp)
        # 0111 0010 0111 0000 0010 0000 0110 0100
        # 72 70 20 64 - "rp d"
movl $1634887535, -20(%rbp)
        # 0110 0001 0111 0010 0110 0111 0110 1111
        # 61 72 67 6F - "argo"
movw $2669, -16(%rbp)
        # 0000 1010 0110 1101
        # 0A 6D - "\nm"
movb $0, -14(%rbp)
        # 0000 0000
        # 00 - '\0'
        leaq     -32(%rbp), %rax              # rax <-- (rbp - 32) (str)
        movl     $19, %edx                    # edx <-- 19 (LEN)
        movq     %rax, %rsi                   # esi <-- rax (str)
        movl     $1, %edi                     # edi <-- 1 (stdout)
        call     write                        # call write
        movl     $0, %edi                     # edi <-- 0
        call     _exit                        # call exit
.LFE0:
        .size main, .-main
        .ident "GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5)4.6.3"
        .section .note.GNU-stack,"",@progbits
```

Using x86-64 Software Interrupt

```
#include <asm/unistd.h>
#include <syscall.h>
#define STDOUT_FILENO 1

.file "second.S"
.section .rodata
L1:
    .string "My Second program\n"
L2:
.text
.globl _start
```

```
_start:
        movl $(SYS_write), %eax           # eax <-- parameters to write
        movq $(STDOUT_FILENO), %rdi       # rdi <-- 1 (stdout)
        movq $L1, %rsi                    # rsi <-- starting address of string
        movq $(L2-L1), %rdx               # rdx <-- L2 - L1 string length
        syscall                           # software interrupt
                                          # user process requesting OS service
        movl $(SYS_exit),%eax             # eax <-- 60(exit) parameters to exit
        movq $0, %rdi                     # rdi <-- 0
        syscall                           # software interrupt
        ret                               # return
```

Creating a Simple C++ Library

Simple Library: Printing an Integer

```
#define BUFF 20                            // filename “printInt.cpp”
void print_int(int n) {
        char buff[BUFF], zero=’0’;
        int i=0, j, k, bytes;
        if(n == 0) buff[i++]=zero;
        else {
                if (n < 0) {
                            buff[i++]=’-’;
                            n = -n;
                          }
                while (n) {
                           int dig = n%10;
                           buff[i++] = (char) (zero + dig);
                           n /= 10;
                         }
                if (buff[0] == '-') j = 1;
                else j = 0;
                k = i - 1;
                while (j < k) {
                        char temp=buff[j];
                        buff[j++] = buff[k];
                        buff[k--] = temp;
                }
        }
        buff[i] = '\n';
        bytes = i + 1;

        __asm__ __volatile__ (
                "movl $1, %%eax \n\t"
                "movq $1, %%rdi \n\t"
                "syscall \n\t"
                :
                :"S"(buff), "d"(bytes)
        );                                  // $1: on stdin
}
```

Tells the compiler that it is not allowed to move this assembly block.

Printing an Integer

```
#ifndef _MYPRINTINT_H                          // printInt.h
  #define _MYPRINTINT_H
  void print_int(int);
#endif
```

```
#include <iostream>
using namespace std;
#include "printInt.h"

int main()                                     // mainPrintInt.cpp
{
        int n;
        cout << "Enter an integer: "; cin >> n;
        print_int(n);
        return 0;
}
```

#ifndef, #endif ::

Handled by the C preprocessor

Creating a Library

```
$ c++ -Wall -c printInt.cpp

$ ar -rcs libprintInt.a printInt.o

$ c++ -Wall -c mainPrintInt.cpp

$ c++ mainPrintInt.o -L. -lprintInt

$ ./a.out
Enter an integer: -123
-123

$
```

Make file

- An utility program that automatically decides which part of a large software requires to be recompiled.
- The structure of a Makefile is a sequence of the following form:

```
Target: Prerequisites
                Command
```

- Target: name of a file generated by a program, e.g. main.o
- Prerequisites: files required to create the target, e.g. main.cpp, xyz.h
- Command: used to create the target, e.g. cpp –Wall main.cpp

A Simple Makefile

```
a.out:            mainPrintInt.o libprintInt.a
                          c++ mainPrintInt.o -L. -lprintInt

mainPrintInt.o:   mainPrintInt.cpp printInt.h
                          c++ -Wall -c mainPrintInt.cpp

libprintInt.a:    printInt.o
                          ar -rcs libprintInt.a printInt.o

printInt.o:       printInt.cpp printInt.h
                          c++ -Wall -c printInt.cpp

clean:
                  rm a.out mainPrintInt.o libprintInt.a printInt.o
```

Using Makefile

```
$ make clean
   rm a.out mainPrintInt.o libprintInt.a printInt.o


$ make
   c++ -Wall -c mainPrintInt.cpp
   c++ -Wall -c printInt.cpp
   ar -rcs libprintInt.a printInt.o
   c++ mainPrintInt.o -L. -lprintInt
```

Point to Note

- We may copy the library to a standard directory as a super-user. In that case specifying the library path is not necessary.

```
$ cp libprintInt.a /usr/lib

$ c++ mainPrintInt.o -lprintInt
```

Shared Library

- Following are the steps for creating a shared library:

```
$ c++ -Wall -fPIC -c printInt.cpp
$ c++ -shared -Wl,-soname,libprintInt.so -o
           libprintInt.so printInt.o
```

- Perform the following steps as super-user:

```
$ cp libprintInt.so /usr/lib
$ ldconfig -n /usr/lib
```

- The soft link “libprintInt.so.1” is created under /usr/lib.
- Final compilation:

```
$ c++ mainPrintInt.o -lprintInt
```

The new “a.out” does not contain the code of “print_int()”. Rather, it contains the corresponding Procedure Linkage Table (plt).

Thank you