Notes on Systems Programming in C

September 18, 2007

1 Mechanics of Compilation

The simplest way to compile a C program `myprog.c` is to type:

```
> gcc myprog.c
```

Assuming successfull compilation, the output is an executable file called `a.out` (in unix) or `a.exe` (in cygwin). But in this invocation, you are calling more than just the compiler!

1. First, the compiler invokes a PREPROCESSOR to handle lines that begins with a "#". Macro calls are also handled by the preprocessor.
2. The compiler (gcc) translates your preprocessed program into assembly language *but in symbolic terms*. E.g., a store instruction will read "STORE VAR_A" where VAR_A is a variable name.
3. This output is processed by another program called the ASSEMBLER that converts "STORE VAR_A" into a binary word (with VAR_A resolved into a numeric address). If you stop the process the this point (with the `-c` option) (see below).
4. Next, gcc calls the LINKER (or link editor) which builds the executable program by linking the object file to the library routines.

Suppose you want to prevent the compiler from calling the linker and loader. Then you type

```
> gcc myprog.c -c
```

(so -c means "just compile"). This will produce the output file `myprog.o`, called an "object file".

If you want to create an executable from myprog.o, you can now call the loader directly:

`> ld -lc myprog.o`

This will produce the `a.out` or `a.exe` you expected. NOTE: that `ld` must be explicitly given the library `-lc`.

Q: Why would you want to do this round-about way to producing `a.exe`? A: Your `myprog.o` file may contain routines that can be linked to other programs that wants to use them. This saves time on re-compiling (an issue in large projects).

This brings us to the subject of libraries. Every program refer to library routines. For instance, `main()` is actually the name of a library routine. The standard C library routines are found in a library file named `libc.a`. This library file is usually found in the system include directory called `/usr/lib`. The file `libc.a` is basically a list of a bunch of object files (`*.o`)!

CONVENTION: All library files are named `libXXX.a` where `XXX` is any identifier. To ask the compiler to search the library `libXXX.a`, you give it the option `-lXXX`. Thus, if `XXX=c`, we use the option `-lc` as in our illustration.

Suppose your program uses a mathematical library routine like "sqrt". This is in another library called `libm.a`. In this case, you MUST tell gcc to search the `libm.a`. You should now invoke

`gcc myprog.c -lc -lm`

If you have several libraries to search, give each one as a separate `-lXXX` option. Below, we provide more notes on Runtime libraries and Shared libraries.

QUIZ: what is odd about this command line: `> gcc myprog.c -lm -c`.

2 Run-Time Organization of C Programs

The compiled code is loaded into memory in three segments (code, data and stack):

As illustrated in Figure 1, these segments are laid out in a specific order: (starting from lowest address `0000` to highest `FFFF`) (1) code segment, (2) the data segment, (3) a gap of free space, and finally (4) the stack segment.

Because the gap is flexible, we can arrange the top of the stack in the stack segment to coincide with the highest address (`FFFF` in the figure). Then as the stack grows during runtime, it grows into the free space found it the gap.

The data segments are subdivided into initialized and uninitialized data sections, and a third section for heap allocation. The initialized and uninitialized data sections have fixed size. Like the stack, the heap can grow during run-time. So it is arranged to be adjacent to the gap of free space.

During runtime, we get an error when the gap between the heap and stack is reduced to nothing. We either have a "stack overflow" or an error while attempting to obtain space from the heap.

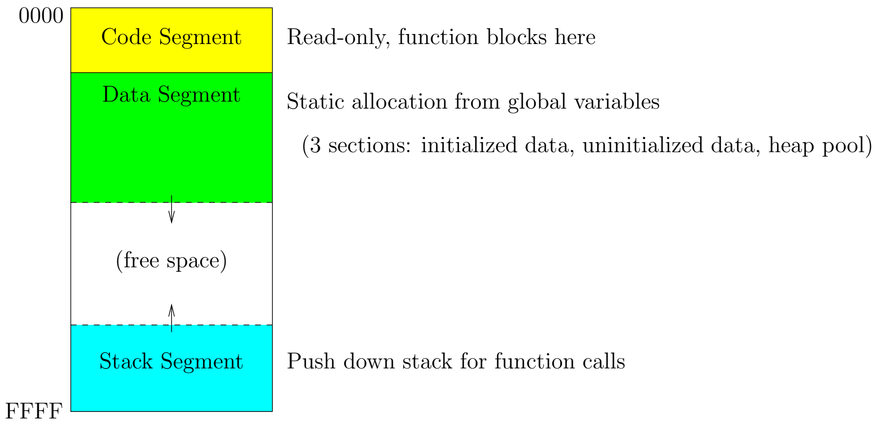

Figure 1: Process Image

3 System Calls

In Unix and C, there is no difference between a system call and an ordinary function call!

```
/**************************************************
 * io.c: minimal cat version (from Kernighan and Pike)
 * This program will echo whatever you type.
 * To terminate, type control-D
 **************************************************/

#define SIZE 512 // arbitrary

int main() {
char buf[SIZE];
int n;

while ((n = read(0, buf, sizeof buf)) > 0)
write( 1, buf, n);
return(0);
}
```

The function `read(...)` is actually a system call.