11 Variable Length Argument List

Programming in C

Teacher: Po-Wen Chi
neokent@gapps.ntnu.edu.tw

January 5, 2019

Department of Computer Science and Information Engineering,
National Taiwan Normal University

Program Argument

Linux Command

- You have used Linux at least one semester.
- Many commands can accepts lots of options. Can you give some examples?

Linux Command

- You have used Linux at least one semester.
- Many commands can accepts lots of options. Can you give some examples?
- How to do this?

Example

Please see example.11.main.

Example

Please see example.11.main.

The problems are ...

- who calls the main function.
- who provides *argc* and *argv* to the main function?

Is main() the actual starting entry point?

NO!

Actually, the process start from _start which is defined in **crt1.o**.

Is main() the actual starting entry point?

NO!

Actually, the process start from _start which is defined in **crt1.o**.

Wait a minute! I do not use crt1.o ...

Is main() the actual starting entry point?

NO!

Actually, the process start from **_start** which is defined in **crt1.o**.

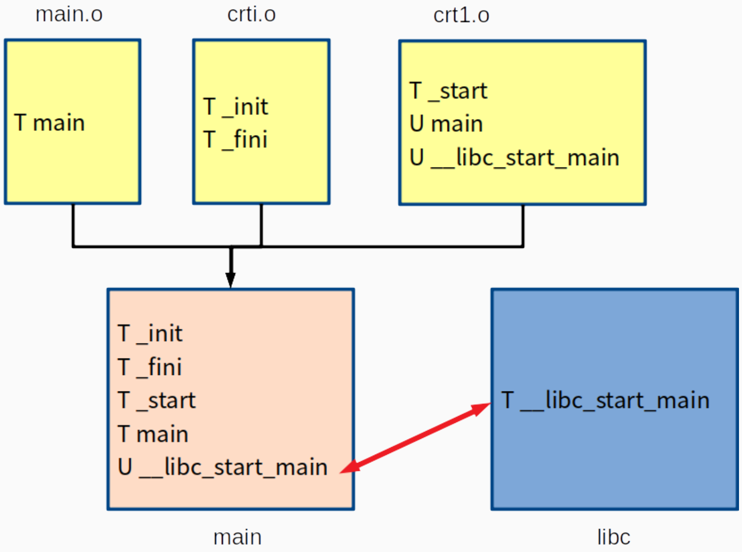
Wait a minute! I do not use crt1.o ...

Actually you have. Please try the command gcc -v.

crt1.o

```
$ nm /usr/lib/x86_64-linux-gnu/crt1.o
 0000000000000000  D   __data_start
 0000000000000000  W   data_start
 0000000000000030  T   _dl_relocate_static_pie
                   U   _GLOBAL_OFFSET_TABLE_
 0000000000000000  R   _IO_stdin_used
                   U   __libc_csu_fini
                   U   __libc_csu_init
                   U   __libc_start_main
                   U   main
 0000000000000000  T   _start
```

Simplified C Linking Process



How main Gets argc and argv?

- Actually, _start will prepare *argc* and *argv* first.
- Then _start will register main function to __libc_start_main.
- _start will call __libc_start_main and __libc_start_main will call main.

More details will be described in *Assembly*.

Back to the Program

- OK, now I know how to get user options. The next step should be string processing.
    - Undoubtedly, yes.
    - Please see ifconfig.c
    - However, there are some useful tools for you.

getopt

int getopt(int argc, char * const argv[], const char *optstring);

The getopt() function parses the command-line arguments. Its arguments argc and argv are the argument count and array as passed to the main() function on program invocation. An element of argv that **starts with '-'** (and is not exactly "-" or "–") is an option element. The characters of this element (aside from the initial '-') are option characters. If getopt() is called repeatedly, it returns successively each of the option characters from each of the option elements.

Reminder: This is a **POSIX standard** instead of **C standard**.

Extern Variables

- extern char *optarg;
- extern int optind, opterr, optopt;

This implies we can use these variables in our code.

Example

Please see example.11.getopt

Please use the following commands:

- ./test -a -b -c
- ./test -abc
- ./test -d

optstring

- **optstring** is a string containing the legitimate option characters.
  - Example: "abc" implies supporting -a, -b, -c.
  - -ab, -bc, -ac, -abc are also supported.

Return Values

- If an option was successfully found, then getopt() returns the option character.
- If all command-line options have been parsed, then getopt() returns -1.
- If getopt() encounters an option character that was not in optstring, then '?' is returned.
- If getopt() encounters an option with a missing argument, then the return value depends on the first character in optstring:
    - If it is ':', then ':' is returned;
    - Otherwise '?' is returned.

optstring

- If such a character is followed by a **colon**, it means the option requires an argument. So getopt() places a pointer to the following text in the same argv-element, or the text of the following argv-element, in **optarg**.
- Please see the example code.
- Try the following command:
  - ./test -d 1
  - ./test -d2

Experience Sharing

- Generally speaking, we do not write too many codes in the switch block.
- We often only set flags in the switch block.
  - optionA = 1;
- More works will be done after the switch block.
- Please see nmap.c

That is Not Good Enough

```
用法：ls [選項]... [檔案]...
List information about the FILEs (the current directory by def
Sort entries alphabetically if none of -cftuvSUX nor --sort is

Mandatory arguments to long options are mandatory for short op
  -a, --all                  do not ignore entries starting wi
  -A, --almost-all           do not list implied . and ..
      --author               with -l, print the author of each
  -b, --escape               print C-style escapes for nongrap
```

We know how to support -a. How about –all?

Long Commands

int getopt_long(int argc, char * const argv[],
const char *optstring,
const struct option *longopts, int *longindex);

The getopt_long() function works like getopt() except that it also accepts long options, started with two dashes. If the program accepts only long options, then optstring should be specified as an empty string (""), not NULL. Long option names may be abbreviated if the abbreviation is unique or is an exact match for some defined option. A long option may take a parameter, of the form –arg=param or –arg param.

If longindex is not NULL, it points to a variable which is set to the index of the long option relative to longopts.

struct option

```
struct option {
    const char *name;
    int         has_arg;
    int        *flag;
    int         val;
};
```

- name: the name of the long option.
- has_arg:
    - 0: no arguments.
    - 1: required arguments.
    - 2: optional arguments.
- flag: specifies how results are returned for a long option.
- val: the value to return, or to load into the variable pointed to by flag.

Important Note

The last element of the array has to be filled with zeros.

My Suggestion

- It is very important to know how to use these functions if you want to develop an useful commands.
- Do not forget to provide **help**.

Practice

Please write a program called **wc**.

用法：wc [選項]... [檔案]...

Print newline, word, and byte counts for each FILE.

- -c, –bytes: print the byte counts
- -m, –chars: print the character counts
- -l, –lines: print the newline counts
- -w, –words: print the word counts

Variable Length Arguments

Can I Write a Function Like printf?

- **printf()** can accept lots of arguments. Argument number is not static.
- I want to do the same thing.
- Wait a minute, have you ever seen the declaration of **printf()**?

printf() Declaration

```c
int printf(const char *format, ...);
```

What does ... mean?

Variable Length Argument in C

- Variable length argument is a feature that allows a function to receive any number of arguments.
- Variable number of arguments are represented by three dotes `...` .
- Please see example.11.valist.

Variable Argument Lists

```
#include <stdarg.h>

void va_start(va_list ap, last);
type va_arg(va_list ap, type);
void va_end(va_list ap);
void va_copy(va_list dest, va_list src);
```

Actually, they are **macros** instead of functions.

Will You Explain How These Works?

No.

I will teach you Macro details later. And there will be a **homework** for you to explain why these macros works.

Practice

Please write a function to concatenate variable strings.