System Programming And C Language

Prof. Jin-soo Kim. (jinsookim@skku.edu)
Pintos TA – Jin-yeong, Bak. (dongdm@gmail.com)
Kyung-min, Go. (gkm2164@gmail.com)
2010.09.28
Contents

- Important thing in system programming.
  - Correctness
  - Reliability
  - Performance issue

- Design Issue.

- Some technique.
  - DEFINE Macro
  - inline prefix
  - struct bit field
  - another issue – using goto
Important things in system programming

- There are three important issues when make or improving computer program.
  - Correctness.
    - What if your program has calculated $1 + 1$ as $3$?
  - Performance
    - Time complexity
      » How long does it take to do the things?
      » What if your program does not respond on expected time?
    - Space complexity
      » How many space we need to perform those operation?
      » What if your program occupy more than we need?
  - Reliability
    - How can you guarantee your program’s safety from trivial bug?
    - What if some BAD BAD BAD Programmer has attacked your kernel? Your program must be able to survive from those attack.
Important things in system programming – Correctness

- **Exact output**
  - Your program must do exactly what you, someone or client expected. In case of us, Pintos project. 😊

- **Provability**
  - You must be able to prove your logic.
  - So you must **THINK** before rush in to the coding mode. → More detail on design issue.

- **If your program doesn’t work exactly, than your grade will be 0 -_- because of the grade program (actually, script)**
Important things in system programming - Performance

- **Time complexity**
  - What does it mean?
    - How long does it takes to calculate the answer on time?
  - If you can shorten your program, than do it.
    - BUT! Shortening is not good always. For example, arithmetic operation. vs. bit manipulation. on CPU.
  - Than how?
    - Answer is... suit to the CPU’s taste.
  - For example
    - Bit manipulation vs. Arithmetic manipulation
    - Natural number Operation vs. Real number Operation.
    - If you can consider cache property, then do it.
Important things in system programming - Performance

- Space Complexity
  
  - What does it means?
    - How many your program occupy the memory? (source code itself)
    - And how many your program use the memory? (on dynamic situation)
  
  - Your program must be able to use the memory space efficiently.
Important things in system programming. - Performance

- Trade off between Time vs. Space.
  - Did you learned DATA STRUCTURE?
    - metaphore) Hash vs. Linked list.
  - Programming...
    - memory allocation(malloc() function) vs. page allocation.
      » (Project 2, 3 will treat it more detail.)
  - On mathematics...
    - Induction proving. vs. Deduction proving
  - In coding...
    - Readability vs. Shortness.

- You must think those trade off well when choose some method.
Important things in system programming. - Reliability

- Reliability is one of the most important thing!
  - What if your program has dead when user has worked on your program?
    - It makes your client mess...
    - In case of you, you are messed when you do the job.
  - Case of some issue.
    - Privilege(Kernel & User Program)
    - Memory leaking (PAGE FAULT)
    - Insufficient Resource (More focus on is MEMORY)
    - Unexpected attack from networks. (We don’t treat network.)
    - When user has input wrong number – (It is so called data integrity – 무결성)
Important things in system programming. - Reliability

- trade off between performance vs. reliability
  - think about it what can be?
  - It is about exception processing
    - many if branches will degrade your program’s performance.
  - Synchronization issue.
    - Parallelize for speed. vs. Serialize for secure from data hazard
      » Data hazard : If you learned computer architecture...
Design Issue.

- When you design your program
  - First, analysis requirements carefully until you understand enough.
  - Second, spread many method as you can.
  - Third, compare those method with any tools you can use.
    - It is important thing because it improves your proving ability about those logic. → You can defense your algorithm from meeting with us.
    - Measure: Correctness, Efficiency (Time, Space), Reliability
  - Fourth, design algorithm.
  - Fifth, Coding start!
  - Sixth, Evaluation your program with debugging tools.
  - Seventh, if there is some unsatisfying result, go to appropriated step. And do it iterative.
  - Submit your program to TA. 😊
C Language technique – Basic

- In case of System programming
  - Bit manipulation

- In case of C Language
  - `#define` pre-processor
  - `inline` prefix
  - for memory saving with setting bit number on `struct`
  - goto operation (Re think. – positive position.)

- Coding style issue
C Language technique – Bit manipulation

- Why bit manipulation?
  - CPU Likes logical operation. ← It is designed to do that.

- Kind of operation (As you know...)
  - AND (&, &&)
    - It is used for mask operation.
    - Condition operation.
  - OR (|, ||)
    - It is used for set operation.
    - Condition operation.
  - XOR (^)
    - It is used for flip operation.
  - NOT (~, !)
    - It is used for flipping all operation.
    - Condition operation.
  - SHIFT (Arithmetic(>>, Logical (unsigned casting))
    - It is used for dividing, shifting by number two
C Language technique – #define Macro (1)

- DEFINE Macro

- What do you know about #define macro?

  ```c
  #define MAX 100
  #define ABS(X) (((X) > 0) ? (X) : -(X))
  ```

- Some hidden technique.

  ```c
  #define SWAP(X, Y) {
      *X = *X ^ *Y;
      *Y = *X ^ *Y;
      *X = *X ^ *Y;
  }
  ```
C Language technique – #define Macro (2)

• DEFINE Macro (Cont.)
  • You can also use more complicate define function as call – return
    #define testfunc(X, Y) ({
      int Z;
      Z = (X) + (Y);  // (X) + (Y)
      Z <<= 2;   // (X) + (Y) * 4
    Z;
  })
  • I will add some more things!
    – define condition.
    – #ifdef, #ifndef, #elseif, #else, #endif
    – Why?... Next slide
C Language technique – #define Macro (3)

- Condition Macro (Cont.)
  - If you use header file.
    - If you use directly...
      
      test.h main.c test.c
      ...
      #include <test.h> #include <test.h>

    - You may encounter compile error
  - You must modify it as below.

    test.h main.c test.c
    #ifndef __TEST_H__
    #define __TEST_H__
    ...
    #endif
    #include <test.h> #include <test.h>
C Language technique – #define Macro (4)

- Condition Macro (Cont.)
  - You can compile as two mode
    - DEBUG Mode
    - RELEASE Mode → It is well defined on Visual Studio

  - Ex)
    
    ```c
    #ifdef DEBUG
    ...
    #elseif RELEASE
    ...
    #endif
    ```

    - on GCC option → `gcc ... -DDEBUG=1` or `–DRELEASE=1`
C Language technique – 
#define Macro (5)

- Condition Macro (Cont.)

```c
#if 0
You can write long comment without any notation.
#endif
```
• **inline prefix**
  - What is it?
    - Usually, for reducing function call overhead on run-time.
    - It is used on small size code
      » swap function
      » some boolean function.
  - Let’s see some situation. (CSAPP Books...)
    - When your program call some function, your program use stack.
    - So if you use your own function recursively, stack usage may be increase.
    - Next page...
C Language technique – inline prefix (2)

Only the used memory regions have valid mapping entries in page tables.

Fixed in size until Project 3.

Address space:
- Code segment: 0x00000000 to 0x08048000
- Initialized data segment
- BSS
- User stack
- Kernel address space: 0xc0000000 = PHYS_BASE

CSE3008: Operating Systems | Fall 2010 | Jin-Soo Kim (jinsookim@skku.edu)
C Language technique – struct

- **about struct**
  
  ```c
  struct help
  {
      char test1;
  };
  - Its size is 1 Byte.

  struct help
  {
      int test1 : 2;
      int test2 : 2;
      int test3 : 2;
      int test4 : 2;
  };
  - Its size is also 1 Byte.
  ```
C Language technique – Another Issue on goto

- goto operation (positive aspective)
  - Did you learned “Goto is always bad”?
    - The Answer is No.
    - Why answer was no?
      » Because it decrease the aspect of structural programming which was introduced by Dijkstra
  - But in case of kernel programming, there might use the goto operation.
    - For example, if the program must do the same task before exit the procedure, it might be useful.

```c
if (error)
  goto label1;
...
  // Doing something. (Maybe it is long job to do.
label1:
...
  // Ending task.
```