CSE 2421/5042: Systems I

Low-Level Programming and Computer Organization

Introduction

Presentation A

Gojko Babić

Read carefully: Bryant Chapter 1
Study: Reek Chapter 2
Skim: Reek Chapter 1

08/22/2018

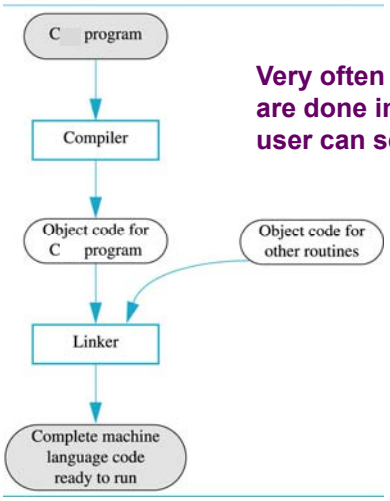
Some Basic Terms

- **Algorithm**
  - a sequence of precise instructions which leads to a solution;
- **Program**
  - an algorithm expressed in a language the computer can understand;
- **Source code**
  - the original program in a high level language;
- **Compiler**
  - translates programs in high-level language to machine language;
- **Object code**
  - the (compiler) translated version in machine language;
- **Linker**
  combines
  - the object code for the programs we write,

  and
  - the object code for the pre-compiled routines,

  into
  - the executable code the CPU can run;

Preparing C Program for Running



Very often compilation and linking are done in one step, as far as a user can see.

Layout of a Simple C Program

```
/* include statements; if any
define statements; if any  */

int main( )                    // main function is mandatory
{
      Variable_Declarations

      Statement_1
      Statement_2
        ...
      Statement_Last

      return 0;    // if you do not include this statement,
}                  // compiler will do it automatically
```

- The source code of a C program is stored in one or more files;
- Each function has to be completely contained in one file;
- Each source file is individually compiled to produce an object file;

Main Memory

- Long list of memory locations
  - each contains some pattern of zeroes and ones
  - can change during program execution
- Binary Digit or Bit: a digit that can only be zero or one
- Byte: each memory location has eight bits
- Address: a number that identifies a memory location
- Some data is too large for a single byte
  - e.g. most integers and real numbers are too large
  - next few consecutive bytes can store the additional bits for larger data
  - address of larger data refers to the first byte

Encoding Byte Values

- Byte = 8 bits
  - Binary (base 2 representation): 00000000_2 to 11111111_2
    - ✓ Use characters '0' and '1' → e.g. 01100101
  - Decimal (base 10 representation): 0_{10} to 255_{10}
    - ✓ Use characters '0' to '9' → e.g. 127
  - Hexadecimal (base 16 representation): 00_{16} to FF_{16}
    - ✓ Use characters '0' to '9' and 'A' to 'F' → e.g. 0x7D
  - Octal (base 8 representation): 000_8 to 377_8
    - ✓ Use characters '0' to '7' → e.g. 075
  - Write $FA1D37B_{16}$ in C → 0xFA1D37B or 0xfa1d37b → ? bits
  - Write 114376_8 in C → 0114376 → ? bits

Unsigned Integer Encoding

- Here is how in C an 16-bit unsigned integer variable y can be declared and initialized to 15,213:
  short unsigned int x = 15213;
- But how is y represented as a binary number?

Let binary number X have w-bit pattern: $x_{w-1}\ x_{w-2}\ x_{w-3}\ \ldots\ x_3\ x_2\ x_1\ x_0$
x_i = 0 or 1

Binary to Unsigned decimal number interpretation:

$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101

Unsigned Integer Encoding Examples

x = 15213: 00111011 01101101

Weight	15213	
1	1	1
2	0	0
4	1	4
8	1	8
16	0	0
32	1	32
64	1	64
128	0	0
256	1	256
512	1	512
1024	0	0
2048	1	2048
4096	1	4096
8192	1	8192
16384	0	0
32768	0	0
Sum		15213

The hello program

```
# include <stdio.h>

int main()
{
   printf("hello, world\n");
   return 0;
}
```

***ASCI text* representation of hello.c program:**

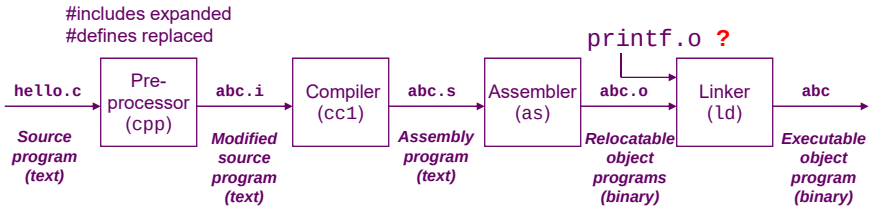
```
 #    i    n    c    l    u    d    e <sp>  <    s    t    d    i    o    .
35  105  110  99  108  117  100  101  32   60  115  116  100  105  111  46

 h    >   \n   \n    i    n    t  <sp>  m    a    i    n    (    )   \n    {
104  62   10   10  105  110  116   32  109   97  105  110   40   41   10  123

\n  <sp> <sp> <sp> <sp>  p    r    i    n    t    f    (    "          …
10   32   32   32   32  112  114  105  110  116  102   40   34          …
```

ASCII 7-bit code; ASCII is a part of UTF-8 that uses up to 4 bytes.

Compilation

- On Linux system, the translation from C program in the source file hello.c to executable object program in the file abc is performed by the following command:

  unix> gcc –o abc hello.c



- To run executable file on a Unix system, type its name:

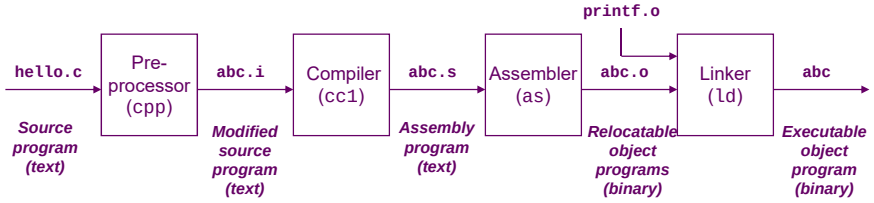
  > abc

  hello, world

  >

More About gcc Command

- –o *file* → put the output in *file*
  gcc –o abc hello.c
- -E → stop after preprocessing; do not compile
  gcc –E –o abc.i hello.c
- -S → stop after compilation; do not assembly
  gcc –S –o abc.s hello.c
- -c → stop after assembly, do not link
  gcc –c –o abc.o hello.c



hello.c x86-64 Assembly Code

```
#include <stdio.h>

int main()
{
 printf("hello, world\n");
 return 0;
}
```

```
main:
  subq  $8, %rsp
  movl  $.LC0, %edi
  call  puts
  movl  $0, %eax
  addq  $8, %rsp
  ret
```

➢ This Linux gcc command:
gcc -S –O1 hello.c
produces x86-64 assembly code (64 bit code as default in our system) in file hello.s (on right) from C code in the file hello.c (on left) .
Note: switch –O1 sets level 1 optimization

hello.c x86 Assembly Code

```
#include <stdio.h>

int main()
{
 printf("hello, world\n");
return 0;
}
```

```
main:
  pushl%ebp
  movl %esp, %ebp
  andl $-16, %esp
  subl $16, %esp
  movl $.LC0, (%esp)
  call puts
  movl $0, %eax
  leave
  ret
```

- This Linux gcc command:
  `gcc -S –O1 –m32 hello.c`

produces x86 assembly code (32 bit code) in file hello.s (on right) from C code in the file hello.c (on left).
Note: switch –m32 produces 32 bit code

64-bit Executable Code of hello.c Main Functior

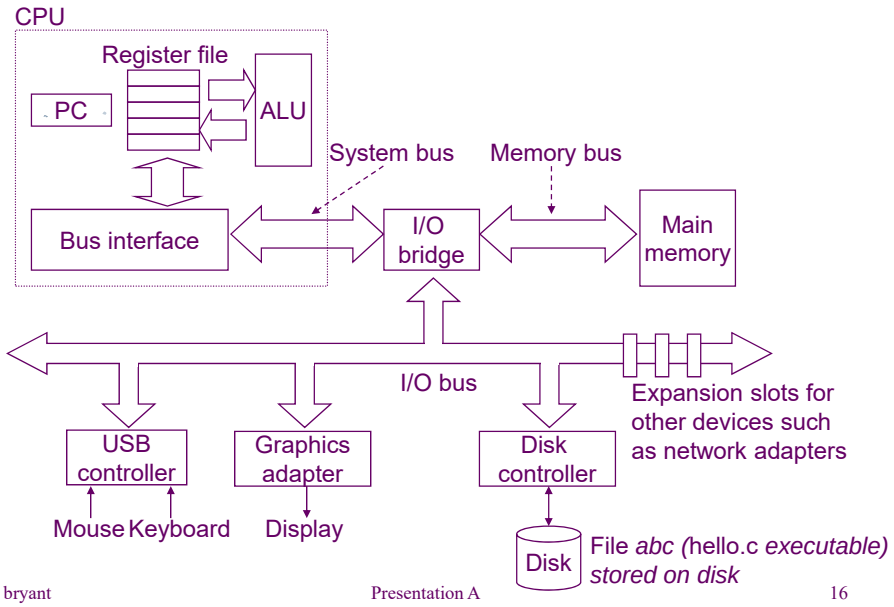
```
0000000004004c4 <main>:
 4004c4:   48 83 ec 08             sub    $0x8,%rsp
 4004c8:   bf d8 05 40 00          mov    $0x4005d8,%edi
 4004cd:   e8 e6 fe ff ff          callq  4003b8 <puts@plt>
 4004d2:   b8 00 00 00 00          mov    $0x0,%eax
 4004d7:   48 83 c4 08             add    $0x8,%rsp
 4004db:   c3                      retq
 4004dc:   90                      nop
 4004dd:   90                      nop
 4004de:   90                      nop
 4004df:   90                      nop
```

32-bit Executable Code of hello.c Main Function

```
080483b4 <main>:
80483b4:   55                          push   %ebp
80483b5:   89 e5                       mov    %esp,%ebp
80483b7:   83 e4 f0                    and    $0xfffffff0,%esp
80483ba:   83 ec 10                    sub    $0x10,%esp
80483bd:   c7 04 24 94 84 04 08        movl   $0x8048494,(%esp)
80483c4:   e8 27 ff ff ff              call   80482f0 <puts@plt>
80483c9:   b8 00 00 00 00              mov    $0x0,%eax
80483ce:   c9                          leave
80483cf:   c3                          ret
```

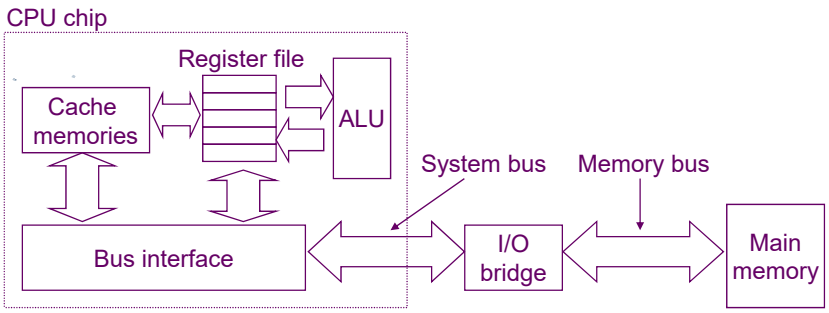
15

Typical Computer Organization



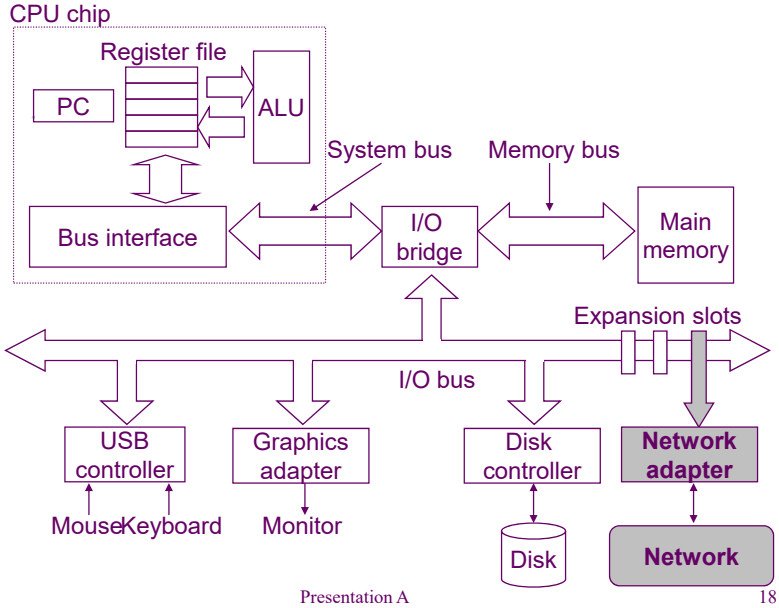
bryant Presentation A 16

Cache memory



Main memory size: in Giga bytes; 1GB = ??
Cache size: in Mega bytes; 1MB = ??
Disk size: in Tera bytes; 1TB = ??

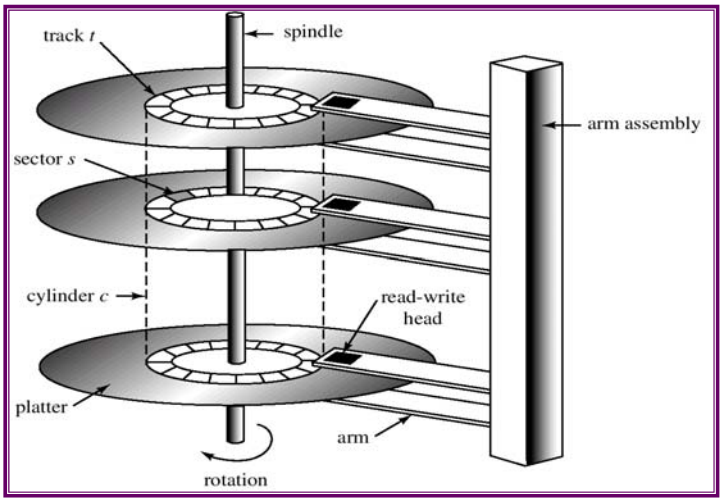
Network Is Just Another I/O Device



Operating System

- Protect the computer from misuse,
- Provide an abstraction for using the hardware so that programs can be written for a variety of different hardware,
- Operating system provide to user programs service of accessing I/O devices: keyboard, display, disc and network,
- Manage the resources to allow for reasonable use by all users and programs on a computer.
- Programs are often written as if they are the only things running on a system and OS allows them to work this way by providing an abstraction known as a process,
- Process is a running program (one or more threads of control), along with all the data associated with it ,
- OS uses context switching to give the appearance of multiple processes executing at once on a single processor.

Moving-Head Disk Mechanism



A sector size = 512B,
A sector (***only***) is a unit of transfer between memory and disc.

Reality No. 1: Computer Arithmetic

- Is $x^2 \geq 0$?
  - Floating point numbers: Yes!
  - But, what are floating point numbers?
  - Integers:
    - 40000^2 → 1600000000
    - 50000^2 → ??
- Is (x + y) + z = x + (y + z)?
  - Integers: Yes!
  - Floating point numbers:
    - (1e20 – 1e20) + 3.14 --> 3.14
    - 1e20 + (– 1e20 + 3.14) --> ??

Reality No. 2: Assembly Programming

- Chances are, you'll never write programs in assembly

  Compilers are much better
- But: Understanding assembly is key to machine-level execution model
  - Behavior of programs in presence of bugs
    - ✓high-level language models break down
  - Tuning program performance
    - ✓understand optimizations done (or not done) by the compiler
    - ✓ understanding sources of program inefficiency
  - Implementing system software

Reality No. 3: Memory Matters

- Memory is not unbounded
  - ✓ it must be allocated and managed
  - ✓ many applications are memory dominated
- Memory referencing bugs especially pernicious
  - ✓ effects are distant in both time and space
- Memory performance is not uniform
  - ✓ Cache, main memory and disk effects can greatly affect program performance
  - ✓ adapting program to characteristics of memory system can lead to major speed improvements

Memory Reference Errors

- C (as well as and C++) doesn't provide memory protection:
  - ✓ out of bounds array references
  - ✓ invalid pointer values
  - ✓ abuses of C functions malloc/free
- Can lead to nasty bugs and whether or not bug has any effect depends on system and compiler
  - ✓ corrupted object logically unrelated to one being accessed
  - ✓ effect of bug may be first observed long after it is generated
- How can I deal with this?
  - ✓ understand what possible interactions may occur

Reality No. 4: About Performance

```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{ int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

4.3ms

```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

81.8ms

- Programs above run on 2.0 GHz Intel Core i7 Haswell

The AdHello program

```
#include<stdio.h>
int main()
{
char c1;
printf("Enter 'a', 'b' or 'c':");
scanf("%c", &c1);
if (c1=='a') printf("hello A, world\n");
  else if (c1=='b') printf("hello B, world\n");
    else if (c1=='c') printf("hello C, world\n");
      else printf("Wrong input\n");
return 0;
}
```

Monday In Class Assignment

- Caldwell Lab room CL112
- Logon to Windows and then Linux.
- Create directory Cse2421 and subdirectory A1.
- In A1 directory, using one of Linux editors create hello.c; include your name at the beginning of the file.
- Compile hello.c and run it; you should get expected results.
- In the same directory, using one of Linux editors create AdHello.c; include your name at the beginning of the file.
- Compile AdHello.c and run it; you should get expected results.
- At the end of class, turn in hard copies of both source files.
- Also, electronically submit hello.c and AdHello.c using Carmen at Testing1. To activate Carmen on Linux server first issue command: %firefox & to run browser.

27