CDOT
SICT AR Meeting Area
People

get involved with CDOT

as a Student
as an Open Source Community Member
as a Company

courses

BTC640
BTH740
BTP300
DPI908
DPS901
DPS905
DPS909
DPS911
DPS914
DPS915
DPS924
DPS931
EAC234
ECL500
GAM531
GAM666
GAM670
GPU610
LUX Program
MAP524
OOP344
OPS235
OPS245
OPS335
OPS345
OPS435
OPS445
OPS535
OPS635
OSD600
OSD700
OSL640
OSL740
OSL840

ULI101 Week 4

Contents [hide]

Data Representation [edit]

Why Study Data Representation? [edit]

- Computers process and store information in binary format
- For many aspects of programming and networking, the details of data representation must be understood
- C Programming - sending information over networks, files
- Unix / Linux - setting permissions for files and directories
- Web Pages - setting color codes

Data Representation [edit]

- In terms of this course, we will learn how a simple decimal number (integer) is stored into the computer system as a binary number.
- We will also learn other numbering systems (octal and hexadecimal) that can be used as a “short-cut” to represent binary numbers.
- Before we learn numbering systems, we have to “goback in time” to see how we learned the decimal numbering system.
- The decimal numbering system (base 10) uses 10 symbols for each digit (0, 1, 2, … 9). Since most humans have 10 extensions on their hands (2 thumbs, 8 fingers), many suspect that is why humans work with decimal numbers.

Real World Mozilla
RHT524
SBR600
SEC520
SPO600
SRT210
ULI101

course projects

Course Project List
Potential Course Projects
Contrib Opportunities

links

CDOT
Planet CDOT
FSOSS

Tools

What links here
Related changes
Upload file
Special pages
Printable version
Permanent link
Page information

Decimal Numbers [edit]

In grade school, we probably learned to break-down, as follows, the decimal `3572`:

```
3 thousands
5 hundreds
7 tens
2 ones
```

Another way to look at this number is multiplying the digit by 10 (the numbering base) raised to increasing powers (starting at 0 from the “ones” and moving towards the higher digits)

```
3 thousands = 3 x 10^3 = 3 x 1000
5 hundreds = 5 x 10^2 = 5 x 100
7 tens = 7 x 10^1 = 7 x 10
2 ones = 2 x 10^0 = 2 x 1
```

This way of understanding decimal numbers is the basis for math operations such as addition, subtraction, multiplication, decimal numbers, etc!

Binary Numbers [edit]

We can use a similar method to convert a binary number to a decimal number. We do the same thing in the previous slide, but we multiply by base 2 instead of base 10. Take the binary number `1101`:

```
  1 x 2^3 = 1 x 8 = 8
+ 1 x 2^2 = 1 x 4 = 4
+ 0 x 2^1 = 0 x 2 = 0
+ 1 x 2^0 = 1 x 1 = 1
---------------------
Sum of 8 + 4 + 0 + 1 = 13
```

Remember start from the right-hand-side and move to the left. Therefore, `1101` in binary is `13` in decimal. For programmers, the 8-bit binary number `00001101` represents the unsigned integer 13.

Octal Numbers [edit]

The octal numbering system (base 8) uses 8 symbols for each digit (0, 1, 2, … 7). We can use the same process to convert an octal number to a decimal number (but use base 8 instead). Convert the octal number `2741` to decimal:

```
  2 x 8^3 = 2 x 512 = 1024
+ 7 x 8^2 = 7 x 64 = 448
+ 4 x 8^1 = 4 x 8 = 32
+ 1 x 8^0 = 1 x 1 = 1
---------------------
Sum of 1024+448+32+1 = 1505
```

Remember, start from the right-hand-side and move to the left. Therefore, `2741` in octal is `1505` in decimal.

Hexadecimal Numbers [edit]

The hexadecimal numbering system (base 16) uses 16 symbols for each digit (0, 1, 2, … 9, A, B, C, D, E, F). Why use letters? Because we are only human and we need to use letters to represent higher digits 10 - 15 as a single digit! Let’s convert the hexadecimal number F2A to decimal:

```
F x 16^2 = 15 x 16^2 = 15 x 256 = 3840
2 x 16^1 = 2  x 16^1 =  2 x 16 = 32
A x 16^0 = 10 x 16^0 = 10 x 1  = 10
----------------------------------
Sum of 3840+32+10 = 3882
```

Therefore, `F2A` in Hexadecimal is `3882` in decimal. I can understand now how decimal numbers can be stored in the computers as binary numbers, but why are we learning Octal and Hexadecimal numbers? As computers and computer programming languages evolved, octal and hexadecimal numbers were considered “short-hand” a short-cut to represent binary numbers.

- Each octal digit represents 3 binary digits.
- Each hexadecimal digit represents 4 binary digits.
- Linux/Unix operating system commands, networking specialists, programming analysts as well as car-crash investigators use these types of shortcuts which help save space and time issuing a command. Cars provide hexadecimal codes to record info prior to impact. Hexadecimal numbers can refer to memory addresses which point to incorrect programming procedure.
- `chmod 700 secretfile` Unix/Linux command to allow file read, write and execute access to the file’s owner only.
- You will be converting between any number system whether it is from binary to decimal, binary to octal, decimal to binary, octal to hexadecimal, etc.
- The next series of slides provide interesting shortcut how to perform these numbering system conversions. The symbol ^ is used to represent “raised to the power of..”. For Example: 10^3 = 10^3

Converting Binary to Octal [edit]

- Convert the binary number `111110000` to an octal number:

```
= 1 1 1             1 1 0          0 0 0
x 2^2 2^1 2^0       2^2 2^1 2^0    2^2 2^1 2^0
i.e. (4) (2) (1)    (4) (2) (1)    (4) (2) (1)
1x4+ 1x2+ 1x1       1X4+ 1x2+ 0x1  0X4+ 0x2+ 0x1
= (4+2+1) = 7       (4+2+0) = 6    (0+0+0) = 0
```

Therefore, the binary number 111110000 represents 760 as an octal number. This code can be used to represent directory and file permissions (you will learn how to set permissions soon)

Remember:

1 octal digit is equal to 3 binary digits. Group binary digits into groups of 3 starting from the right. Add leading zeros if left-most group has less then 3 digits. Convert each group of 3 digits to an octal digit.

Converting Octal to Binary [edit]

- Similar to previous calculation, but in reverse:
- Convert octal number `760` to binary.

```
7 6 0
(4)(2)(1) (4)(2)(1) (4)(2)(1)
1 1 1 1 1 0 0 0 0
= 111110000
```

“Spread-out” octal number to make room for binary number result. Determine digits (0’s or 1’s) that are required when multiplied by appropriate power of 2 to add up to octal digit.

Converting Binary to Hex [edit]

- Convert the binary number `111110000` to a hexadecimal number:

```
= 0 0 0 1           1 1 1 1             0 0 0 0
(8) (4) (2) (1)    (8) (4) (2) (1)    (8) (4) (2) (1)
1                  15                  0
1                  F                   0
```

Therefore, the binary number `111110000` represents `1F0` as a hexadecimal number.

> 1 hexadecimal digit is equal to 4 binary digits. Group binary digits into groups of 4 starting from the right. Add leading zeros if last group of digits is less than 4 digits. Convert each group of 4 digits to a hexadecimal digit.

Converting Hex to Binary [edit]

- Similar to previous calculation, but in reverse:
- Convert hexadecimal number `1F0` to binary.

```
  1 F 0
= 1            15            0
= (8)(4)(2)(1) (8)(4)(2)(1)  (8)(4)(2)(1)
= 0 0 0 1      1 1 1 1       0 0 0 0
= 000111110000 = 111110000
```

“Spread-out” hex number to make room for binary number result. Determine digits (0’s or 1’s) that are required when multiplied by appropriate power of 2 to add up to hexadecimal digit.

Converting decimal to binary [edit]

- Convert `78` to a binary number

- List the powers of 2 (until greater than or equal to 78) Start with the highest number equal or just less than 78. Put a binary digit “1” below that number and subtract that decimal equivalent from 78 (eg. 78 - 64 = 14). Repeat the same step for the remainder until result is zero. Any numbers NOT used become binary digit “0”

```
64 32 16 8 4 2 1
1
78-64=14
1
14-
8=6
1
6-4=2
1
2-2=0
0 0 0
```

File Permissions [edit]

As you may recall from our previous notes, that Unix/Linux recognizes everything as a file:

- Regular files to store data, programs, etc…
- Directory files to store regular files and subdirectories
- Special Device files which represent hardware such as hard disk drives, printers, etc… You may ask, “Since I can navigate throughout the Unix/Linux file system - what prevents someone from removing important files on purpose or by accident?” Answer: Ownership of the file, and file permissions.
- In previous classes, you only noted a few items from a detailed listing such as type of file, file size and date of creation/modification. Let’s look at the following detailed listing of a device (a harddisk partition) located in the /dev (devices) directory and explore more items: Let’s explore the results of this detailed listing in the next slide

```
[username] ls -l /dev/hda
brw-r----- 1 root disk 3,0 2003-03-14 08:07 /dev/hda
```

This indicates the user who “owns” the file. In this case, the superuser or “root” probably created the file…

- File Type (i.e. “b” or “c” for device file, “-” for regular files, “d” for directory file)
- File Permissions (i.e. what permissions are granted by the owner regarding file access, file modification, and/or file execution).
- In this case, the owner (in this case root) can access (read) the file, the owner can modify (write) the file, but a dash instead of an “x” means that the owner cannot run (execute) the file like a program…
- OK, I can now see that the owner (root) is the only user that has permissions to make changes (write) to the file /dev/hda, so no other user can damage or edit and save changes to that file. But what if an owner of a file wanted other users to view or write to their file? Can the owner of the file allow access to some users, and not to others? Answer: That is what the other 2 sets of permissions are for.

```
[joe.professor] ls -l ~/work_together
-rw-rw---- 1 joe.professor users 0 2006-02-02 10:47
~/work_together
# This indicates the user "joe.professor" owns the file
"work_together".
# The owner "joe.professor" can read and write to
"work_together"
# joe.professor can change ownership of work_together (using
chown command).

# Let's look at the detailed listing for a regular file owned by
someone else:
[joe.professor] ls -l ~/work_together
-rw-rw---- 1 joe.professor users 0 2006-02-02 10:47
~/work_together

# user "joe.professor" has given permission to anyone in "users"
group
# to read from and write to (incl. delete) the file
"work_together".

[joe.professor] ls -l ~/work_together
-rw-rw---- 1 joe.professor users 0 2006-02-02 10:47
~/work_together
# What does this last set of permissions refer to?
# Answer: all "other" users - users that DO NOT belong to the
"users" group.!
```

Directory Permissions [edit]

- We use the same letters for permissions as for regular files and permissions are assigned for owner, group, and others
- However, since a directory is a special kind of file which holds lists of other files, permissions work differently than for regular files:

r allows listing contents of the directory

w allows creating and deleting files inside

x allows access to files inside

- In order to have access to directory contents, at least the "x" permission is necessary.
- This is called the "pass-through" permission. The pass-through permission is the key to grant access to only selected directories and/or files.

  Consider this example- you are giving others access to the following: `/home/you/documents/uli101/jokes.txt`
- The following directories need pass-through permissions set by you for others: you, documents, uli101
- Even if the above directories have other files, possibly readable by others, they would have to know/guess their name before accessing them
- Just in case, you should not grant access permissions to others by default.
- Only in specific case, such as this one (`jokes.txt`), give read permissions to specific

files

Changing Permissions via chmod command [edit]

`chmod [permissions] file(s)`

- Can be used to change permissions for directories and regular files.
- There are two ways to set [who][operation][permission]:

Symbolic Method (using characters)

Absolute Method (using Octal Numbers)

Symbolic Method [edit]

- Permissions are set for: user (u), group (g), others (o), or all (a)
- Permissions are set through: adding (+), removing (-) and/or setting (=)
- Permissions are set to: read (r), write (w) and execute (x)

1. Examples:

Add Permission	`chmod g+rw file1`
Remove Permission	`chmod g-w *txt`
Set Permission	`chmod o=rx /tmp/xyz`
Combined	`chmod u=rwx g+x o` .=

2. Octal Method

You can use the chmod command with 3 octal number to represent permissions for user, group and others In this method, each permission has a numerical value:

r = 4

w = 2

x = 1

The resulting/intended permission is the sum of the above, for example “rw” permission has a value of “6”, for example:

```
chmod 755 abc  # abc permissions after: rwxr-xr-x
chmod 531 abc  # abc permissions afer: r-x-wx--x
```

umask [edit]

- Sets default file permissions for new files and directories in the current shell
- How does it work? umask permissions

For example: umask 467 Represents later: chmod 310

- Has the reverse effect to chmod - you set permissions which you do not want
- Each file permission is a subtraction result:

```
default 7 7 7
umask 4 6 7
```

```
= = =
result 3 1 0 (for a directory)
```

- For ordinary files any execute permissions are no applied
- For example: umask 310 would result in permissions 466 (r--rw-rw-)

Categories: ULI101 | ULI101-2018

This page was last edited on 4 September 2019, at 20:30.

Privacy policy About CDOT Wiki Disclaimers Mobile view