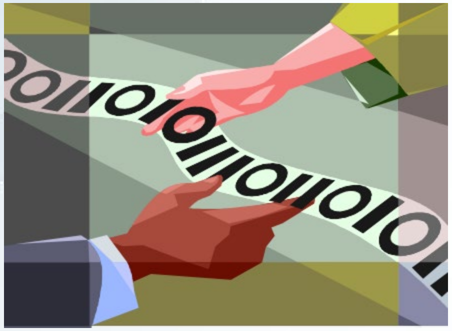
ULI101

Week 04

Week Overview

- Data Representation
- Binary, octal, decimal and hexadecimal numbering systems
- Number conversions
- Unix file permissions

Data Representation



Why Study Data Representation?

- 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

- 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.

Data Representation

- Before we learn numbering systems, we have to "go-back 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 digits on their hands (2 thumbs, 8 fingers), many suspect that is why humans work with decimal numbers.

Data Representation

Decimal Numbers

- Back in grade school we learn how to understand decimal numbers. For example, take the decimal number 3572. In grade school, we probably learned to break-down this number as follows:

3 thousands
5 hundreds
7 tens
2 ones

Data Representation

Decimal Numbers

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

3 thousands $= 3 \times 10^3 = 3 \times 1000$
5 hundreds $= 5 \times 10^2 = 5 \times 100$
7 tens $= 7 \times 10^1 = 7 \times 10$
2 ones $= 2 \times 10^0 = 2 \times 1$

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

Data Representation

Binary Numbers

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

$1 \times 2^3 = 1 \times 8 = 8$
$1 \times 2^2 = 1 \times 4 = 4$
$0 \times 2^1 = 0 \times 2 = 0$
$1 \times 2^0 = 1 \times 1 = 1$

$+ \quad \overline{}$
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 can represent the unsigned integer 13!

Data Representation

Octal Numbers

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

$2 \times 8^3 = 2 \times 512 = 1024$

$7 \times 8^2 = 7 \times 64 = 448$

$4 \times 8^1 = 4 \times 8 = 32$

$1 \times 8^0 = 1 \times 1 = 1$

+

1505

Remember, start from the right-hand-side and move to the left.

Therefore, 2741 in octal is 1505 in decimal.

Data Representation

Hexadecimal Numbers

- 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 it is convenient to represent higher digits 10 – 15 as a single character! Let's convert the hexadecimal number F2A to decimal:

$F \times 16^2 = 15 \times 16^2 = 15 \times 256 = 3840$

$2 \times 16^1 = 2 \times 16^1 = 2 \times 16 = 32$

$A \times 16^0 = 10 \times 16^0 = 10 \times 1 = 10$

$+$ ______

3882

Therefore, F2A in Hexadecimal is 3882 in decimal.

Data Representation

- You should 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.

  For example:
  - Each octal digit represents exactly 3 binary digits.
  - Each hexadecimal digit represents exactly 4 binary digits.

Data Representation

- Linux/Unix administrators, networking specialists, programming analysts, and car-crash investigators use these types of shortcuts which help save space and time issuing a command.

chmod 700 secretfile

Unix/Linux command to allow file read, write and execute access to the file's owner only!

Hexadecimal numbers can refer to memory addresses which point to incorrect programming procedure!

Cars provide hexadecimal codes to record info prior to impact!

Data Representation

- 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 shortcuts for performing 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

Convert the binary number 11110000 to an octal number:

=		0	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)
		0x4+	1x2+	1x1	1X4+	1x2+	0x1	0X4+	0x2+	0x1
=			3			6			0	

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.

Therefore, the binary number 11110000 represents 360 as an octal number. This code can be used to represent directory and file permissions (you will learn how to set permissions soon)

Converting Octal to Binary

Similar to previous calculation, but in reverse:

Convert octal number 760 to binary.

3	6	0
(4)(2)(1)	(4)(2)(1)	(4)(2)(1)
0 1 1	1 1 0	0 0 0

= 011110000

"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

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

Note:

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.

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

Converting Hex to Binary

Similar to previous calculation, but in reverse:

Convert hexadecimal number 1F0 to binary.

1	F	0
1	15	0

"Spread-out" hex number to make room for binary number result.

(8)(4)(2)(1)	(8)(4)(2)(1)	(8)(4)(2)(1)
0 0 0 1	1 1 1 1	0 0 0 0

Determine digits (0's or 1's) that are required when multiplied by appropriate power of 2 to add up to hexadecimal digit.

= 000111110000 = 111110000

Data Representation

Converting decimal to binary

Example: 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	0	0	1	1	1	0
78-64=14			14-8=6	6-4=2	2-2=0	

File Permissions

As you may recall from our previous notes, 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

File Permissions

In previous classes, you may have 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 hard-disk partition) located in the /dev (devices) directory and explore more items:

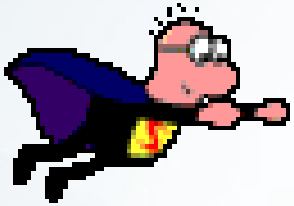
```
[username] ls -l /dev/hda
brw-r----- 1 root disk 3,0 2003-03-14 08:07 /dev/hda
```

Let’s explore the results of this detailed listing in the next slide

File Permissions

`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 Permissions

```
brw-r----- 1 root disk 3,0 2003-03-14 08:07 /dev/hda
```

This indicates:

1. **File Type** (i.e. "**b**" or "**c**" for device file, "**-**" for regular files, "**d**" for directory)

2. **File Permissions** (i.e. what permissions are granted by the owner regarding file access, file modification, and/or file execution)

Let’s look at these permissions in more detail in the next slide…

File Permissions

File type

File permissions

`brw-r-----`

File Permissions

File type File permissions

```
brw-r-----
```

File owner permissions:

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….

File Permissions

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. Look at the next slide…

File Permissions

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
```

File Permissions

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
```

This indicates the user "joe.professor" owns the file "work_together". The owner "joe.professor" can read and write to that file.

By the way, you can change the ownership of files (using the chown command, assuming you own them)

File Permissions

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
```

This indicates a group name (called "users") that is assigned to that file "work_together".

File Permissions

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
```

In this case the user "joe.professor" has given permission to users that belong to the "users" group to read and write to the file "work_together".

File Permissions

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
```

What does this last set of permissions refer to?

Answer: all "other" users - users that <u>DO NOT belong</u> to the "users" group.!

Directory Permissions

- 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 within the directory, but only when combined with the "x" permission
  - x – allows access to files inside, called "pass-through" permission

Pass-Through Permission

Pass-through permission is the key to grant access to only selected directories and/or files

For example – you wish to give "others" access to:

`/home/you/documents/uli101/jokes.txt`

- Besides read permission for **jokes.txt**, the "other" group also needs pass-through permission for the directories **you**, **documents**, and **uli101**
- For security, you should not grant access permissions to others by default
- Give others permissions to files in specific cases only, such as in this example (**jokes.txt**)

File Permissions

Changing Permissions via **chmod** command

```
chmod permissions file(s)
```

- Can be used to change permissions for **directories** and **regular files**.
- There are two ways to set permissions:
  - `Symbolic Method (using alphabetic characters)`
    `- sometimes called relative method`
  - `Octal Method (using Octal Numbers)`
    `- sometimes called absolute method`

chmod Symbolic Method

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

Examples:

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

chmod 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 permission is the sum of the above, for example "rw" permission has a value of "6".

```
chmod 755 file - results in:  rwxr-xr-x
chmod 531 file - results in:  r-x-wx--x
```

umask

- Sets default permissions for newly created files and directories

- Octal only, but specifies permissions to remove:
  `umask permissions-to-remove`
  For example: **`umask 026`**
  is similar to: **`chmod 751`** (for a directory)

- Each file permission is a subtraction result:

default	7	7	7
	-	-	-
umask	0	2	6
	=	=	=
result	7	5	1 (for a directory)

- For ordinary files any execute permissions are then removed

- So for ordinary files, umask 026 would result in permissions 640 (rw-r-----)