CENG328
Operating Systems

Laboratory II
Unix Tutorial & C Review I

1. Unix File System

- All the stored information on a UNIX computer is kept in a filesystem.
- The place in the filesystem tree where you are located is called the current working directory.
- Every item in the UNIX filesystem tree is either a file, or a directory.
- A directory is like a file folder. A directory contained within another is called the child of the other. A directory in the filesystem tree may have many children, but it can only have one parent.
- A file can hold information, but cannot contain other files, or directories. The file is the smallest unit in which information is stored.

1. Unix File System

- The UNIX file system has several important features, it contains several types of file.
  - **Ordinary files:** This type of file is used to store your information, such as some text you have written or an image you have drawn. Files which you create belong to you - you are said to "own" them - and you can set access permissions to control which other users can have access to them. Any file is always contained within a directory.
  - **Directories:** A directory is a file that holds other files and other directories. You can create directories in your home directory to hold files and other sub-directories. Directories which you create belong to you, too.
  - **Special files:** This type of file is used to represent a real physical device such as a printer, tape drive or terminal. It may seem unusual to think of a physical device as a file, but it allows you to send the output of a command to a device in the same way that you send it to a file. For example:

    cat scream.au > /dev/audio

    This sends the contents of the sound file **scream.au** to the file **/dev/audio** which represents the audio device attached to the system.

1. Unix File System

- Pipes; UNIX allows you to link two or more commands together using a pipe. The pipe acts as a temporary file which only exists to hold data from one command until it is read by another. The pipe takes the standard output from one command and uses it as the standard input to another command.

  command1 | command2 | command3

  The **|** (vertical bar) character is used to represent the pipeline connecting the commands. With practice you can use pipes to create complex commands by combining several (two or more) simpler commands together.

- For example, analyze the following command:

  cat /etc/passwd | grep *student* | cut -d\: -f5

- This command consists of three seperate parts:
  - **cat /etc/passwd:** Displays all lines in /etc/passwd file and directs the output to the next command via 1st pipe.
  - **grep student:** Matches the previous output for lines containing the word "student" and directs the output to the next command via 2nd pipe.
  - **cut -d\: -f 5:** Splits the previous output with ':' character and selects the substring with 5th index.
- As a whole, this command will seek real name (if available) of a given Unix username.

1.2. Unix File System Structure

- The UNIX file system is organized as a hierarchy of directories starting from a single directory called **root** which is represented by a **/** (slash). Immediately below the root directory are several system directories that contain information required by the operating system. The standard system directories are given below. The details may vary between different UNIX systems.
  - **/ -** The root of ALL files and directories.
  - **/bin/ -** Executable system utilities, like ls, cp, rm.
  - **/boot/ -** The kernel program.
  - **/dev/ -** Where special device files are kept.
  - **/etc/ -** System configuration files and databases.
  - **/home/ -** Where the personal files and directories of all users are kept.
  - **/lib/ -** Operating system and programming libraries.
  - **/usr/bin/ -** Additional user commands.
  - **/root/ -** The home directory of super user. The contents of this directory is usually hidden for other users available on the system.

1.2. Unix File System Structure

- **/tmp/ -** System scratch files (all users can write here).
- **/usr/bin/ -** Additional user commands.
- **/usr/include/ -** Standard system header files.
- **/usr/lib/ -** More programming and system call libraries.

- **Home Directory:** Any UNIX system can have many users on it at any one time. As a user you are given a home directory in which you are placed whenever you log on to the system. Users' home directories are usually grouped together under a system directory such as **/home**. A large UNIX system may have several hundred users, with their home directories grouped in subdirectories according to some schema such as their organizational department.

- **Pathnames:** Every file and directory in the file system can be identified by a complete list of the names of the directories that are on the route from the root directory to that file or directory. Each directory name on the route is separated by a **/** (forward slash). For example: **/usr/bin/gcc**. This gives the full pathname starting at the root directory and going down through the directories usr and bin to the file gcc - the GNU c compiler.

2. C Review I

- Using **argc** and **argv** as command line arguments (code5.c):

```
#include <stdio.h>
int main(int argc, char *argv[ ])
{
     int i;
     for (i=0; i < argc; i++)
     printf("command line argument [%d] = %s \n",i, argv[i]);
}
```

  - Analyze the code.
  - Execute the code. What is the output and why?
  - Execute as:

    ./code5 <your name> <your surname> <your age>

  - Describe the functionalities of argc and argv.

2. C Review I

- Arrays, Pointers and Dynamic Memory Allocation
  - Pointer: A variable that contains the address of a variable. (code6.c)

```
#include <stdio.h>
int main(int argc, char *argv[])
{
     int x = 1, y = 2, z[10];
     int *ip; /* ip is a pointer to int */
     ip = &x; /* ip now points to x */
     printf("The address of pointer 'ip' is %p \n",&ip);
     printf("The thing that pointer 'ip' contains inside is %p \n",ip);
     printf("The thing that pointer 'ip' points is %d \n",*ip);
     printf("The address of variable 'x' is %p \n",&x);
     printf("The value of variable 'x' is %d \n",x);
     y = *ip; /* y has the value of x now */
     *ip = 0; /* x is 0 now */
     ip = &z[0]; /* ip now points to z[0] */
     return 0;
}
```

  - Analyze the code.
  - Execute the code. What is the output and why?

2. C Review I

- Exercise: Modify the code in the previous slide that;
  - creates two "double" type pointers,
  - puts numbers in,
  - prints out the values inside the pointers (address information),
  - prints out the values that pointers point.
- In C, there is a strong relationship between pointers and arrays, strong enough that pointers and arrays should be discussed simultaneously. The pointer version of any code will in general be faster (why?). (code7.c)
  - Analyze the code.
  - Execute the code several times. What is the output and why? Observe the changes in the addressing scheme.

2. C Review I

- Dynamic Memory Allocation: Allocating memory at runtime.
  - Malloc; code8.c
    - Analyze the code.
    - What is the function of "**malloc**"?
  - Realloc; code9.c
    - Analyze the code.
    - What is the function of "**realloc**"?
    - What is happening by the assignment "**ip = tmp;**"?
  - For other memory related functions take a look at man pages. e.g., **man malloc**.
    - What is the difference between **malloc** and **calloc**? (We will discuss later)
    - What is **brk()**? (We will discuss later)

2. C Review I

- Sending output to a device file; you should first open a terminal window and run **w** command to see which users are logged on and which virtual terminal they are using:

```
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU  WHAT
student  pts/0    :0               09:44    0.00s  0.23s  0.01s w
```

- ```
  #include <stdio.h>
  #include <stdlib.h>
  #include <string.h>

  int main(int argc, char** argv)
  {
      char command[80];
      FILE* outs;

      if ((outs = fopen(argv[1], "w")) == NULL)
      {
          fprintf(stderr, "Can't open file, exiting.\n");
          exit(-1);
      }

      do
      {
          printf("> ");
          fgets(command, 80, stdin);
          fprintf(outs, "%s", command);
      } while (strcmp(command,"exit\n") != 0);

      fclose(outs);
      return 0;
  }
  ```
  ```

2. C Review I

- Analyse the code.
- Open an another terminal window and execute the code, e.g. "**./code /dev/pts/0**"
- Observe how the program executes.
- Try to achieve the same using **echo** command.