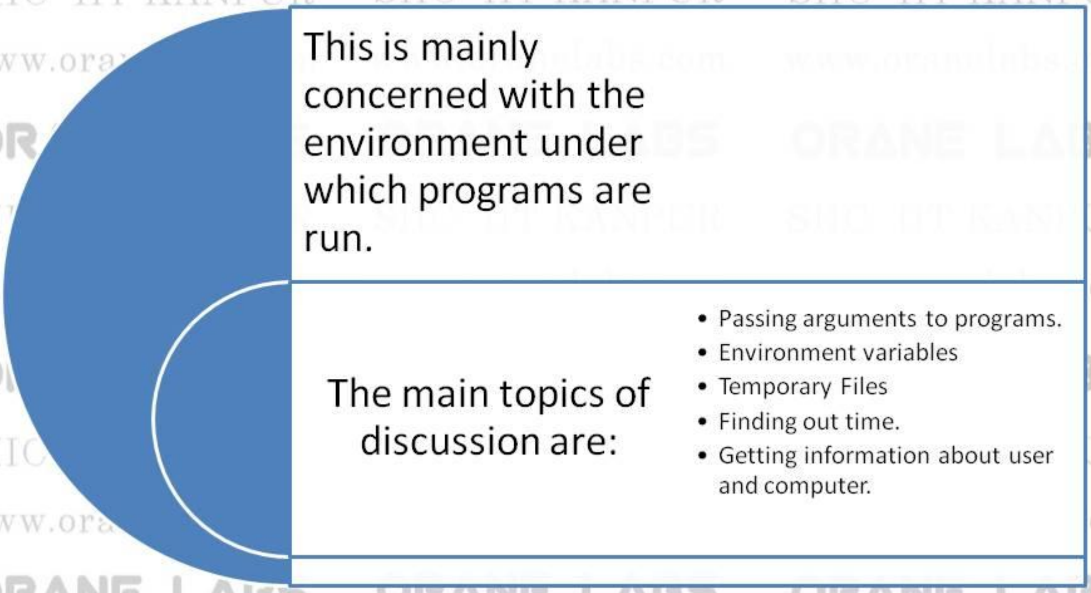
The Linux Environment

> Unix is simple and coherent, but it takes a genius (or at any rate a programmer) to understand and appreciate the simplicity.
> — Dennis Ritchie

The Environment



This is mainly concerned with the environment under which programs are run.

The main topics of discussion are:

- Passing arguments to programs.
- Environment variables
- Temporary Files
- Finding out time.
- Getting information about user and computer.

Program Arguments

- When a Linux or UNIX program written in C runs, it starts at the function main. For these programs, main is declared as
  - int main(int argc, char *argv[ ])
- where argc is a count of the program arguments and argv is an array of character strings representing the arguments themselves.
- For example, if we give the shell the following command,
  - $ myprog left right ‘and center’
  - the program myprog will start at main with parameters:
    - argc: 4
    - argv: {“myprog”, “left”, “right”, “and center”}

Example

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
int arg;
for(arg = 0; arg < argc; arg++) {
   if(argv[arg][0] == ‘-‘)
   printf(“option: %s\n”, argv[arg]+1);
   else
     printf(“argument %d: %s\n”, arg, argv[arg]);
   }
exit(0);
}
```

Output

```
$ ./args -i -lr 'hi there' -f fred.c
argument 0: ./args
option: i
option: lr
argument 3: hi there
option: f
argument 5: fred.c
```

getopt

- Linux provides the getopt facility, which supports the use of options with and without values and is simple to use.
  - **#include <unistd.h>**
  - **int getopt(int argc, char *const argv[ ], const char *optstring);**
  - **extern char *optarg;**
  - **extern int optind, opterr, optopt;**

getopt() parameters

- *argc*: number of arguments passed
- *argv*: The parameters passed
- The *optstringis* simply a list of characters, each representing a single character option. If a character is followed by a colon, it indicates that the option has an associated value that will be taken as the next argument.
- For example, the following call would be used to handle our preceding example
  - getopt(argc, argv, “if:lr”);

Return results

- The return result for getopt is the next option character found in the argv array (if there is one). Call getopt repeatedly to get each option in turn. It has the following behavior:
  - If the option takes a value, that value is pointed to by the external variable optarg.
  - getopt returns -1 when there are no more options to process. A special argument, --, will cause getopt to stop scanning for options.
  - getopt returns ? if there is an unrecognized option, which it stores in the external variable optopt.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
int opt;
while((opt = getopt(argc, argv,
    “:if:lr”)) != -1) {
switch(opt) {
case ‘i’:
case ‘l’:
case ‘r’:
printf(“option: %c\n”, opt);
break;
case ‘f’:
printf(“filename: %s\n”, optarg);
break;
case ‘:’:
printf(“option needs a value\n”);
break;
case ‘?’:
printf(“unknown option: %c\n”,
    optopt);
break;
}
}
for(; optind < argc; optind++)
printf(“argument: %s\n”,
    argv[optind]);
exit(0);
}
```

Environment Variables

- These are variables that can be used to control the behaviour of shell scripts and other programs.
- You can also use them to configure the user’s environment.
- you can examine environmen

  the shell prompt:

  $ echo $HOME

  /home/Ankur

Using C Program

- A C program may gain access to environment variables using the putenv and getenv functions.
  - #include <stdlib.h>
  - char *getenv(const char *name)
  - int putenv(const char *string);

Trying putenv and getenv

- The first few lines after the declaration of main ensure that the program, has been called correctly with just one or two arguments:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[])
{
char *var, *value;
if(argc == 1 || argc > 3) {
fprintf(stderr,”usage: environ var [value]\n”);
exit(1);
}
```

Step 2

- fetching the value of the variable from the environment, using getenv:

```
var = argv[1];
value = getenv(var);
if(value)
printf(“Variable %s has value %s\n”, var, value);
else
printf(“Variable %s has no value\n”, var);
```

Step 3

- Next, check whether the program was called with a second argument. If it was, you set the variable to the value of that argument by constructing a string of the form name=value and then calling putenv:

```
if(argc == 3) {
char *string;
value = argv[2];
string = malloc(strlen(var)+strlen(value)+2);
if(!string) {
fprintf(stderr,”out of memory\n”);
exit(1);
}
strcpy(string,var);
strcat(string,”=”);
strcat(string,value);
printf(“Calling putenv with: %s\n”,string);
if(putenv(string) != 0) {
fprintf(stderr,”putenv failed\n”);
free(string);
exit(1);
```

Step 4

- Finally, getting the new value of the variable by calling getenv once again:

```
value = getenv(var);
if(value)
printf(“New value of %s is %s\n”, var, value);
else
printf(“New value of %s is null??\n”, var);
}
exit(0);
}
```

Sample Run

```
$ ./environ HOME
Variable HOME has value /home/Ankur
$ ./environ Ankur
Variable Ankur has no value
$ ./environ Ankur hello
Variable Ankur has no value
Calling putenv with: Ankur=hello
New value of Ankur is hello
$ ./environ Ankur
Variable Ankur has no value
```

The environ Variable

-

- This array of strings is made available to programs directly via the environ variable, which is declared as
  - **#include <stdlib.h>**
  - **extern char **environ;**

Example

```
#include <stdlib.h>
#include <stdio.h>
extern char **environ;
int main()
{
char **env = environ;
while(*env) {
printf("%s\n",*env);
env++;
}
exit(0);
}
```

Time and Date

- Times are handled using a defined type, a time_t. This is an integer type intended to be large enough to contain dates and times in seconds.

```
#include <time.h>
time_t time(time_t *tloc);
```

Example

```
#include <time.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main()
{
int i;
time_t the_time;
for(i = 1; i <= 10; i++) {
the_time = time((time_t *)0);
printf(“The time is %ld\n”, the_time);
sleep(2);
}
exit(0);
}
```

Converting time into a readable format

- – **char *ctime(const time_t *timeval);**
- It takes a raw time value and converts it to a more readable local time.

Example

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
int main()
{
time_t timeval;
(void)time(&timeval);
printf(“The date is: %s”, ctime(&timeval));
exit(0);
}
```

Temporary Files

- Often, programs will need to make use of temporary storage in the form of files.
- These might hold intermediate results of a computation or represent backup copies of files made before critical operations.
- The serious issue here is that each temp file should have a new name.

A unique filename can be generated by the tmpnam function:

- Syntax:
  - #include <stdio.h>
  - char *tmpnam(char *s);
- If the temporary file is to be used immediately, you can name it and open it at the same time using the tmpfile function.
  - #include <stdio.h>
  - FILE *tmpfile(void);

Example

```
#include <stdio.h>
int main()
{
char tmpname[L_tmpnam];
char *filename;
FILE *tmpfp;
filename = tmpnam(tmpname);
printf(“Temporary  file name is: %s\n”, filename);
tmpfp = tmpfile();
if(tmpfp)
printf(“Opened a temporary file OK\n”);
else
perror(“tmpfile”);
exit(0);
}
```

User Information

- When a user logs in to a Linux system, he or she has a username and password.
- Once these have been validated, the user is presented with a shell.
- Internally, the user also has a unique user identifier known as a UID.
- Each program that Linux runs is run on behalf of a user and has an associated UID.
- Information can also be extracted from this UID

The sys/types.h header file

- The UID has its own type — uid_t— defined in sys/types.h.
- Normally, users usually have UID values larger than 100.

```
#include <sys/types.h>
#include <unistd.h>
uid_t getuid(void);
char *getlogin(void);
```

The Password File

- All information pertaining to a logged in user, can be extracted from this file, barring the password.

```
#include <pwd.h>
struct passwd *getpwuid(uid_t uid);
struct passwd *getpwnam(const char *name);
```

The password database structure, passwd

passwd Member	Description
char *pw_name	The user's login name
uid_t pw_uid The	UID number
gid_t pw_gid	The GID number
char *pw_dir	The user's home directory
char *pw_gecos	The user's full name
char *pw_shell	The user's default Shell

Example

```
#include <sys/types.h>
#include <pwd.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main()
{
uid_t uid;
gid_t gid;
struct passwd *pw;
uid = getuid();
gid = getgid();
printf("User is %s\n",
    getlogin());
printf("User IDs: uid=%d,
    gid=%d\n", uid, gid);
pw = getpwuid(uid);
printf("UID passwd entry:\n
    name=%s, uid=%d, gid=%d,
    home=%s, shell=%s\n",
pw->pw_name, pw->pw_uid,
    pw->pw_gid, pw->pw_dir,
    pw->pw_shell);

pw = getpwnam("root");
printf("root passwd entry:\n");
printf("name=%s, uid=%d,
    gid=%d, home=%s,
    shell=%s\n",
pw->pw_name, pw->pw_uid,
    pw->pw_gid, pw->pw_dir,
    pw->pw_shell);
exit(0);
}
```

Host Information

- Just like UID, you can also get useful information about the host machine.
- If the system has networking components installed, you can obtain its network name very easily with the gethostname function:

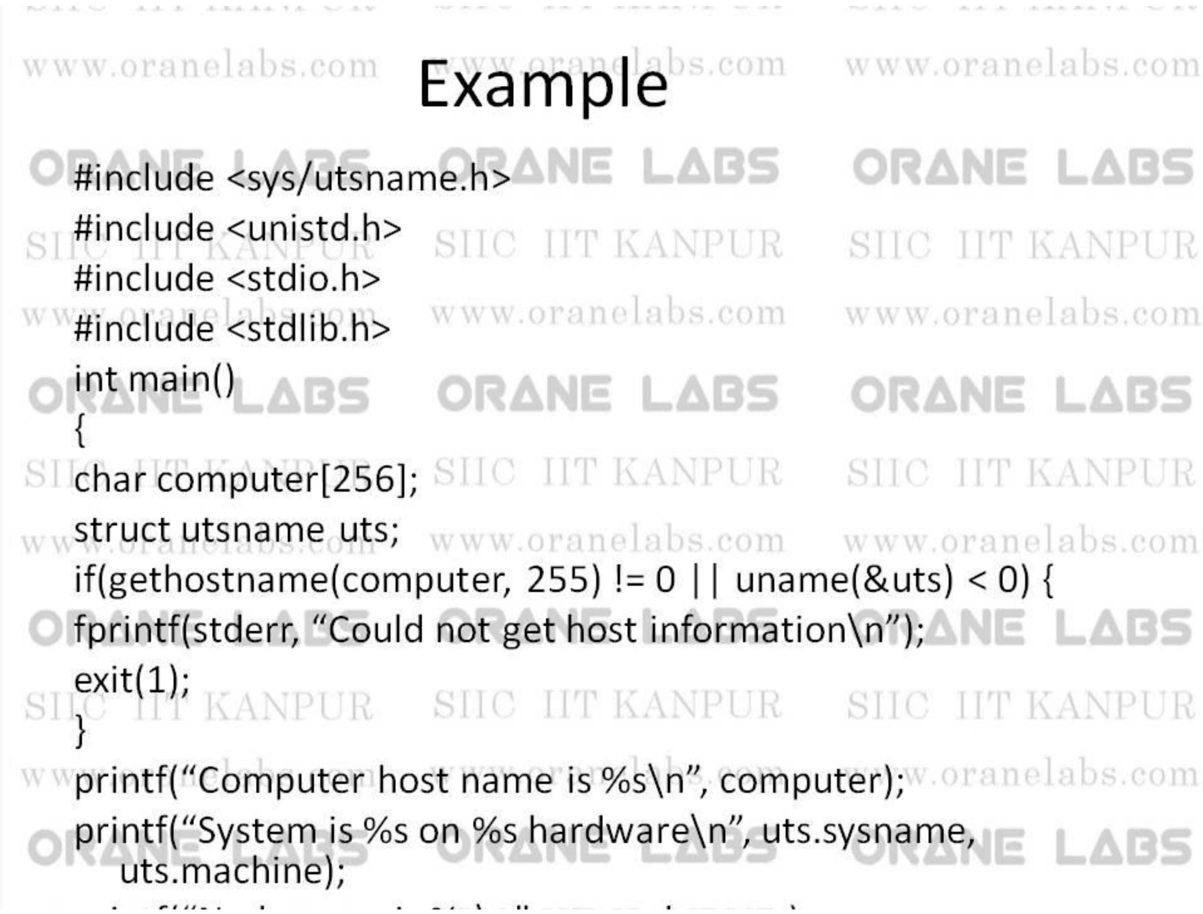
#include <unistd.h>

int gethostname(char *name, size_t namelen);

- The gethostname function writes the machine’s network name into the string name.
- gethostname returns 0 if successful and –1

- You can obtain more detailed information about the host computer from the uname system call:
  - **#include <sys/utsname.h>**

Utsname	Member Description
char sysname[ ]	The operating system name
char nodename[]	The host name
char release[]	The release level of the system
char version[]	The version number of the system
char machine[]	The hardware type



Example

```
#include <sys/utsname.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main()
{
char computer[256];
struct utsname uts;
if(gethostname(computer, 255) != 0 || uname(&uts) < 0) {
fprintf(stderr, “Could not get host information\n”);
exit(1);
}
printf(“Computer host name is %s\n”, computer);
printf(“System is %s on %s hardware\n”, uts.sysname,
   uts.machine);
```