LAB NO.: 2 **Date:**

WORKING WITH DIRECTORY STRUCTURES

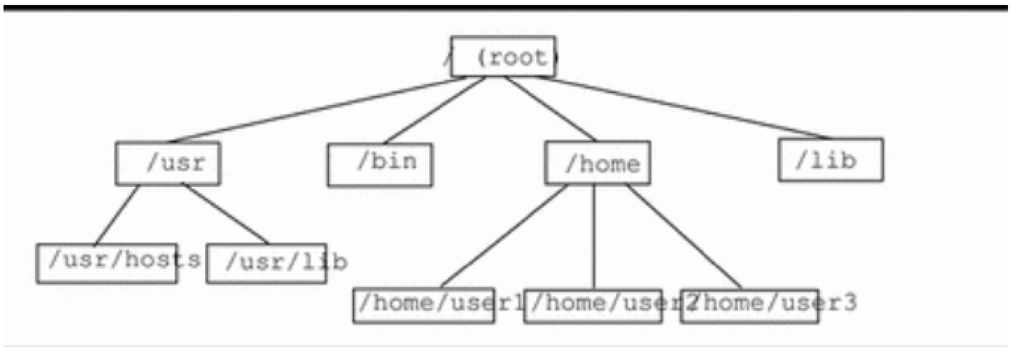
Objectives:
In this lab, the student will be able to:

1. Understand how programs can manipulate directories.
2. Work with system calls to create, scan and work with directories.

All subdirectory and file names within a directory must be unique. However, names within different directories can be the same. For example, the directory /usr contains the subdirectory /usr/lib. There is no conflict between /usr/lib and /lib because the path names are different.

Pathnames for files work exactly like path names for directories. The pathname of a file describes that file's place within the file system hierarchy. For example, if the /home/user2 directory contains a file called report5, the pathname for this file is /home/user2/report5. This shows that the file report5 is within the directory user2, which is within the directory home, which is within the root (/) directory.

Directories can contain only subdirectories, only files, or both.



Print Working Directory (◻◆♎)

The pwd command will give the present working directories

```
👓  ◻◆♎
📪♒□❍♏📪◆⬧♏❒📂
```

Your Home Directory

home
Every user has a **home** directory. When you first open the Command Tool or Shell Tool window in the OpenWindows environment, your initial location (working directory) is your home directory.

A program can determine its current working directory by calling the getcwd function.

```
#include <unistd.h>
char *getcwd(char *buf, size_t size);
```

The getcwd function writes the name of the current directory into the given buffer, buf . It returns NULL
if the directory name would exceed the size of the buffer (an ERANGE error), given as the parameter
size .
*It returns buf on success.
*getcwd may also return NULL if the directory is removed (EINVAL) or permissions changed (EACCESS) while the program is running.

mkdir and rmdir
You can create and remove directories using the mkdir and rmdir system calls.

```
#include <sys/types.h>
#include <sys/stat.h>
int mkdir(const char *path, mode_t mode);
```

The mkdir system call is used for creating directories and is the equivalent of the mkdir program. **mkdir** makes a new directory with a path as its name. The directory permissions are passed in the parameter mode and are given as in the O_CREAT option of the open system call and, again, subject to umask .

```
#include <unistd.h>
int rmdir(const char *path);
```

The rmdir system call removes directories, but only if they are empty. The rmdir program uses this
system call to do its job.

Scanning Directories

A common problem on Linux systems is scanning directories, that is, determining the files that reside in a particular directory. In shell programs, it's easy — just let the shell expand a wildcard expression. In the past, different UNIX variants have allowed programmatic access to the low-level file system structure. You can still open a directory as a regular file and directly read the directory entries, but different file system structures and implementations have made this approach nonportable. A standard suite of library functions has now been developed that makes directory scanning much simpler. The directory functions are declared in a header file dirent.h. They use a structure, DIR , as a basis for directory manipulation. A pointer to this structure, called a directory stream (a DIR *), acts in much the same way as a file stream (FILE *) does for regular file manipulation. Directory entries themselves are returned in dirent structures, also declared in dirent.h, because one should never alter the fields in the DIR structure directly.
We'll review these functions:
opendir
closedir
readdir
telldir
seekdir
closedir

opendir
The opendir function opens a directory and establishes a directory stream. If successful, it returns a pointer to a DIR structure to be used for reading directory entries.

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *name);
```

opendir returns a null pointer on failure. Note that a directory stream uses a low-level file descriptor to access the directory itself, so opendir could fail with too many open files.

readdir
The readdir function returns a pointer to a structure detailing the next directory entry in the directory stream dirp. Successive calls to readdir return further directory entries. On error, and at the end of the directory, readdir returns NULL . POSIX-compliant systems leave errno unchanged when returning NULL at end of directory and set it when an error occurs.

```
#include <sys/types.h>
#include <dirent.h>
struct dirent *readdir(DIR *dirp);
```

Note that readdir scanning isn't guaranteed to list all the files (and subdirectories) in a directory if other processes are creating and deleting files in the directory at the same time.

The dirent structure containing directory entry details includes the following entries:

❑ino_t d_ino : The inode of the file

❑char
d_name[] : The name of the file

telldir

The telldir function returns a value that records the current position in a directory stream. You can use
this in subsequent calls to seekdir to reset a directory scan to the current position.

```
#include <sys/types.h>
#include <dirent.h>
long int telldir(DIR *dirp);
```

seekdir

The seekdir function sets the directory entry pointer in the directory stream given by dirp . The value of loc , used to set the position, should have been obtained from a prior call to telldir .

```
#include <sys/types.h>
#include <dirent.h>
void seekdir(DIR *dirp, long int loc);
```

closedir

The closedir function closes a directory stream and frees up the resources associated with it. It returns 0 on success and –1 if there is an error.

```
#include <sys/types.h>
#include <dirent.h>
int closedir(DIR *dirp);
```

Sample Program

A Directory-Scanning Program

1. Start with the appropriate headers and then a function, printdir , which prints out the current directory. It will recurse for subdirectories using the depth parameter for indentation.

```
#include<unistd.h>
#include<stdio.h>
#include<dirent.h>
#include<string.h>
#include<sys/stat.h>
#include<stdlib.h>

void printdir(char *dir, int depth)
{
DIR *dp;
struct dirent *entry;
struct stat statbuf;
if((dp = opendir(dir)) == NULL) {
fprintf(stderr,"cannot open directory: %s\n", dir);
return;
}
chdir(dir);
while((entry = readdir(dp)) != NULL) {
lstat(entry->d_name,&statbuf);
if(S_ISDIR(statbuf.st_mode)) {
/* Found a directory, but ignore . and .. */
if(strcmp(".",entry->d_name) == 0 ||
strcmp("..",entry->d_name) == 0)
continue;
printf("%*s%s/\n",depth,"",entry->d_name);
/* Recurse at a new indent level */
printdir(entry->d_name,depth+4);
}
else printf("%*s%s\n",depth,"",entry->d_name);
}
chdir("..");
closedir(dp);
}
```

Lab Exercises:

1. Write a C program to emulate the ls -l UNIX command that prints all files in a current directory and lists access privileges, etc. DO NOT simply exec ls -l from the program.

2. Write a program that will list all files in a current directory and all files in subsequent subdirectories.
3. How do you list all installed programs in Linux?
4. How do you find out what RPM packages are installed on Linux?

Additional Exercises:

1. Write a program that will only list subdirectories in alphabetical order.
2. Write a program that allows the user to remove any or all of the files in a current working directory. The name of the file should appear followed by a prompt as to whether it should be removed.