BITS Pilani
Pilani Campus

Directory System Calls

Computer Science and Information Systems Department, BITS Pilani

Points to be Covered

- **Introduction**
- **Directory system calls**
  - getcwd()
  - mkdir()
  - chdir()
  - rmdir()
  - opendir()
  - closedir()
  - readdir()

Directory System Calls

- A directory is a file
  - It gives hierarchical structure to the file system
  - Play an important role in conversion of a file name to an inode number
  - Its data is a sequence of entries
  - Contents of each entries
    - an inode number and the name of a file
- Path name is a null terminated character string divided into components by slash (“/”)
- Each component must be the name of a directory and last may not be
- UNIX System V
  - Maximum of component name : 14 characters
  - Inode number entry : 2 bytes
  - Size of a directory entry : 16 bytes

Directory System Calls

- getcwd()
- mkdir()
- chdir()
- rmdir()
- opendir()
- closedir()
- readdir()

getcwd() System Call

DESCRIPTION

- The **getcwd**() function copies an absolute pathname of the current working directory to the array pointed to by *buf*, which is of length *size*.

RETURN VALUE

- **-1** on failure (for example, if the current directory is not readable), with *errno* set accordingly, and the number of characters stored in *buf* on success.
- The contents of the array pointed to by *buf* is undefined on error.

ERRORS

Tag	Description
ENOMEM	if user memory cannot be mapped
ENOENT	if directory does not exist (i.e. it has been deleted)
ERANGE	if not enough space available for storing the path
EFAULT	if memory access violation occurs while copying

getcwd() System Call

Syntax: char *getcwd(char *buffer, size_t size);

Size: The number of characters in the buffer area.

Buffer: The name of the buffer that will be used to hold the path name of the working directory. buffer must be big enough to hold the working directory name, plus a terminating NULL to mark the end of the name.

Returned value

- If successful, getcwd() returns a pointer to the buffer.
- If unsuccessful, getcwd() returns a NULL pointer and sets errno to one of the above mentioned values.

Program that uses the getcwd()

```
#include <stdlib.h>
#include <unistd.h>
#include < errno.h >

int main(void) {
        long max;
        char *buf;
        max= pathconf("/",_PC_PATH_MAX);
        buf=(char*)malloc(max);
        if (getcwd(buf,max)==NULL)
            { printf(Error: Could not obtain current working directory.\n"); }
        else
            { printf("Path: current working directory is: %s\n", buf); }
        return 0;
 }
```

getcwd() System Call

DESCRIPTION

- **chdir**() changes the current working directory to that specified in *path*.

RETURN VALUE

- On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

Error Code	Description
EACCES	Search permission is denied for one of the directories in the path prefix of *path*.
EFAULT	*path* points outside your accessible address space.
EIO	An I/O error occurred.
ELOOP	Too many symbolic links were encountered in resolving *path*.
ENAMETOOLONG	*path* is too long.
ENOENT	The file does not exist.
ENOMEM	Insufficient kernel memory was available.
ENOTDIR	A component of *path* is not a directory.

chdir() System Call

Syntax: int chdir(const char *pathname);

Returned value

- If successful, chdir() changes the working directory and returns 0.
- If unsuccessful, chdir() does not change the working directory, returns -1, and sets errno to one of the following values:

Program for chdir()

```
#include <unistd.h>
#include <stdio.h>

main() {
char cwd[256];

if (chdir("/tmp") != 0)
        printf("chdir() error()");

else {
        if (getcwd(cwd, sizeof(cwd)) == NULL)
                printf("getcwd() error");
 else
         printf("current working directory is: %s\n", cwd);
      }
}
```

mkdir() System Call

DESCRIPTION: Creates a new, empty directory, pathname.

Syntax: int mkdir(const char *pathname, mode_t mode);

- The parameter *mode* specifies the permissions to use. It is modified by the process's *umask* in the usual way: the permissions of the created directory are (*mode* & ~*umask* & 0777).
- umask() sets the calling process's file mode creation mask (umask) to *mask* & 0777.

Returned value

- If successful, mkdir() returns 0.
- If unsuccessful, mkdir() does not create a directory, returns -1, and sets errno to one of the following values:

mkdir() System Call

Tag	Description
EACCES	The parent directory does not allow write permission to the process, or one of the directories in *pathname* did not allow search permission. (See also **path_resolution**(2).)
EEXIST	*pathname* already exists (not necessarily as a directory). This includes the case where *pathname* is a symbolic link, dangling or not.
EFAULT	*pathname* points outside your accessible address space.
ELOOP	Too many symbolic links were encountered in resolving *pathname*.
ENAMETOOLONG	*pathname* was too long.
ENOENT	A directory component in *pathname* does not exist or is a dangling symbolic link.
ENOMEM	Insufficient kernel memory was available.
ENOSPC	The device containing *pathname* has no room for the new directory.
ENOSPC	The new directory cannot be created because the user's disk quota is exhausted.
ENOTDIR	A component used as a directory in *pathname* is not, in fact, a directory.
EPERM	The filesystem containing *pathname* does not support the creation of directories.
EROFS	*pathname* refers to a file on a read-only filesystem.

rmdir() System Call

DESCRIPTION: Removes a directory, pathname, provided that the directory is empty.

NOTE: pathname must not end in . (dot) or .. (dot-dot).

- If no process currently has the directory open, rmdir() deletes the directory itself. The space occupied by the directory is freed for new use.
- If one or more processes have the directory open when it is removed, the directory itself is not removed until the last process closes the directory.

Syntax: int rmdir(const char *pathname);

Returned value

- If successful, rmdir() returns 0.
- If unsuccessful, rmdir() returns -1 and sets errno to one of the following values:

rmdir() System Call

Tag	Description
EACCES	Write access to the directory containing *pathname* was not allowed, or one of the directories in the path prefix of *pathname* did not allow search permission. (See also **path_resolution**(2).
EBUSY	*pathname* is currently in use by the system or some process that prevents its removal. On Linux this means *pathname* is currently used as a mount point or is the root directory of the calling process.
EFAULT	*pathname* points outside your accessible address space.
EINVAL	*pathname* has . as last component.
ELOOP	Too many symbolic links were encountered in resolving *pathname*.
ENAMETOOLONG	*pathname* was too long.
ENOENT	A directory component in *pathname* does not exist or is a dangling symbolic link.
ENOMEM	Insufficient kernel memory was available.
ENOTDIR	*pathname*, or a component used as a directory in *pathname*, is not, in fact, a directory.
ENOTEMPTY	*pathname* contains entries other than . and .. ; or, *pathname* has .. as its final component.
EPERM	The directory containing *pathname* has the sticky bit (**S_ISVTX**) set and the process's effective user ID is neither the user ID of the file to be deleted nor that of the directory containing it, and the process is not privileged (Linux: does not have the **CAP_FOWNER** capability).
EPERM	The filesystem containing *pathname* does not support the removal of directories.
EROFS	*pathname* refers to a file on a read-only filesystem.

Make and remove dir System Call

```
# include<sys/stat.h>
int mkdir(
      const char *path,    /* Pathname */
       mode_t  perms      /* Permissions */
);
/* Returns 0 on success and -1 on error */

int rmdir(
      const char *path  /* Pathname */
);
/* Returns 0 on success and -1 on error */
```

Program for mkdir and rmdir

```
int main (int argc, char *argv[])
{
int ret;
ret = mkdir("mydir", 0777);
ret = rmdir("mydir");
return 0;
}
```

Program to make and remove directory


```
int main(int argc,char *argv[])
{  int md, rd;
   //DIR *ds;
   struct dirent *dir;
   md = mkdir(argv[1],0777);
   if(md == 0)
        printf("%s directory is created\n",argv[1]);
   else
        printf("%s directory is not created\n",argv[1]);
   rd = rmdir(argv[2]);
   if(rd == 0)
        printf("%s directory is removed\n",argv[2]);
   else
        printf("%s directory is not removed\n",argv[2]);
}
```

opendir and closedir System Calls

```
# include<dirent.h>
DIR* opendir(
            const char  *path          /* directory pathname */
);
/* Returns a DIR pointer or NULL on error */

# include<dirent.h>
int closedir(
        DIR *dirp              /*DIR pointer from opendir */
);
/* Returns a 0 on success or -1 on error */
```

readdir System Call and dirent structure


```
#include <dirent.h>
struct dirent *readdir(
            DIR *dirp /* DIR Pointer from opendir */
);
/* Returns pointer to the structure or NULL on EOF or error */

struct dirent {
    ino_t  d_ino;  /*  i-number */
    char d_name[];   /*  name */
};
```

```
int main(int argc,char *argv[])
{
    DIR *ds; struct dirent *dir;
    ds = opendir(argv[1]);
    if(ds == NULL)
        printf("directory %s is not opened\n",argv[1]);
    else
            printf(“Directory opened/n”);
    printf("list of files and directories\n");
    while((dir = readdir(ds)) != NULL)
            printf("%s\n",dir->d_name);
    if((cld = closedir(ds)) == 0)
            printf("%s is successfully closed\n",argv[1]);
    else
        printf("%s is not successfully closed\n",argv[1]);
}
```

```
[CSIS@localhost dirProgs]$ ./a.out
/home/CSIS/Downloads/

Directory opened

list of files and directories
..
.
SimplescalarHowto.pdf
Installation_Guide_for_SimpleScalar.pdf
p130-spradling.pdf
/home/CSIS/Downloads/ is successfully closed
[CSIS@localhost dirProgs]$
```

Exercise program

- Write a program to mimic ls –l command in unix.
- Write a program to display number of files in current working directory.
- Write a program which performs rm –r unix command.

Programming Exercises

- Write a program to display the files in current working directory.

Solution

```
#include <stdio.h>
#include <dirent.h>

int main(void)
{
    struct dirent *de;  // Pointer for directory entry

    // opendir() returns a pointer of DIR type.
    DIR *dr = opendir(".");

    if (dr == NULL)  // opendir returns NULL if couldn't open directory
    {
        printf("Could not open current directory" );
        return 0;
    }

    while ((de = readdir(dr)) != NULL)
            printf("%s\n", de->d_name);

    closedir(dr);
    return 0;
}
```

Programming Exercise

- Write a program to mimic mv command in unix using system call programming.

Solution

```
#include<stdio.h>
# include<fcntl.h> int main(int argc,char *argv[])
{
int fd1,fd2,count; char buf[512]; if(argc!=3){
printf("give sufficient filenames"); exit(1);}
else{
fd1=open(argv[1],O_RDONLY); if(fd1==-1)
{ printf("source file does not exist"); exit(1);
}
fd2=open(argv[2],O_WRONLY); if(fd2==-1)
fd2=creat(argv[2],0666);
while((count=read(fd1,buf,512))>0)
{
write(fd2,buf,count);
}
close(fd2);
close(fd1);
}
int c=unlink(argv[1]); if(c==0)
printf("unlinked successfully"); else
printf("link error");
}
```

Programming Exercise

Write a C program to mimic cp command in Unix.

Solution

```
#define BUFF_SIZE 256
int main(int argc, char* argv[])
{
int srcFD,destFD,nbread,nbwrite;
char *buff[BUFF_SIZE];
if(argc != 3){
printf("\nplease enter the file names");
exit(EXIT_FAILURE);}
 srcFD = open(argv[1],O_RDONLY);
if(srcFD == -1){
printf("\nError in opening file);}
destFD = open(argv[2],O_WRONLY | O_CREAT , 0777);
if(destFD == -1){
printf("\nError in opening file);}
while((nbread = read(srcFD,buff,BUFF_SIZE)) > 0){
if(write(destFD,buff,nbread) != nbread)
printf("\nError in writing data to %s\n",argv[2]);}
if(nbread == -1)
printf("\nError in reading data from %s\n",argv[1]);
close(srcFD);
close(destFD);}
```

Programming Exercise

1. Write a program to make a directory, create two text files inside it, open the directory and file. Write to one of the files and then copy content of one file to another. Close the files and the directory.

Any Queries?