Subsections

- Directory handling functions: <unistd.h>
  - Scanning and Sorting Directories: <sys/types.h>,<sys/dir.h>
- File Manipulation Routines: unistd.h, sys/types.h, sys/stat.h
  - File Access
    - errno
  - File Status
  - File Manipulation:stdio.h, unistd.h
  - Creating Temporary FIles:<stdio.h>
- Exercises

File Access and Directory System Calls

There are many UNIX utilities that allow us to manipulate directories and files. `cd, ls, rm, cp, mkdir` ***etc.*** are examples we have (hopefully) already met.

We will now see how to achieve similar tasks from within a C program.

Directory handling functions: <unistd.h>

This basically involves calling appropriate functions to traverse a directory hierarchy or inquire about a directories contents.

`int chdir(char *path)` -- changes directory to specified path string.

Example: C emulation of UNIX's `cd` command:

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

main(int argc,char **argv)
   {
                                  if (argc < 2)
                                                  {  printf(``Usage: %s
                                                                    <pathnar

                                                                   exit(1);
                                                   }
                                  if (chdir(argv[1]) != 0)
                                                  { printf(``Error in chdir'

                                                                   exit(1);
                                                  }
                  }
```

`char *getwd(char *path) -- get the full pathname of the current working directory. path is a pointer to a string where the pathname will be returned. getwd returns a pointer to the string or NULL if an error occurs.`

Scanning and Sorting Directories: <sys/types.h>, <sys/dir.h>

Two useful functions (On BSD platforms and **NOT** in multi-threaded application) are available

`scandir(char *dirname, struct direct **namelist, int (*select)(), int (*compar)())` -- reads the directory dirname and builds an array of pointers to directory entries or -1 for an error. `namelist` is a pointer to an array of structure pointers.

`(*select))()` is a pointer to a function which is called with a pointer to a directory entry (defined in <sys/types> and should return a non zero value if the directory entry should be included in the array. If this pointer is NULL, then all the directory entries will be included.

The last argument is a pointer to a routine which is passed to `qsort` (see `man qsort`) -- a built in function which sorts the completed array. If this pointer is NULL, the array is not sorted.

`alphasort(struct direct **d1, **d2)` -- alphasort() is a built in routine which will sort the array alphabetically.

Example - a simple C version of UNIX `ls` utility

```
#include <sys/types.h>
#include <sys/dir.h>
#include <sys/param.h>
#include <stdio.h>

#define FALSE 0
#define TRUE !FALSE

extern  int alphasort();

char pathname[MAXPATHLEN];

main()   { int count,i;
                                struct direct **files;
                                int file_select();

                                if (getwd(pathname) == NULL )
                                                { printf("Error getting pa

                                                                exit(0);
                                                }
                                printf("Current Working Directory = %s\n"

                                count =
                                  scandir(pathname, &files, file_select, a

                                /* If no files found, make a non-selectabl
                                if              (count <= 0)
                                                {                printf(``

                                                                exit(0);
                                                }
                                printf(``Number of files = %d\n'',count);

                                for (i=1;i<count+1;++i)
                                                 printf(``%s  '',files[i-1
                                printf(``\n''); /* flush buffer */

                        }
```

```
int file_select(struct direct   *entry)

                {if ((strcmp(entry->d_name, ``.'') == 0) ||
                                          (strcmp(entry->d_name, ``.
                                           return (FALSE);
                                 else
                                                                  return (TR
                }
```

scandir returns the current directory (.) and the directory above this (..) as well as all files so we need to check for these and return FALSE so that they are not included in our list.

Note: scandir and alphasort have definitions in sys/types.h and sys/dir.h.
MAXPATHLEN and getwd definitions in sys/param.h

We can go further than this and search for specific files: Let's write a modified
file_select() that only scans for files with a .c, .o or .h suffix:

```
int file_select(struct direct   *entry)

                {char *ptr;
                                  char *rindex(char *s, char c);

                                 if ((strcmp(entry->d_name, ``.'')== 0) ||
                                                (strcmp(entry->d_name, ``.
                                                 return (FALSE);

                                  /* Check for filename extensions */
                                  ptr = rindex(entry->d_name, '.')
                                  if ((ptr != NULL) &&
                                                  ((strcmp(ptr, ``.c'') ==
                                                  || (strcmp(ptr, ``.h'') =

                                                  || (strcmp(ptr, ``.o'') =

                                                                  return (T
                                  else
                                                  return(FALSE);
                }
```

NOTE: rindex() is a string handling function that returns a pointer to the last occurrence of character c in string s, or a NULL pointer if c does not occur in the string. (index() is similar function but assigns a pointer to 1st occurrence.)

The function struct direct *readdir(char *dir) also exists in <sys/dir.h>> to return a given directory dir listing.

File Manipulation Routines: unistd.h, sys/types.h, sys/stat.h

There are many system calls that can applied directly to files stored in a directory.

File Access

`int access(char *path, int mode)` -- determine accessibility of file.

`path` points to a path name naming a file. `access()` checks the named file for accessibility according to `mode`, defined in `#include <unistd.h>`:

R_OK
- test for read permission

W_OK
- test for write permission

X_OK
- test for execute or search permission

F_OK
- test whether the directories leading to the file can be searched and the file exists.

`access()` returns: 0 on success, -1 on failure and sets `errno` to indicate the error. See `man` pages for list of errors.

errno

`errno` is a special system variable that is set if a system call cannot perform its set task.

To use `errno` in a C program it must be declared via:

```
extern int errno;
```

It can be manually reset within a C program other wise it simply retains its last value.

`int chmod(char *path, int mode)` change the mode of access of a file. specified by `path` to the given `mode`.

`chmod()` returns 0 on success, -1 on failure and sets `errno` to indicate the error. Errors are defined in `#include <sys/stat.h>`

The access mode of a file can be set using predefined macros in `sys/stat.h` -- see `man` pages -- or by setting the mode in a a 3 digit octal number.

The rightmost digit specifies owner privileges, middle group privileges and the leftmost other users privileges.

For each octal digit think of it a 3 bit binary number. Leftmost bit = read access (on/off) middle is write, right is executable.

So 4 (octal 100) = read only, 2 (010) = write, 6 (110) = read and write, 1 (001) = execute.

so for access mode 600 gives user read and write access others no access. 666 gives everybody read/write access.

NOTE: a UNIX command `chmod` also exists

File Status

Two useful functions exist to inquire about the files current status. You can find out how large the file is (`st_size`) when it was created (`st_ctime`) ***etc.*** (see `stat` structure definition below. The two functions are prototyped in `<sys/stat.h>`

```
int stat(char *path, struct stat *buf),
int fstat(int fd, struct
```

```
stat *buf)
```

`stat()` obtains information about the file named by path. Read, write or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable.

`fstat()` obtains the same information about an open file referenced by the argument descriptor, such as would be obtained by an `open` call (Low level I/O).

`stat(), and fstat()` return 0 on success, -1 on failure and sets `errno` to indicate the error. Errors are again defined in `#include <sys/stat.h>`

`buf` is a pointer to a stat structure into which information is placed concerning the file. A stat structure is define in `#include <sys/types.h>`, as follows

```
struct stat {
          mode_t   st_mode;     /* File mode (type, perms) */
          ino_t    st_ino;      /* Inode number */
          dev_t    st_dev;      /* ID of device containing */
                                /* a directory entry for this file */
          dev_t    st_rdev;     /* ID of device */
                                /* This entry is defined only for */
                                /* char special or block special files */
          nlink_t  st_nlink;    /* Number of links */
          uid_t    st_uid;      /* User ID of the file's owner */
          gid_t    st_gid;      /* Group ID of the file's group */
          off_t    st_size;     /* File size in bytes */
          time_t   st_atime;    /* Time of last access */
          time_t   st_mtime;    /* Time of last data modification */
          time_t   st_ctime;    /* Time of last file status change */
                                /* Times measured in seconds since */
                                /* 00:00:00 UTC, Jan. 1, 1970 */
          long     st_blksize;  /* Preferred I/O block size */
          blkcnt_t st_blocks;   /* Number of 512 byte blocks allocated*/
}
```

File Manipulation:stdio.h, unistd.h

There are few functions that exist to delete and rename files. Probably the most common way is to use the `stdio.h` functions:

```
int remove(const char *path);
int rename(const char *old, const char *new);
```

Two system calls (defined in `unistd.h`) which are actually used by `remove()` and `rename()` also exist but are probably harder to remember unless you are familiar with UNIX.

`int unlink(cons char *path)` -- removes the directory entry named by `path`

`unlink()` returns 0 on success, -1 on failure and sets `errno` to indicate the error. Errors listed in `#include <sys/stat.h>`

A similar function `link(const char *path1, const char *path2)` creates a linking from an existing directory entry `path1` to a new entry `path2`

Creating Temporary FIles:<stdio.h>

Programs often need to create files just for the life of the program. Two convenient functions (plus some variants) exist to assist in this task. Management (deletion of files etc) is taken care of by the Operating System.

The function `FILE *tmpfile(void)` creates a temporary file and opens a corresponding stream. The file will automatically be deleted when all references to the file are closed.

The function `char *tmpnam(char *s)` generate file names that can safely be used for a temporary file. Variant functions `char *tmpnam_r(char *s)` and `char *tempnam(const char *dir, const char *pfx)` also exist

NOTE: There are a few more file manipulation routines not listed here see `man` pages.

Exercises

Exercise 12675

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.

Exercise 12676

Write a program to print the lines of a file which contain a word given as the program argument (a simple version of `grep` UNIX utility).

Exercise 12677

Write a program to list the files given as arguments, stopping every 20 lines until a key is hit.(a simple version of `more` UNIX utility)

Exercise 12678

Write a program that will list all files in a current directory and all files in subsequent sub directories.

Exercise 12679

Write a program that will only list subdirectories in alphabetical order.

Exercise 12680

Write a program that shows the user all his/her C source programs and then prompts interactively as to whether others should be granted read permission; if affirmative such permission should be granted.

Exercise 12681

Write a program that gives the user the opportunity 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.

Dave Marshall
1/5/1999