C Functions for Directory Operation under Windows

by Dr. Xiaowen Chu
Department of Computer Science
Hong Kong Baptist University

Remark:
This document aims to help you implement the COMP2330 course project.

I. Directory Browsing

You can use the following three functions to browse the files/subdirectories of a directory:
_findfirst, _findnext, _findclose

Required header file
<io.h>

Before introducing the three functions, you need to understand the data structure **_finddata_t** which stores **file-attribute information** (file type, file size, file name, etc.) returned by functions _findfirst and _findnext.

Structure **_finddata_t** is defined in header file io.h:

```
---------------------------------------------------------------------------------------------------
typedef unsigned long _fsize_t;

struct _finddata_t {
      unsigned    attrib;         /* file attributes */
      time_t      time_create;    /* -1 for FAT file systems */
      time_t      time_access;    /* -1 for FAT file systems */
      time_t      time_write;
      _fsize_t    size;           /* file size, 0 for directory */
      char        name[260];      /* file name */
};

/* File attribute constants for _findfirst() */

#define _A_NORMAL      0x00    /* Normal file - No read/write restrictions */
#define _A_RDONLY      0x01    /* Read only file */
#define _A_HIDDEN      0x02    /* Hidden file */
#define _A_SYSTEM      0x04    /* System file */
#define _A_SUBDIR      0x10    /* Subdirectory */
#define _A_ARCH        0x20    /* Archive file */
---------------------------------------------------------------------------------------------------
```

Now let's study the prototypes of the three functions:

- **_findfirst**

Description: Provide information about the first instance of a filename that matches the file specified in the *filespec* argument.

```
intptr_t _findfirst(
   const char *filespec,
   struct _finddata_t *fileinfo
);
```

Parameters

filespec
: Target file specification (may include wildcards).

fileinfo
: File information buffer.

Return Value

If successful, **_findfirst** returns a unique search handle identifying the file or group of files matching the *filespec* specification, which can be used in a subsequent call to _findnext, or to **_findclose**. Otherwise, **_findfirst** will return –1 and set **errno** to one of the following values:

ENOENT
: File specification that could not be matched.

EINVAL
: Invalid filename specification.

- **_findnext**

Description: Find the next name, if any, that matches the *filespec* argument in a previous call to _findfirst, and then alter the *fileinfo* structure contents accordingly.

```
int _findnext(
   intptr_t handle,
   struct _finddata_t *fileinfo
);
```

Parameters

handle
: Search handle returned by a previous call to **_findfirst**.

fileinfo
: File information buffer.

Return Value

If successful, return 0. Otherwise, return –1 and sets **errno** to **ENOENT**, indicating that no more matching files could be found.

You must call _findclose after you are finished using either the **_findfirst** or **_findnext** function. This will free up resources used by these functions in your application.

- **_findclose**

Description: Closes the specified search handle and releases associated resources.

```
int _findclose(
   intptr_t handle
);
```

Parameter

handle
Search handle returned by a previous call to **_findfirst**.

Return Value

If successful, **_findclose** returns 0. Otherwise, it returns –1 and sets **errno** to **ENOENT**, indicating that no more matching files could be found.

Example code:

```
// Output the names and sizes of all files under the current directory

#include <stdio.h>
#include <stdlib.h>
#include <io.h>

int main()
{
        struct _finddata_t myfile;
        intptr_t p;

        p = _findfirst("*.*", &myfile);
        if( p != -1) {
                if (myfile.attrib & _A_SUBDIR)
                        printf("%15s\t%s\n", "<DIR>", myfile.name);
                else
                        printf("%15d\t%s\n", myfile.size, myfile.name);
                while( _findnext(p, &myfile) != -1 ) {
                        if (myfile.attrib & _A_SUBDIR)
```

```
                printf("%15s\t%s\n", "<DIR>", myfile.name);
            else
                printf("%15d\t%s\n", myfile.size, myfile.name);
        }
        _findclose(p);
    }

    return 0;
}
```

II. Directory-Control Routines

A set of directory control routines are provided to access, modify, and obtain information about the directory structure.

Required header
<direct.h>

- **_mkdir**

Description: Create a new directory.

```
int _mkdir(
   const char *dirname
);
```

Parameter

dirname
Path for new directory.

Return Value

Each of these functions returns the value 0 if the new directory was created. On an error the function returns –1 and sets **errno** as follows:

EEXIST
Directory was not created because *dirname* is the name of an existing file, directory, or device.
ENOENT
Path was not found.

- **_rmdir**

Description: Delete a directory.

```
int _rmdir(
   const char *dirname
);
```

Parameters

dirname
Path of directory to be removed.

Return Value

Each of these functions returns 0 if the directory is successfully deleted. A return value of –1 indicates an error, and **errno** is set to one of the following values:

ENOTEMPTY
Given path is not a directory; directory is not empty; or directory is either current working directory or root directory.
ENOENT
Path is invalid.
EACCESS
A program has an open handle to the directory.

Example Code (from [1])

```
// crt_makedir.c

#include <direct.h>
#include <stdlib.h>
#include <stdio.h>

int main( void )
{
  if( _mkdir( "\\testtmp" ) == 0 )
  {
    printf( "Directory '\\testtmp' was successfully created\n" );
    system( "dir \\testtmp" );
    if( _rmdir( "\\testtmp" ) == 0 )
      printf( "Directory '\\testtmp' was successfully removed\n"  );
    else
      printf( "Problem removing directory '\\testtmp'\n" );
  }
  else
```

```
    printf( "Problem creating directory '\\testtmp'\n" );
}
```

Sample Output

```
Directory '\testtmp' was successfully created
 Volume in drive C has no label.
 Volume Serial Number is E078-087A

 Directory of C:\testtmp

02/12/2002  09:56a      <DIR>          .
02/12/2002  09:56a      <DIR>          ..
               0 File(s)              0 bytes
               2 Dir(s)  15,498,690,560 bytes free
Directory '\testtmp' was successfully removed
```

- **_chdir**

Description: Change the current working directory.

```
int _chdir(
   const char *dirname
);
```

Parameter

dirname
Path of new working directory.

Return Value

These functions return a value of 0 if successful. A return value of –1 indicates that the specified path could not be found, in which case **errno** is set to **ENOENT**.

Remarks

The **_chdir** function changes the current working directory to the directory specified by *dirname*. The *dirname* parameter must refer to an existing directory. This function can change the current working directory on any drive. If a new drive letter is specified in *dirname*, the default drive letter will be changed as well. For example, if A is the default drive letter and \BIN is the current working directory, the following call changes the current working directory for drive C and establishes C as the new default drive:

```
_chdir("c:\\temp");
```

When you use the optional backslash character (\) in paths, you must place two backslashes (\\) in a C string literal to represent a single backslash (\).

Example Code (from [1])

```
// crt_chdir.c

/* This program uses the _chdir function to verify
 * that a given directory exists.
 */

#include <direct.h>
#include <stdio.h>
#include <stdlib.h>

int main( int argc, char *argv[] )
{
  if( _chdir( argv[1] )   )
    printf( "Unable to locate the directory: %s\n", argv[1] );
  else
    system( "dir *.exe");
}
```

Sample Output

```
 Volume in drive C is CDRIVE
 Volume Serial Number is 0E17-1702

 Directory of C:\write

04/21/95  01:06p             3,200 ERRATA.WRI
04/21/95  01:06p             2,816 README.WRI
           2 File(s)         6,016 bytes
                        71,432,116 bytes free
```

- **_getcwd**

Description: Get the current working directory.

```
char *_getcwd(
   char *buffer,
   int maxlen
);
```

Parameters

buffer
Storage location for path.
maxlen
Maximum length of path in characters: **char** for **_getcwd** and **wchar_t** for **_wgetcwd**.

Return Value

Returns a pointer to *buffer*. A **NULL** return value indicates an error, and **errno** is set either to **ENOMEM**, indicating that there is insufficient memory to allocate *maxlen* bytes (when a **NULL** argument is given as *buffer*), or to **ERANGE**, indicating that the path is longer than *maxlen* characters.

Remarks

The **_getcwd** function gets the full path of the current working directory for the default drive and stores it at *buffer*. The integer argument *maxlen* specifies the maximum length for the path. An error occurs if the length of the path (including the terminating null character) exceeds *maxlen.* The *buffer* argument can be **NULL**; a buffer of at least size *maxlen* (more only if necessary) will automatically be allocated, using **malloc**, to store the path. This buffer can later be freed by calling **free** and passing it the **_getcwd** return value (a pointer to the allocated buffer).

_getcwd returns a string that represents the path of the current working directory. If the current working directory is the root, the string ends with a backslash (\ ). If the current working directory is a directory other than the root, the string ends with the directory name and not with a backslash.

Example Code (from [1])

```
// crt_getcwd.c
/* This program places the name of the current directory in the
 * buffer array, then displays the name of the current directory
 * on the screen. Specifying a length of _MAX_PATH leaves room
 * for the longest legal path name.
 */

#include <direct.h>
#include <stdlib.h>
#include <stdio.h>

int main( void )
{
  char buffer[_MAX_PATH];
```

```
   /* Get the current working directory: */
   if( _getcwd( buffer, _MAX_PATH ) == NULL )
      perror( "_getcwd error" );
   else
      printf( "%s\n", buffer );
}
```

References:

[1]: Run-Time Library Reference, Microsoft MSDN