Project 3: Tar (tape archive) File Manipulation

Learning Objectives

Upon completion of this assignment, you should be able to:

1. Read and understand directory entries and file attributes.
2. Perform complex file input/output and file manipulation operations.

The mechanisms you will practice include:

- Buffered I/O: `fopen()`, `fclose()`, `fread()`, `fwrite()`, `fseek()`, `feof()`
- Reading directory entries: `opendir()`, `readdir()`, `closedir()`
- File metadata: `stat()`/`lstat()`, `chmod()`, `utimes()`, `gettimeofday()`
- Making hard links and directories: `link()`, `mkdir()`

Program Specifications

NAME

`mytar` – create and manipulate tape archives

SYNOPSIS

```
mytar [cxtf:] [file] filename
```

DESCRIPTION

`mytar` creates an archive from a directory tree, extracts files from an archive, or prints the contents and details of an archive file.

The options[1] are as follows:

`-c`
: creates an archive of the specified directory tree

`-x`
: extract the directory tree contained in the specified archive

`-t`
: print the contents of the specified archive

`-f:`
: the subsequent argument is the name of the archive file (to create, extract or print)

EXIT STATUS

`mytar` exits with 0 on success and -1 on failure.

[1] Options followed by a ':' expect a subsequence parameter

ERRORS

Upon error, `mytar` exits after printing to the standard error stream an appropriate message using one of the format strings below:

"Error: No tarfile specified.\n"

"Error: Multiple modes specified.\n"

"Error: No mode specified.\n"

"Error: Specified target(\"%s\" does not exist.\n)"

"Error: Specified target(\"%s\" is not a directory.\n)"

"Error: No directory target specified.\n"

"Error: Bad magic number (%d), should be: %d.\n"

If a library/system call fails, `mytar` calls `perror()` with the name of the failed routine, then exits. Example: `perror(''fread'')`

Note: Do not cut/paste the error messages from this pdf, and the characters are not be encoded in ASCII and cause problems with the program.

EXAMPLES

`mytar -c -f a.tar a`
: create an archive, `a.tar`, containing all files in the directory tree, `a`

`mytar -x -f a.tar`
: extract the files in the archive, `a.tar`

`mytar -t -f a.tar`
: print the details of all files in the archive, `a.tar`

`mytar -c a`
: Error: No tarfile specified.

Implementation Details

mytar File Format

`mytar` archives can be read or printed by any other program that observes the proper format[2]. The `mytar` format specification follows: first, a magic number that identifies mytar archives. Then for each file in the archive, the archive contains in order: inode number, filename length, filename, mode, file modification time, and for regular files, file size and file content.

[2]Beware of endianness!

	Size	Content	Notes
Magic Number	4 bytes	0x7261746D	Once per archive
Regular Files	8 bytes	file inode number	
	4 bytes	file name length	
	n bytes	file name	n is file name length
	4 bytes	file mode	
	8 bytes	file modification time	in seconds
	8 bytes	file size	
	n bytes	file content	n is file size
Directories	8 bytes	file inode number	
	4 bytes	file name length	
	n bytes	file name	n is file name length
	4 bytes	file mode	
	8 bytes	file modification time	in seconds
Hard Links	8 bytes	file inode number	
	4 bytes	file name length	
	n bytes	file name	n is file name length

Printing

In print (a.k.a. "`-t`" or "test") mode, `mytar` reads an archive and prints information for each file using the following formatted statements:

Directories: "%s/ inode: %llu, mode: %o, mtime: %llu\n"

Regular Files: "%s inode: %llu, mode: %o, mtime: %llu, size: %llu\n"

Executable Files: "%s* inode: %llu, mode: %o, mtime: %llu, size: %llu\n"

Hard links: "%s/ inode: %llu\n"

In each case, the initial "%s" is the file name, including its relative path.

Hard links, symbolic links, and special directories

`mytar` ignores symbolic (soft) links, "." and "..". `mytar` tracks inodes to exclude redundant information for hard links that reference the same inode. For redundant hard links, `mytar` only archives the filename (and length) and inode number.

File modes and modification times

Reestablishing file modes, using `chmod()`, is straightforward.

`mytar` archives file modification times (`st_mode` from `struct stat`) in seconds. Upon extraction, `mytar` uses the `utimes()` system call to reestablish file modification times. `utimes()` requires an array of two `struct timevals`, one for access time and one for modification time. A `struct timeval` has two fields: `tv_sec` (seconds) and `tv_usec` (microseconds).

For the two `struct timevals` required by `utimes()`, set the first (access time) to the current time using `gettimeofday()`. For the second (modification time), set `tv_sec` to the file modification time retrieved from the archive and `tv_usec` to 0.

Don't forget, directory files have modes too. However, do not worry about directory modification times.

Requirements and Constraints

1. In the archive, directories should appear before any contained files or directories.
2. Your archive must specify file and directory paths relative to the specified directory: you should not use absolute paths
3. Close files and directories immediately when you are finished reading or writing them.
4. Contents should appear in the archive in the order returned by `readdir()`.
5. Traverse and tar directories in a depth-first manner.
6. Check every system and library call for failure!

Hints and Tips

1. There is no (portable) way to retain original inode values during extraction.
2. You may find the provided `inodemap.c` and `inodemap.h` helpful for tracking hard links inodes
3. `strncpy()`, `strncat()`, `strlen()` and `strcmp()` may be helpful for preprending directory paths to file names and otherwise comparing and manipulating strings.
4. Consider files of size 0.
5. `feof()` can tell you when you have reached EOF (end of file).