Tar (tape archive) File Manipulation

Learning Objectives

Upon completion of this assignment, you should be able:

1. To read and understand directory entries and file attributes;
2. To do complex file input/output and file manipulation operations.

The mechanisms you will practice using 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 Specification

NAME

`mytar` – create and manipulate tape archives

SYNOPSIS

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

DESCRIPTION

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

Options followed by a ':' expect a subsequent parameter. The options[1] are as follows:

`-c`
: create an archive of the given directory tree. A directory name must be specified.

`-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). This option must always be specified.

EXIT STATUS

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

[1] Options followed by a ':' expect a subsequent 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.

EXIT STATUS

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

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, filename length, filename, mode, modification time, and for regular files, file size and file content.

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

For directories:	`"%s/ -- inode: %llu, mode: %o, mtime: %llu\n"`
For regular files:	`"%s -- inode: %llu, mode: %o, mtime: %llu, size: %llu\n"`
For executable files:	`"%s* -- inode: %llu, mode: %o, mtime: %llu, size: %llu\n"`
For hard links:	`"%s -- inode: %llu\n"`

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

[2] If you transfer tar files between heterogeneous platforms, endianness may be an issue.

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 2 `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 2 `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.

(See man pages and examples for `chmod`(), `utimes`(), `stat`(), and `gettimeofday`().)

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 library call for failure!

Hints and Tips

1. There is no (portable) way to retain original inode values during extraction.
2. For symbolic links, `stat`() returns information about the file the link references and `lstat`() returns information about the link itself.
3. You may find the provided inodemap.c and inodemap.h helpful for tracking hard link inodes. (This was demo'd in the du9.c program in class.)
4. `strncpy`(), `strncat`(), `strlen`() and `strcmp`() may be helpful for prepending directory paths to file names and otherwise comparing and manipulating strings.
5. Consider files of size 0.
6. `feof`() can tell you have you have reached EOF (end of file).

Testing Considerations

1. You can test your print functionality by creating an archive with my implementation and testing whether your implementation and mine print the same output.

2. You can test your create functionality by creating an archive with your implementation and testing whether my implementation can properly print and extract the contents.

3. You can test your create functionality by creating an archive with my implementation and testing whether yours can properly extract the contents.

4. I used `ls -lR` and `diff -r` extensively during testing to compare the structure and contents of original and extracted directory trees.

Submission (via Mimir Classroom)

You must submit only the following files, containing your code solution:

- Sources: any and all .c and .h files needed to build the functions in `my_tar` program
- Makefile: a makefile that properly builds your `my_tar` program from your source files.

Mimir Classroom will immediately build, test and grade your submission. We recommend you submit your solution before the due date as many times as necessary to identify and fix issues. Any submissions made after the due date will consume your late days.

Intermediate testing and debugging should be done on the CS workstations, where you can compare your solution with the instructor's.