

Tar Archiver

CS3411 Spring 2024

Program Two

Due: Wednesday, February 21, midnight

In this project, we will develop two programs; a program to *tar* a given list of files and create an archive file, and another program that can *untar* a given archive file. You should call the program which combines the given list of files *ctar* and the program which extracts an archive file *utar*.

The usage information for the commands is given below:

To create an empty archive file:

```
ctar -a <archive file name>
```

To append files to a given archive (and create if not present) :

```
ctar -a <archive file name> file1 file2 ... fileN
```

To delete a file from the archive

```
ctar -d <archive file name> <file name>
```

To extract contents of the archive file :

```
utar <archive file name>
```

In order to simplify your implementation, assume that each of the file parameters should be the name of a file, and not a directory and no wild-cards will be allowed. In order to find if a given argument is a regular file or not, you can use *lstat* kernel call. Part of this assignment is for you to learn how to use a system call by reading its manual page.

The archive file should begin with a header which contains pointers to where each file name starts and the size of the file. This file name pointer is a relative offset in number of bytes from the beginning of the file to where the name is stored within the archive if a file exists. If a file does not exist in that position in the header, the file name pointer must be 0. When storing a file name, the string should be preceded by 2 bytes (a short) that contain the string's length. The string should NOT be null terminated. The file contents should begin immediately after the file name ends.

The header will provide enough space to store information about 4 files and an overflow link *next*. The link, when not zero, is a pointer to another header of identical structure.

The C structure describing the header is shown below :

```
typedef struct
{
    int  magic;           /* This must have the value 0x63746172.          */
    int  eop;             /* End of file pointer.                          */
    int  block_count;     /* Number of entries in the block which are in-use. */
    int  file_size[4];    /* File size in bytes for files 1..4              */
    char deleted[4];      /* Contains binary one at position i if i-th entry was deleted. */
    int  file_name[4];    /* pointer to the name of the file.               */
    int  next;            /* pointer to the next header block.              */
} hdr;
```

There is no maximum file size. File name length will not exceed what can be stored in a short. For the size of the header, do not assume a constant value; use `sizeof()` instead. After the header has been written to file, up to four files can be written to the archive. If there are more than 4 files to be archived, create a new header at the end of the file after archiving the first 4 files, then repeat the process until all files are processed. Note that you must create a new header after inserting 4 files even if a 5th file is not present. For example, if 4 files are added to an empty archive, the archive must contain: The first header, file 1 name and contents, file 2 name and contents, file 3 name and contents, file 4 name and contents, empty header.

If a delete command is given, your program should locate the header for the required file and rewrite the header back by marking the corresponding header position as deleted. Do NOT update any other information!

Error Checking

Your program should implement "all or nothing" semantics, i.e., you should verify all input arguments for validity and unless all are valid, you should not create a new archive, or modify an existing archive file. Some possible problems that you should definitely check for are:

1. Not an archive file.
2. Non existing file argument.
3. Non-regular file input argument.
4. File already in the archive.

In order to avoid erroneously updating a non-archive file, the program should check to see if the given file was indeed created by the `ctar` program. The program can do this by ensuring the initial header contains the magic number in the `magic` variable, and the size of the archive file in `eop`. While each header must contain the exact magic number, every header that is not the first must have 0 in the `eop` field.

The above error list is not exhaustive. Your program should check the return value from any kernel call, give an appropriate error message on standard error and terminate. The program does not need to protect against file corruption if an unanticipated I/O error is encountered.

utar program

Program `utar` should read a given archive file, validate that it is indeed an archive file and extract each of the stored files one by one. Do not worry about the time stamps of the files you are restoring. Each untarred file must have 0644 permissions. If the directory contains a file with the same name, the program should not override the file by generating an error message of the form:

File `samplefilename` already exists. Move away that file and restart.

and terminate.

Submission Requirements

Your submission must be written in C. You may NOT use any stdio functions (no printf, scanf, etc.) except sprintf() and sscanf(). That means if you open the manpage of a function, and it asks you to include `<stdio.h>`, you may NOT use it.

Use Canvas to submit a GNU tar file named `prog2.tgz` that contains:

- A copy of the source **with comments**.
- A makefile with *all*, *ctar*, *utar* and *clean*
- A file named TESTS in the main project directory that contains a description of the test cases you executed against the code to verify correct operation.
- A README file which describes what works and what does not and how to use any of the tests in the TESTS directory.

When I execute the commands: `tar xzf prog2.tgz; make` against your submission, two executables named `ctar` and `utar` should be created in the current directory. The tar file can be created by executing the command `tar czvf prog2.tgz *` from the directory containing your makefile and final program code.