Tar Archiver

CS3411 Spring 2020
Program Two
Assigned: Thursday, February 13
Due: Tuesday, February 25, AoE

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.
The archive file should begin with a header which contains pointers to where each file starts, the size of the file, and another pointer to its name. This pointer is a relative offset in number of bytes from the beginning of the file to where the name is stored within the archive. When storing a file name, the string should be preceded by 2 bytes (short) that contain the strings length. The string should NOT be null terminated. The header will provide enough space to store information about 8 files and an overflow link *next*. The link, when not zero is a pointer (byte offset from beginning of file) 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[8];     /* File size in bytes for files 1..8                          */
  char deleted[8];       /* Contains binary one at position i if i-th entry was deleted. */
  int  file_name[8];     /* 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 eight files can be written to the archive. If there are more than 8 files to be archived, create a new header at the end of the file. The process will be repeated for the entire list of files to be archived.

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. You do not have to delete the file contents from the archive file.

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 erronously 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.

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 blah blah 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: `gtar zxf 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 `gtar czvf prog2.tgz *` from the directory containing your makefile and final program code.