COSC 3750
Linux Programming
Homework 6, Write "wytar"

1 Intro

This time we write a more complex utility. The tar utility is a method of archiving files. It requires storing metadata (information about data) for files in the archive as well as the actual file data. The information includes the path to the file, ownership info and permissions.

In addition to basic file I/O, we are reading directory information and filesystem object status. This project will, of course, include a Makefile, and a pseudocode "listing". Again, please do some serious planning before programming. I do not mind answering questions when you have a plan and some little thing is not working. But explaining the lectures, the man pages, and any documentation I have provided or referenced gets really old, really fast.

2 Your job

You will create a simple text file named **hw6_pcode.txt** that contains a pseudocode version of the program. This is pseudocode. That means no braces, no C-like indentation although some indentation is okay. No "if(a==0)" type statements, just words! Create this file first, instead of after you have done all the other work. Put in more than "program does stuff" statements. See the pseudocode guide on the WyoCourse main page.

Write a C program. This program will be called *wytar*. It will be our version of the standard system utility "tar". This will be a typical program using argc and argv and contained in file named *wytar.c*. You will NOT put function code in the main source code file. I **insist** that you put any functions in a separate file (or files) AND have at least one header file with the function prototypes. The only thing that should come before main is *#include* statements. I do not want to see any global variables. If seems to be a problem, talk to me.

Along with the C source code file you will create a Makefile, named *Makefile*. It will have a target, *wytar*, that is dependent on at least *wytar.c* and generates the executable, *wytar*. The Makefile will also have an appropriate .PHONY target, a "clean" target that will remove the executable, any intermediate files, and any core files (core.*). If there are any intermediate files, like object (.o) files, there will also be a "tidy" target that removes just the intermediate files. And if you use "tidy" and then add it as a prerequisite for "clean", you do not have repeat the commands used for tidy. The Makefile will also correctly define

and use CC and CFLAGS. Make sure that you are using -ggdb and -Wall for the flags. You will make sure there are NO warnings generated by YOUR code.

The "wytar" target **could** have more dependencies than wytar.c if you choose to create functions and place them in separate files from the main program and create header files with declarations/prototypes. Just make sure that they are correctly part of the dependencies AND DO NOT COMPILE HEADER FILES. If you do choose to add a set of functions in separate files, make sure that you create a header so that you can include the prototypes in the main program. Note that you may have ten C source code files and only one header file. As long as all the info is there, the compiler does not care. Do NOT put C source code in headers, do NOT *include* .c files.

The compiled program will

1. Process, without modifying, its arguments via **argv** and **argc**.
2. Arguments, other than options, will be assumed to be the names of filesystem objects. If some argument cannot be opened or accessed, a suitable error message (just use perror()) about that particular object will be printed and the program will continue.
3. This program will support three options
   - "-c": This option tells *wytar* to create a archive.
   - "-x": This option tells *wytar* to extract the files from a archive. -c and -x are mutually exclusive. That means the use of both at the same time is an error, a simple message will be printed and the program will exit.
   - "-f": This option tells *wytar* that the following name (next argument) is the name of the archive to process with either the -c or the -x option. Other than the fact that the user should make sure the name is reasonable (like no spaces) you do NOT have to do any kind of verification on the string. Just try to open that file appropriately. As you are opening for writing with the "-c" option (NOT append) it does not have to exist. If the open fails, perror() and terminate the program.
4. The options must come before any non-option arguments. They can appear in any order. It is an error if one of *-c* or *-x* is not specified. It is always an error if *-f filename* is not specified. Do not bother to check beyond the third argument (argv[4]) for options. Just treat everything else as a filesystem object. If it cannot be accessed with stat(), perror() and continue.
5. The archive will consist of a set of records. Each record will consist of a 512 byte header followed by the content of the filesystem object. There will be no special processing of the file data. Files will be written to the archive in 512 byte blocks and if the final block would not have 512, then pad with 0 bytes. Do you really have to only write 512 bytes? Yes!!!!!!!!!

6. The archive **will** be terminated with two 512 byte blocks of 0 bytes. Do not forget this!

7. *wytar* will only handle directories, symbolic links, and regular files. For directories and symbolic links the archive will contain no data blocks, just the header.

8. Close all files opened by the process as soon as possible.

9. When extracting an archive, if one of the arguments is of the form *directory/filename*, the possibility exists that *directory* does not yet exist when extracting *directory/filename*. If this happens *fopen()* will fail and set **errno** to ENOENT. The program will then determine which path component does not exist, create that component (or components), then continue with extracting the file.

10. The only functions that will be used to read/write files are **fread()** and **fwrite()**.

11. You will check AND correctly test the return values of all functions (you may ignore fprintf/sprintf/sscanf return values). If there are any errors, a reasonable error message will be printed (to stderr, see if perror() would be appropriate). Whether or not the error causes the program to terminate is a judgment call. Note here, you CAN be wrong and lose points.

The format of the header is approximately that specified by POSIX. Here is the format. Please pay particular attention to the notes after the format.

Refer to: **man 5 tar**, and

http://www.gnu.org/software/automake/manual/tar/Standard.html, and
/usr/include/tar.h

You will put the *struct* declaration into a header file of your own to clean up the main program. And no, you do not have to name either the structure OR the header file "tar_header."

```
struct tar_header
{                               /* byte offset */
  char name[100];               /*   0 */
  char mode[8];                 /* 100 */
  char uid[8];                  /* 108 */
  char gid[8];                  /* 116 */
  char size[12];                /* 124 */
  char mtime[12];               /* 136 */
  char chksum[8];               /* 148 */
```

```
  char typeflag;                /* 156 */
  char linkname[100];           /* 157 */
  char magic[6];                /* 257 */
  char version[2];              /* 263 */
  char uname[32];               /* 265 */
  char gname[32];               /* 297 */
  char devmajor[8];             /* 329 */
  char devminor[8];             /* 337 */
  char prefix[155];             /* 345 */
  char pad[12]                  /* 500 */
};

##############
# These macros can be added by including tar.h
# typing or copy and pasting them will surely result in errors
##############
#define TMAGIC   "ustar"        /* ustar and a nul */
#define TMAGLEN  6
#define TVERSION "00"           /* 00 and no nul */
#define TVERSLEN 2

/* Values used in typeflag field.  */
#define REGTYPE  '0'            /* regular file */
#define AREGTYPE '\0'           /* regular file */
#define LNKTYPE  '1'            /* link */
#define SYMTYPE  '2'            /* reserved */
#define CHRTYPE  '3'            /* character special */
#define BLKTYPE  '4'            /* block special */
#define DIRTYPE  '5'            /* directory */
#define FIFOTYPE '6'            /* FIFO special */
#define CONTTYPE '7'            /* reserved */


/* Bits used in the mode field, values in octal.  */
#define TSUID    04000          /* set UID on execution */
#define TSGID    02000          /* set GID on execution */
#define TSVTX    01000          /* reserved */
                                /* file permissions */
#define TUREAD   00400          /* read by owner */
#define TUWRITE  00200          /* write by owner */
#define TUEXEC   00100          /* execute/search by owner */
#define TGREAD   00040          /* read by group */
#define TGWRITE  00020          /* write by group */
#define TGEXEC   00010          /* execute/search by group */
```

```
#define TOREAD   00004          /* read by other */
#define TOWRITE  00002          /* write by other */
#define TOEXEC   00001          /* execute/search by other */
```

MORE NOTES:

In this "nul" means the character '\0' which is a single byte of all zeros. The word "blanks" means **space** characters.

All characters in header blocks are represented by using 8-bit characters in the local variant of ASCII. Each field within the structure is contiguous; that is, there is no padding used within the structure. Each character on the archive medium is stored contiguously.

Bytes representing the contents of files (after the header block of each file) are not translated in any way and are not constrained to represent characters in any character set. The tar format does not distinguish text files from binary files, and no translation of file contents is performed.

The *name*, *linkname*, *magic*, *uname*, and *gname* are nul-terminated character strings. If the *name* or *linkname* strings are 100 characters, or the *prefix* is 155 characters, the nul is NOT included (this allows 255 character paths). All other fields are zero-filled octal numbers in ASCII. Each numeric field of width **w** contains **w** minus 1 digits, and a nul. NOTE: you might want think about how to test this situation.

The *linkname* is only valid if the *typeflag* is SYMTYPE. It does not use *prefix*; files that are links to pathnames >100 chars long can not be stored in a tar archive. NOTE: there is no error message for this, just quietly ignore such a link.

If the first character of *prefix* is '\0', the file name is *name*; otherwise, it is *prefix*/*name*. Files whose pathnames don't fit in that length can not be stored in a tar archive.

If *typeflag*=={SYMTYPE,DIRTYPE} then size must be 0.

devmajor and *devminor*, we will not use these at all. Set to all 0.

The *name* field is the file name of the file, with directory names (if any) preceding the file name, separated by slashes.

The *mode* field provides nine bits specifying file permissions and three bits to specify the Set UID, Set GID, and Save Text (sticky) modes. Values for these bits are defined above. When special permissions are required to create a file with a given mode, and the user restoring files from the archive does not hold such permissions, the *mode* bit(s) specifying those special permissions are ignored. Modes which are not supported by the operating system restoring files from the archive will be ignored. Unsupported modes should be faked up when creating or updating an archive; e.g., the group permission could be copied from

the other permission.

The *uid* and *gid* fields are the numeric user and group ID of the file owners, respectively. If the operating system does not support numeric user or group IDs, these fields should be ignored. NOTE: YOU will not be able to change these on the files you extract.

The *size* field is the size of the archived file in bytes; linked files are archived with this field specified as zero.

The *mtime* field is the data modification time of the file at the time it was archived. It is the ASCII representation of the octal value of the last time the file's contents were modified, represented as an integer number of seconds since January 1, 1970, 00:00 Coordinated Universal Time.

The *chksum* field is the ASCII representation of the octal value of the simple sum of all bytes in the header block. Each 8-bit byte in the header is added to an unsigned integer, initialized to zero, the precision of which shall be no less than seventeen bits. When calculating the checksum, the *chksum* field is treated as if it were all blanks.

Further note on checksum. **man 5 tar** states that for the "old-style archive format", *This field should be stored as six octal digits followed by a null and a space character.* The system tar utility can be made to use the **ustar** format (tar -H ustar) when it creates an archive. Regardless what the previous note about fields of width **w** containing **w-1** characters and a nul, the system utility when using this format, uses the OLD format of 6 characters, a nul, and a space. Go figure. That probably means that the comment about "set the checksum field to all spaces" really IS talking about *space* characters.

The *typeflag* field specifies the type of file archived. If a particular implementation does not recognize or permit the specified type, the file will be extracted as if it were a regular file. As this action occurs, tar issues a warning to the standard error.

We will only use the REGTYPE, DIRTYPE and SYMTYPE flags.

REGTYPE, AREGTYPE: These flags represent a regular file. In order to be compatible with older versions of tar, a typeflag value of AREGTYPE should be silently recognized as a regular file. New archives should be created using REGTYPE. Also, for backward compatibility, tar treats a regular file whose name ends with a slash as a directory. YOU JUST WORRY ABOUT REGTYPE and do not bother to check anything for backward compatibility.

SYMTYPE: This represents a symbolic link to another file. The linked-to name is specified in the linkname field with a trailing nul.

DIRTYPE: This flag specifies a directory or sub-directory. The directory name in the name field should end with a slash. On systems where disk allocation is performed on a directory basis, the size field will contain the maximum number of bytes (which may be rounded to

the nearest disk block allocation unit) which the directory may hold. A size field of zero indicates no such limiting. Systems which do not support limiting in this manner should ignore the size field. WYTAR uses 0.

The *magic* field indicates that this archive was output in the P1003 archive format. If this field contains TMAGIC, the *uname* and *gname* fields will contain the ASCII representation of the owner and group of the file respectively. If found, the user and group IDs are used rather than the values in the *uid* and *gid* fields. NOTE: That last means look up the names on the system where the archive is being extracted and possibly change the UID and GID.

For references, see ISO/IEC 9945-1:1990 or IEEE Std 1003.1-1990, pages 169-173 (section 10.1) for Archive/Interchange File Format; and IEEE Std 1003.2-1992, pages 380-388 (section 4.48) and pages 936-940 (section E.4.48) for pax - Portable archive interchange.

3 What to turn in

You will upload a tar archive, created with the **system** tar (not our version, I don't trust you). It will be named hw06.tar or hw06.tgz if you choose to compress it. The archive will contain at least four files, **wytar.c**, **hw06_pcode.txt**, **struct header file**, and **Makefile**. If you have other files, include those too. Please do not make me ask you for the extra files. And only tar the source FILES, not directories, and NOT objects or executables, and not text editor backup files. Executing *make* in a directory containing these files will generate the executable *wytar*.