Computer Science 311

Assignment 3 Write-up

Austin Dubina

1 Assumptions

- This program will only work on even sized files.
- The program always assumes the second argument from the commandline is an actual archive file.
- Read/Write buffers are set to a block size of 1024 bytes.
- Although the program should would with any archive provided, assumption one holds true, this program has only been tested validated on archive file provided with the program.
- The file_perm_string function has been integrated with main program source code.
- While extracting files from archive, the program assumes ALL files are being created for the first time.
- The delete function deletes multiples of specified file names from archive.

2 Program Design

Overview:

The purpose of this program was to reverse engineer and emulate portions of the ar utility from the perspective of a user appending, extracting, deleting, and printing using the ar utility. The inspiration for the design of this program was mostly derived from my experiences using the ar utility and readings about its structure in various headers.

After carefully reading the posted assignment for this project I knew I was going to need at minimum five functions. As always, I would need a main function as a point of entry to my program and where all other functions would be called from. I knew that i needed at one function for printing, extracting, deleting, and appending data. From reading the assignment I knew there was going to be some overlapping in functionality which may not justify a whole extra function; In hindsight after doing this assignment I have realized this is generally bad practice, and have learned creating extra functions makes debugging and coding much easier.

Needless to say extra functions were created to in addition to the five mentioned above to simplify the program and reduce duplication of code. The following is a detailed description of each function and its purpose.

Program Functions:

`int main()`
Purpose:
This function serves as the main entry point into my program

Description:
Int main takes command line arguments (argc and argv) and parses them using getopt. a switch statement is used to make a selection and execute a specific function specified by the user. If a

valid flag (denoted by a -) is not set, the program throws and error and exits.

`void table_cont()`
Purpose:
Print a concise or verbose table of contents of the archive.

Description:
The table of contents function opens an archive specified by the user. If the archive was opened successfully, the function then seeks past SARMAG. While bytes read is not equal to zero the function checks to see if a flag was set designating whether or not the function will print a verbose or concise table. The function then prints a formatted string to the terminal using printf. After each print, the function seeks past the contents of the file equal to number of bytes in ar_name. checks to see if a flag was set designating whether or not the function will print a verbose or concise table.

Additional notes:
This function uses code provided in class, file_perm_string() function, to convert the octal ar_mode number to a human readable string similar to ar. the date modified portion of the string was converted to broken down time using the localtime() function and printed as a formatted string from time struct.

`void append_archive`
Purpose:
Opens existing archive for file appending or creates a new one.

Description:
To successfully append a file to an archive, a check must be made to determine if the archive already exists. If the archive does not exist, the function creates a new archive using the open function and the O_CREAT, O_APPEND, and O_WRONLY flags. The create_flag is set to 1 and the append_files function is called. If the archive already exists, the archive is opened and the append_files function is called with the create_flag set to 0.

`void append_files`
Purpose:
Append named files to archive specified by append_archive function

Description:
For each argument specified in the command line, this function writes the header and file content in an appending fashion to the archive. If the archive was recently created the function will first write the magic header. the stat function is used to gain information about file being written to the archive. Using the snrpintf function, the formatted header string is written following the contents of the file. While the number of bytes read is not equal to 0, write the number of bytes read up to BLOCKSIZE to the archive. The function ends by printing out the number of bytes written using the printf function.

`void extract_files`

Purpose:
Extracts named files from archive and creates files of similar name into current working directory

Description:
The extract works by lseeking past SARMAG and reading the first header into a buffer. The buffers is then passed to the file_position function to check if the current filename in header matches any of the arguments specified in the command line. If a match is found the function attempts to create a new file using open. If the file already exist the program exits immediately. This design choice is not optimal however the program assumes all file being extracted are being done so for the first time. Once a new file has been created in the current working directory file is updated with the correct modification times from the archive. While the total number of bytes read is less than the ar_size of the file, the function writes to to BLOCKSIZE number of bytes. The whole process is repeated until the function has moved though every header in the archive.

`void delete_files`
Purpose:
delete named files from archive.

Description:
The delete_files function works by checking opening an existing archive. If the archive was opened successfully a new file archive is created with the same name by unlinking the original archive. Once SARMAG has been written and the first header has been read into a buffer, the buffer is then pass to the file_position function to check if the current filename matches any of the arguments specified by the command line. If the filename matches a command line argument, the file is skipped and the next header is read into a buffer. this repeats until all headers have been read. For all non matches, both the header and file content is read and written to the archive up to BLOCKSIZE.

`int file_position`
Purpose:
compare current header name to all arguments specified by argv.

Description:
The file_position function compares the current header name to each argument in argv by replacing / character with NULL character for specified header name.

`char** dir_args`
Purpose:
Parse each regular file string into array to be used by append_archive function.

Description:
This function gets a list of regular filenames in the current working directory by using the getcwd function. While readdir function is not NULL place each filename into array.

3 Additional Questions

1. what do you think the main point of this assignment is?
   The main point of this assignment is to become more comfortable with the C programming language. This program introduced a lot of new functions and system call from various libraries that we have not seen before. This required the student to read what each function did, what its arguments were, and what the function returned.

2. how did you ensure your solution was correct? Testing details, for instance.
   I systematically went through each function to test every imaginable test case. As part of my check, I put error checking on most system calls and functions. I also test my program with very large text files. I also listed my assumptions at the top of this document for anything I was not sure about.

3. what did you learn?
   Not only did I learn a lot about system I/O in a unix environment but I learned how to look-up functions using the manual pages. This is also probably the first real program I have made in C completely from scratch, which I am very proud of.

4 Difficulties

The biggest difficulty I had with this assignment was time. I got stuck on a lot of simple things that seemed to take forever to resolve. Once I started becoming more familiar with the manual pages, things got a lot easier. This project introduced a lot of new functions and system calls, the time required to implement these calls and functions for the first forever, however once I completed a couple the rest got much easier.