Assignment 1

CS330: Operating Systems

August 23, 2021

1 Infinite Chain of Unary Operations [20 Marks]

In this question, you need to write three c programs defined in `Part1/square.c`, `Part1/double.c` and `Part1/root.c` which perform square, double and square root operations respectively on a non-negative integer such that generated executables with these programs can be chained in any pattern.
The order of the operations in the chained pattern would be from **left to right**.

Synopsis

```
$ ./square root double root square integer_number
```

Example

```
$ ./square root double root 8
4
```

Since, the order of the operations in the chained pattern is from **left** to **right** hence, the chained pattern should be viewed as:

```
root(double(root(square(8)))) = 4
```

Output

Print the final result only (as shown in the example).

Note

- Atleast 1 unary operation and atmost 100 unary operations will be specified during testing.
- If your implementation results in parent-child relationship between processes, then parent process must wait for its child process to exit.

- You can assume that the result of operations will always fit in unsigned long long data type.
- If square root of number is a fraction then round off the result to the nearest integer, e.g. 2.5 should be rounded to 3 and 2.4 should be rounded to 2.

Error handling

In case of any error print "UNABLE TO EXECUTE" as output.

System calls and library functions

You **must only use** the below mentioned APIs to implement this question.

```
- fork                  - malloc
- exec* family          - free
- strcpy                - strcat
- strcmp                - strtok
- ato* family           - wait/ waitpid
- printf, sprintf       - sqrt
- round                 - exit
```

Testing

Run the script `Part1/run_tests.sh` for running the provided sample test cases which contains 3 test cases. A sample output after running the script would be:
Test 1 is Passed
Test 2 is Passed
Test 3 is Passed

2 Simple Tar Utility [35 Marks]

GNU tar is an archiving program designed to store multiple files in a single file (an archive), and to manipulate such archives. You can read more about it in its man pages ($man tar, $info tar). You can use original tar utility to familiarize yourself with it.
In this question you will be implementing a simpler version of tar utility that should support creation of tar file, extraction of all or a single file from a tar file, listing the contents of tar file.
Detailed description of your tasks is as following:

2.1 Creation and Extraction of tar file [20 Marks]

2.1.1 Creation of tar file

Command line arguments

```
    $./myTar -c <path of a directory> <tar filename>
Eg: $./myTar -c ~/Downloads/Movies   MyMoviesCollection.tar
```

Description of command line arguments

- myTar: simple tar utility that you have to implement
- -c: This argument tells your tar utility that creation of tar file operation needs to be done.
- path of a directory: path of directory whose all files need to be archived. You can assume that this directory will contain only regular files. Name of each regular file will be of max 16 characters long. There will atleast 1 file in this directory and atmost 30 thousand files. Size of each file can vary from 0 bytes to 1GB.
- tar filename: Name of the tar file that will be created by your utility as output file. It should be created in same directory whose file's are to be archived. Example: In figure 1, MyMoviesCollection.tar gets created in ∼/Downloads/Movies directory.

Description of functionality

- Your tar utility should read the contents of input directory to know the name of each file to be archived. You can use readdir() for this purpose. (Please refer the manpages of opendir(), readdir(), closedir() for more info).
- For each file, it should store the contents of that file in output tar file. You can assume that output tar file doesn't already exists and you have to create it with permissions 0644 and fill it with data.
- You should store the contents of each file in tar file in such a manner that other operations i.e. extraction of files from tar files, listing of contents of tar file can be done correctly.

Assumptions

- You can assume that there is enough storage space present for archive file to get created.
- You can assume that during the creation of tar file, no new file will be added to or deleted from the directory being archived.

Error Handling

- In case of any error, print "Failed to complete creation operation"

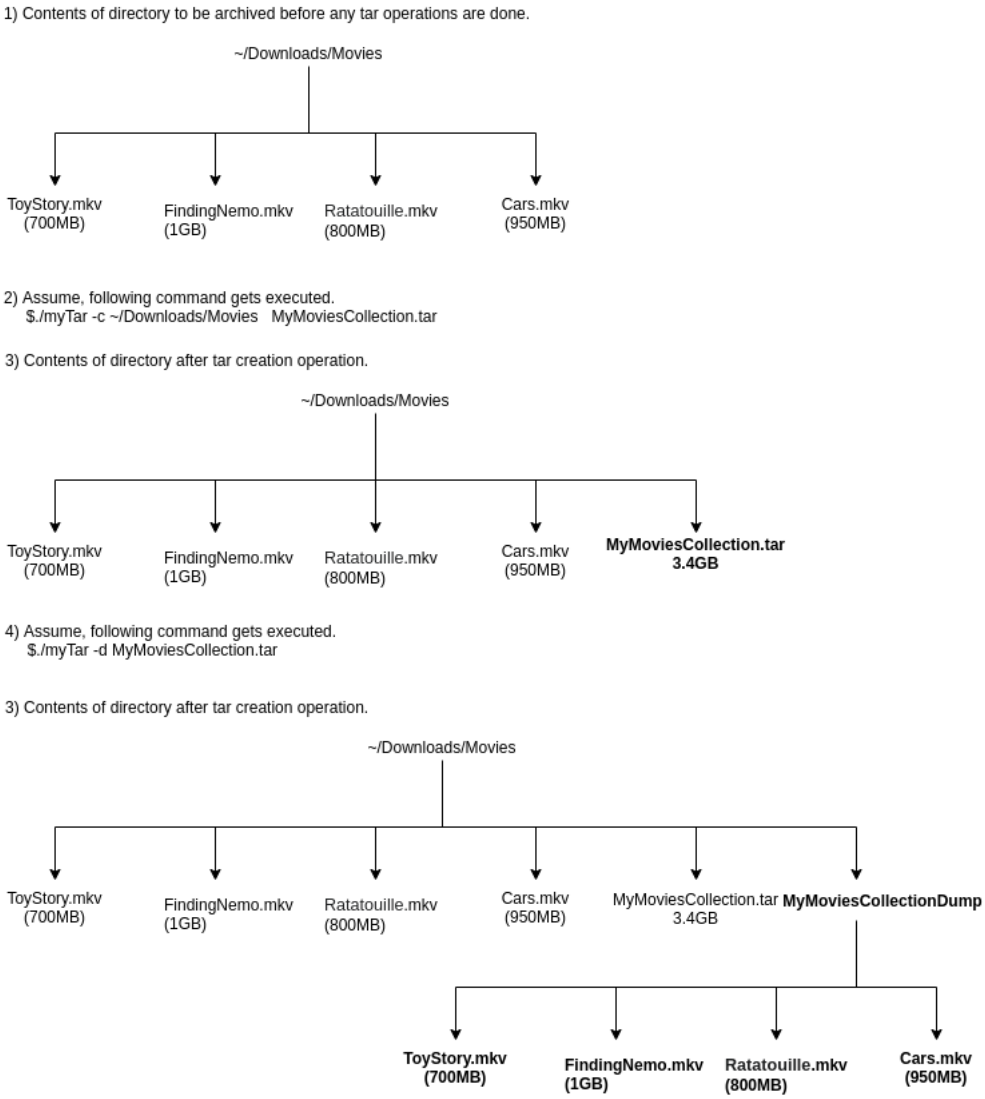

Figure 1: Example of tar file creation and extraction operations

2.1.2 Extraction of all archived files from a tar file

Command line arguments

```
    $./myTar -d <tar filepath>
Eg: $./myTar -d ~/Downloads/Movies/MyMoviesCollection.tar
```

Description of command line arguments

- myTar: simple tar utility that you have to implement

- -d: This argument tells your tar utility that extraction of all files operation needs to be done.
- tar filepath: Path of tar file on which extraction operation needs to be done.

Description of functionality

- Your tar utility should create a new directory named "tar filename(without .tar) + Dump" in the directory where tar file is present. Example: In figure 1, MyMoviesCollectionDump gets created in ∼/Downloads/Movies directory. You can assume that no such directory already exists. This directory should be created with mode 0777.
- For each file, that was archived in the tar file, create a new file in this newly created directory. Name of this file should be same as its original name when it was archived and likewise its contents should also be restored to be same as the original contents. Permissions of each extracted file should be set as 0644.

Assumptions

- You can assume that there is enough storage space present for the contents of archive file to get extracted.

Error Handling

- In case of any error, print "Failed to complete extraction operation"

2.2 Extraction of single file from a tar file [5 Marks.]

Command line arguments

```
    $./myTar -e <tar filepath> <single filename>
Eg: $./myTar -e ~/Downloads/Movies/MyMoviesCollection.tar Cars.mkv
```

Description of command line arguments

- myTar: simple tar utility that you have to implement
- -e: This argument tells your tar utility that extraction of single file operation needs to be done.
- tar filepath : path of tar file on which extraction operation needs to be done.
- single filename : name of the file that needs to be extracted from tar file.

Description of functionality

- Your tar utility should create a new directory called "IndividualDump" in the directory where tar file is present. This directory may or may not already exist. If it already exists, continue, else create it with mode value 0777.
- Scan your tar file and find the file matching the requested file to be extracted. If no match is found, print following message "No such file is present in tar file."
- If relevant file is found, then create a new file in "IndividualDump" directory with searched filename. You can assume no such file already exists in the directory. This new file should be created with 0644 as its permissions. Likewise this searched file's contents should also be restored to be same as the original contents.

Assumptions

- You can assume that there is enough storage space present for extracted file to get created.

Error Handling

- In case of any error, print "Failed to complete extraction operation"

2.3 Listing the contents of tar file [10 Marks.]

Command line arguments

```
    $./myTar -l <tar filepath>
Eg: $./myTar -l ~/Downloads/Movies/MyMoviesCollection.tar
```

Description of command line arguments

- myTar: simple tar utility that you have to implement
- -l: This argument (it is character 'ell', not digit one) tells your tar utility that listing of the contents of tar file needs to be done.
- tar filepath : path to tar file whose contents need to be listed.

Description of functionality

- Create a file "tarStructure" with permissions 0644 in directory where tar file is present. You can assume that this file doesn't already exist.
- Write information about tar file such as size of tar file and number of files archived in tar file into "tarStructure" file.
- Scan tar file and write filename and the size of each archived file to "tarStructure" file.

- Above mentioned information should be written to "tarStructure" file in following format:

```
size of tar file in bytes<newline>
Number of files archived in tar<newline>
filename<space>file size in bytes<newline>
filename<space>file size in bytes<newline>
    .
    .
    .
Eg: Output of following command:
        $./myTar -l MyMoviesCollection.tar
    will be:
        3643801600
        4
        ToyStory.mkv 734003200
        FindingNemo.mkv 1073741824
        Ratatouille.mkv 838860800
        Cars.mkv 996147200
```

Error Handling

- In case of any error, print "Failed to complete list operation"

2.4 System calls and library functions

You **must only use** the below mentioned APIs to implement this question.

```
- open          - close
- read          - write
- opendir       - closedir
- readdir       - mkdir
- chdir         - stat,fstat,lstat
- lseek         - exit
- malloc        - free
- memset        - strcmp
- strncpy       - strlen
- printf        - sprintf
- strtok
```

2.5 Testing

To check whether your implementation of creation and extraction of tar file is working correctly, do make in Part2 directory and then run Part2/run_test.sh script. It creates a tar file with files present in Part2/test directory and then extracts them. It prints following output if your utility is implemented correctly:

```
File 1 extracted correctly
File 2 extracted correctly
File 3 extracted correctly
File 4 extracted correctly
```

3 T20 World Cup [45 Marks]

The annual T20 Cricket World Cup could not be conducted this year due to COVID. However, bowing to public demand, ICC decided to conduct a virtual T20 World Cup with social distancing by which the playing teams do not have to come into direct contact with each other. The task of orgainizing this tournament has now fallen upon you.

3.1 Tournament Details

- The tournament has the typical structure of 2 groups of 4 teams each. The 8 team names are given as an array (`team_names` array in `utils.c`). The array index corresponding to each team is that team's current ICC rank (starting from 0 to 7).
- In each group, each team will play a match against every other team (so in total C_2^4 i.e. 6 matches per group will be played).
- Each match will have a toss, 20 overs play per side using upto 10 wickets and declaration of the result.
- Firstly, group stage matches will happen. The team that wins most matches in its group advances to the finals from that group.
- Thus, the winning teams from each group play against each other in the final. Winner of the final match is declared as the winner of the tournament.
- In case of tie (whether in a match or in group rankings before advancing to finals) the team with lower rank is declared the winner.
- Each team will not interact with the opposite team directly during the match. Instead they will interact with a common mediator. The bowling team will send a message to the mediator telling what type of ball will be bowled (pace, spin, etc.), and the batting team will tell what type of shot will they hit (helicopter shot?). The mediator takes note of both the pieces of information and simulates the ball virtually and decides the result of that ball (whether wicket or some runs scored etc.).

3.2 Implementation Details

- You have been given base templates of `wc.c`, `wc.h` and `utils.c`. You have to complete the definitions of the five functions `teamPlay()`, `endTeam()`, `match()`, `spawnTeams()` and `conductGroupMatches()` present in `utils.c` as per the given specifications. DO NOT EDIT the other two files.

- `wc.c` process will act as the host process. Initially, it will fork 8 processes. Each team is represented by one these 8 forked process.

- `wc.c` process will also fork 2 more processes. These processes are called Group processes. First process represents Group A and second process represents Group B. These Group processes shall conduct the matches of the corresponding group.

- When a match will be played between 2 teams in group stage, for each ball bowled in the match, team processes will read their action on that ball from their corresponding input files and pass this data to their corresponding group process via pipe (i.e. group process acts as mediator in group stage). The Group process will decide the outcome of that ball. For more details, see `match()` and also `teamPlay()` function descriptions below in Sections 3.2.4 and 3.2.5 respectively.

- Since, finalist processes will belong to different groups, hence, host process will conduct the final match i.e. finalist processes will communicate their actions on each ball to host process (host process acts as mediator in during final match).

- **Note:** Group A and Group B matches MUST be played parallel w.r.t to each other, but matches in the same group will follow a sequential order. Eg: If teams 0 to 3 belong to group A and teams 4 to 7 belong to group B, then match between teams 1 and 2 and another match between teams 5 and 6 can happen in parallel.
  However, all matches to be held in group A will happen in following order: 0 vs 1, 0 vs 2, 0 vs 3, 1 vs 2, 1 vs 3, 2 vs 3. Likewise for group B.

- Make use of the `teams` struct declared inside `wc.h` and defined in `wc.c`. It contains two pipes per team. The `matchpipe` allows the team to send messages to the group / host process for each ball. The `commpipe` allows the group / host to send messages to the team (whether to continue sending action for next ball via matchpipe, or to stop and terminate).

3.2.1 void spawnTeams(void)

Description of functionality:

For each of the 8 teams, host does the following:

- assign name to the corresponding entry in `teams` array. The names are given in the `team_names` array.
- setup both pipes using the `matchpipe` and `commpipe` variable. These pipes shall be used to communicate between team and host/group. Team writes to its `matchpipe` and reads from its `commpipe`.
- fork a new process (team) and in it, set `processType` = Team index where Team index is the array index corresponding to the team in the `team_names` array in `utils.c`.
- team process shall call the `teamPlay()` function, while host returns back to main.

3.2.2 void endTeam(int teamID)

Description of args:

- A single int argument `teamID` which represents a single team, i.e. the team corresponding to `team_names[teamID]`.

Description of functionality:
You have to terminate the team process corresponding to `team_names[teamID]`. Use the `commpipe` of that team to send it a message telling it to terminate.

3.2.3 void conductGroupMatches(void)

Description of functionality:

- For each group, the Host does the following:
  1. setup a pipe. This pipe (group pipe) shall be used to communicate between host and group process.
  2. fork a new process (group process)
  3. Finally host reads a number from each group pipe. This digit is the index of the leader of the given group which will advance to the finals. Save these indices in the variables `finalTeam1` and `finalTeam2`.
- Once forked, a group process will do the following:
  1. conducts matches between all four teams in its group (1-2, 1-3, 1-4, 2-3, 2-4, 3-4) in order.
  2. To conduct a match, it calls `match(x,y)` where `x` and `y` are the indices of the contesting processes. `match(x,y)` returns the winner of the match - either `x` or `y`.
  3. after all 6 matches, it determines the group leader which is the team in the group that won most matches (in case of a tie, lower index team leads automatically).

4. All teams in the group EXCEPT the group leader are terminated via an `endTeam()` call.
5. Finally, send the index of the leading process to host through group pipe.
6. Now self-terminate the group process.

3.2.4 int match(int team1, int team2)

Description of args:

Two integer arguments which represent the indices of the two teams playing in this match. `team1` has lower index than `team2`.

Description of functionality:

- First do toss - read a digit from `matchpipe` of both teams. If sum of both the digits read from `matchpipes` is odd, then team1 bats first, else team2 bats first.
- Create a new output file named "XvY" where X and Y are the names of the first batting and bowling teams respectively. If however X and Y are from different groups, then it means that this is the Final match and the filename should be "XvY-Final".
- In output file, append the line "Innings1: X bats". Then in a loop of 20x6 = 120 iterations (balls) do the following:
  1. Read one digit from both batting team and bowling team's `matchpipe`s.
  2. If both the digits are not equal, then the batting team scores runs equivalent to the digit read from its own `matchpipe`. if both digits are equal, bowling team gets a wicket.
  3. the innings is over when either 10 wickets lost or 20 overs (120 balls) done.
  4. At the fall of each wicket, write to the output file the number of runs scored by the batsman that just got out in a new line (assume that all runs scored since the previous wicket belong to this same batsman). The format should be "M:N" which means the Mth batsman scored N runs.
  5. If the innings is over at 20 overs and there is currently a NOT OUT batsman, print it in the format "M:N*".
  6. At the end of the innings write the total score: "X Total: N" where X is the batting team and N is the total runs scored
- Next, write a blank line to output followed the line "Innings 2: Y bats". Simulate the second innings as a loop just as above. This innings can end either when Y meets the target score, in which case Y wins, or finishes 20 overs / loses 10 wickets without meeting the target, in which case X wins. In case of a tie, the team with lower index (`team1`) wins.

- Write the output to the output file in a new line as "X beats Y by n runs" or "Y beats X by n wickets" as per the case. Return the index of the winning team. In case of tie, it should be "TIE: X beats Y"

Return Value:

- Returns the index of the winning team (either `team1` or `team2`).

3.2.5 void teamPlay(void)

Description of functionality:
Each process that enters this function corresponds to a team. The team is identified by the `processType` variable which is the index of the team in the `team_names` array.
There are two tasks to be taken care of:

1. Read one digit at a time from input file and send to host / group using `matchpipe`. This has to be done for each ball bowled.

2. Keep receiving messages from host / group using `commpipe`. If the message received tells the team to terminate, it means that the team is out of the tournament and this team process should self-terminate.

Understand the interaction of the pipes carefully to avoid deadlocks (for example, suppose team process is waiting for input in the `commpipe` while host/group process is waiting for input in the `matchpipe` - now both processes will be stuck and your code will not proceed). Co-ordinate the usage of `matchpipe` and `commpipe` carefully.
There is no need to reset file pointer at the end of a match. In other words, if a match ends while the team has read till the n^{th} position in the input file, then it can start the next match from that file position itself - no need to restart from the beginning.

3.3 Input and Output

There will be 8 input files per test case, one for each team. Each input file consists of a long sequence of digits in {0,6}. When a team plays a match, it will read its input from its corresponding input file.
There will be 13 output files per test case, one for each match.
There will be 1 output file per match (13 in total). The output file format is specified in the description of the match function.

3.4 System calls and library functions

You **must only use** the below mentioned APIs to implement the assignment.

```
- fork                  - exec family
- pipe                  - dup, dup2
- exit                  - wait
```

Additionally you may use common functions provided in the header files included in `wc.h`.
YOU ARE NOT ALLOWED TO INCLUDE ADDITIONAL HEADERS OR EDIT `wc.h` OR `wc.c`.

3.5 Testing

We shall test your submission by running on our testcases, and string matching the output files.
A `Makefile` has been provided with the template. You can compile your code by running the `make` command inside the Part3 directory. Once compiled, an executable by the name `wc` is created.

There is also a directory named `test` which contains folder named "1". This is a testcase (the number is the testcase ID). This folder has two additional folders in it named "inp" and "out". "inp" is where you shall find the 8 input files of the testcase. "out" is where 13 expected output files are placed.
The testcase ID will be given as a commandline argument.

```
$ ./wc [TESTCASE ID]
```

For instance, running it as `./wc 1` will make it read the input files in `test/1/inp` and write the output files to `test/1/out`.

4 Submission

- Make sure that your implementation doesn't print unnecessary data. Your output should match exactly with the expected output specified in each question.
- You have to submit zip file named your_roll_number.zip Eg: 1211405.zip containing **only** the following files in specified folder format:
  - YourRollno/Part1/square.c, YourRollno/Part1/double.c, YourRollno/Part1/root.c
  - YourRollno/Part2/myTar.c.
  - YourRollno/Part3/wc.h, YourRollno/Part3/wc.c, YourRollno/Part3/utils.c