COMP 2401 -- Project cuTunes

Due: Thursday, April 7, 2016 at 12:00 PM (noon)

Goal

For Project cuTunes (**C**arleton **U**niversity **Tunes**), you will implement in C, using the course VM, a client-server application that stores song information in a central server process accessible by multiple client processes. Each client process will offer a menu to allow the end user to manage songs. These requests, including adding/deleting a song or viewing all songs, will be communicated to the server process, which will store all song data in a singly linked list. All client processes will be able to view all the songs on the server, regardless of which client process added the song. Base code for a TCP/IP client-server architecture can be found the file *a4Posted.tar* which is posted in *cuLearn*. Your server process must be able to communicate with multiple client processes.

Learning Objectives

- get familiar with dynamic memory operations by implementing and managing a linked list
- implement programs with inter-process communications, work with signals and sockets

Instructions

1. Data structures

Your program will define the following data types:

- a **SongType** structure type: each song will have a name, an artist, an album and a duration (in minutes)
- a **`ListType`** structure type to represent a **singly** linked list that holds the **head** and the **tail** of the list
  - your linked list must be implemented as we saw in class
  - **do not use dummy nodes!** every node in the list must correspond to a song instance
  - the head of the list always points to the first node in the list
  - the tail of the list always points to the last node in the list
- a **`Node`** structure

2. Client process

You will implement a client process that interacts with the end user and allows them to manage songs on the server. Your client process must:

- establish a TCP/IP connection to the server
- show the end user a menu with the following options:
  - add a song
  - delete a song
  - view all songs currently on the server
  - exit the program
- send the server a message to indicate the operation selected (for example: "ADD", "DEL", "VIEW", "QUIT")
- depending on the selection, prompt the user for more information, and send that information to the server
  - if the user chooses to add a song:
    - prompt the user for the name, artist, album and duration of the new song
    - send each of these to the server individually
  - if the user chooses to delete a song:
    - prompt the user for the name of the song to be deleted
    - send that song name to the server

- if the user chooses to view all songs:
  - wait for the server to send a formatted string containing all song data
  - print this formatted string to the screen
- after every operation (add/delete/view songs), the user is brought back to the main menu, until they select exit

Notes:

- the client process is the only process that communicates with the end user; the server does not
- the client process does **not** store any song instances, that is the server's job; the client only deals with strings
- the client program must be divided into modular functions

3. Server process

You will implement a server process that interacts with client processes and deals with requests. Your server process must:

- define an instance of **`ListType`** local to the **`main`** function; all songs will be stored in this list
- install a signal handler for the **`SIGUSR1`** signal
  - sending a **`SIGUSR1`** signal from the command line is the only way to terminate the server process
  - receiving this signal causes the server to close all sockets and terminate
- continuously wait for a client to establish a connection
- once a connection is established, the server must continuously receive requests from that client until the client sends a "QUIT" request
- if the client sends an "ADD" request:
  - the server must receive from the client all details (name, artist, album and duration) of the song to be added
  - it must dynamically create and initialize a **`SongType`** instance for that song
  - it must add the new song to the **end** of the linked list of songs
- if the client sends an "DEL" request:
  - the server must receive from the client the name of the song to be deleted
  - it must find the correct song in the linked list of songs
  - it must remove the song from the linked list
- if the client sends an "VIEW" request:
  - the server must format into one very long string all the information for all the songs in the linked list
  - it must send that long string to the client
- once a client sends a "QUIT" request, the server must wait for the next connection request from a client

Notes:

- your program must store **one** instance of the linked list
- the list must be declared as a **local** variable in **`main`**
- the server program must be divided into modular functions
- your program must explicitly deallocate all dynamically allocated memory – you must eliminate all memory leaks!

Constraints

- do **not** use any global variables, except for the sockets declared in the base code
- compound data types **must** be passed by reference, not by value
- you must break up your program into modular and/or reusable functions
- you must break up your program into separate files that group together related functions
- you must reuse functions everywhere possible
- your program must be thoroughly commented
- do not leave memory leaks

Submission

You will submit in cuLearn, before the due date and time, one **`tar`** file that includes all the following:

- all source code, including the code provided, if applicable
- a Makefile
- a readme file that includes:
  - a preamble (program author, purpose, list of source/header/data files)
  - the exact compilation command
  - launching and operating instructions

If you are working with a partner:

- only **one partner** submits the assignment, and the other partner submits nothing; partners that make two separate submissions will be considered to have plagiarized each other's assignment and will be reported accordingly
- the readme file must contain the names of both partners
- the submitting partner must enter the names of both partners in the **Online Text** box of the *cuLearn* submission link

***** LATE ASSIGNMENTS WILL NOT BE ACCEPTED FOR ANY REASON *****

Grading

Marking components:

- Client process: 40%
  - 20 marks: Correctly adding a song
    - 4 marks: sending add request to server
    - 16 marks: prompting user and sending to server 4 parts of song (4 marks each)
  - 7 marks: Correctly deleting a song
    - 4 marks: sending delete request to server
    - 3 marks: prompting user and sending to server name of song
  - 8 marks: Correctly viewing songs
    - 4 marks: sending view request to server
    - 4 marks: receiving formatted string from server and printing it out to the user
  - 5 marks: Correctly sending quit request to server
- Server process: 60%
  - 5 marks: Correctly implementing and installing signal handler
  - 5 marks: Correct outer loop for accepting connection requests
  - 5 marks: Correct inner loop for reading consecutive requests from one client
  - 18 marks: Correctly adding a song
    - 8 marks: receiving 4 parts of song (2 marks each) from client
    - 5 marks: dynamic allocation and initialization of song
    - 5 marks: adding song to linked list
  - 12 marks: Correctly deleting a song
    - 2 marks: receiving song name from client
    - 5 marks: finding song in linked list
    - 5 marks: deleting song from linked list
  - 15 marks: Correctly responding to view songs request
    - 10 marks: correctly formatting one long string for all song data in the linked list
    - 5 marks: sending this string to the client

Notes:

- Failure to correctly define the loops in both the client and the server processes may result in the user being unable to add multiple songs, which is an execution error under the deductions below and can result in a 100% deduction of the entire assignment
- Failure to correctly implement the “View all songs feature” means that the user cannot see what songs are stored on the server, which is an execution error under the deductions below and can result in a 100% deduction of the entire assignment

Deductions:

- Packaging errors:
  - 10 marks for missing Makefile
  - 5 marks for missing readme
  - 10 marks for consistent failure to separate functions into separate files
- Major programming and design errors:
  - 50% of a marking component that uses global variables
  - 50% of a marking component that consistently fails to use correct design principles
  - 50% of a marking component that consistently fails to pass compound data types by reference
  - 50% of a marking component where unauthorized changes have been made to provided code
- Minor programming errors:
  - 10 marks for consistently missing comments or other bad style
- Execution errors:
  - 100% of a marking component that cannot be tested because the code does not compile or execute
  - 100% of a marking component that cannot be tested because the feature is not used in the code

