server.c readme

server.c consists of two functions and a main

- int setSpeakers(int decibel)
- int runServer()

Summary
The function of setSpeakers is to change the output given a certain decibel value as an input

int setSpeakers(int decibel)

What this function does is given a decibel input, it can enable and disable certain inputs and output.

The system command **amixer sset …** allows me to turn on/off certain inputs and outputs on my desktop computer.

Note: I learned about amixer sset from https://superuser.com/questions/317296/setting-audio-balance-from-command-line

I feed the decibel input from the value I receive from my client programming running on my laptop as I move around my home.

int runServer()

Note: This method contains socket programming. Large chunks of this code is taken directly from video tutorials I have used to learn socket programming. Each link is a video in the playlist. The first link will bring you to the playlist and video 1.

https://www.youtube.com/watch?v=eVYsIolL2gE&list=PL0JmC-T2nhdgJ2Lw5YdufR8MffaQdAvEf
https://www.youtube.com/watch?v=xfRdYrQUQeQ&list=PL0JmC-T2nhdgJ2Lw5YdufR8MffaQdAvEf&index=2
https://www.youtube.com/watch?v=d9pmc7oObkw&index=3&list=PL0JmC-T2nhdgJ2Lw5YdufR8MffaQdAvEf
https://www.youtube.com/watch?v=0XHIUgMVyV4&list=PL0JmC-T2nhdgJ2Lw5YdufR8MffaQdAvEf&index=4
https://www.youtube.com/watch?v=iSWrInT8CDs&list=PL0JmC-T2nhdgJ2Lw5YdufR8MffaQdAvEf&index=5
https://www.youtube.com/watch?v=pTYNQwWqB2Y&list=PL0JmC-T2nhdgJ2Lw5YdufR8MffaQdAvEf&index=6
https://www.youtube.com/watch?v=FRm9nk9ooC8&index=7&list=PL0JmC-T2nhdgJ2Lw5YdufR8MffaQdAvEf

int serverSock, clientSock;
struct sockaddr_in server, client;
unsigned int addrLen = sizeof(struct sockaddr_in);

serverSock will be the designated listening socket of the server. It will listen for clients trying to connect to the server.

clientSock will be used to accept connections from the client and receive data from.

char data [7];

This will hold the decibel that we receive from the client.
server.sin_family = AF_INET;
server.sin_port = htons(25000);
server.sin_addr.s_addr = INADDR_ANY;
bzero(&server.sin_zero,8);

Similar to the client code this tells us we are using port 25000 for the port ans we have to change from host byte ordering to network byte ordering

INADDR_ANY is to set the address of the sever as the current ip address of the network interface card of the server.

And again we zero out the portion of the struct that we don't need for local ip communications.

```
if((bind(serverSock, (struct sockaddr *)&server, addrLen)) == -1){     //binding server socket to server ip info
perror("bind error");
exit(-1);
}

if((listen(serverSock,5) == -1)){                //listening on socket
perror("listen error");
exit(-1);
}
```

The bind basically binds the server ip address(using sockaddr) to the server socket. Then it is made the listening socket. As explained above the listening socket accepts incoming connection to the server. For the purposes of the demo, I allowed a max of 5 requests to be in the queue of incoming connections. Any more than 5 connections on the queue is dropped

```
while(1){

if((clientSock = accept(serverSock, (struct sockaddr *)&client, &addrLen)) == -1){
perror("accept error");
exit(-1);
}
recv(clientSock, data, sizeof(data), 0);
data[6]='\0';
//setSpeakers(atoi(data));              enabled for demo with laptop and desktop with soundcard only
printf("%s\n", data);
close(clientSock);
}
```

When a connection is accepted it is set to the client socket clientSock and the recv function allows data to be received from it.

setSpeakers is commented out since it will not be demoed in the lab. This function is used on my home setup that I will provide as a video. But setSpeakers is where the magic occurs and changes which speakers are outputting the audio as the user moves around his home.
pid_t ppid = getppid(); //later for forking new process. child processes will handle accepting socket

This is for an upcoming feature regarding expandability. Not used for this demo.