Lab 4: Designing a Jukebox Protocol (Week 1)

Protocol Design Document Due: Monday, Nov 2nd, 11:59 PM

Checkpoint Due: Monday, Nov 9th, 11:59 PM

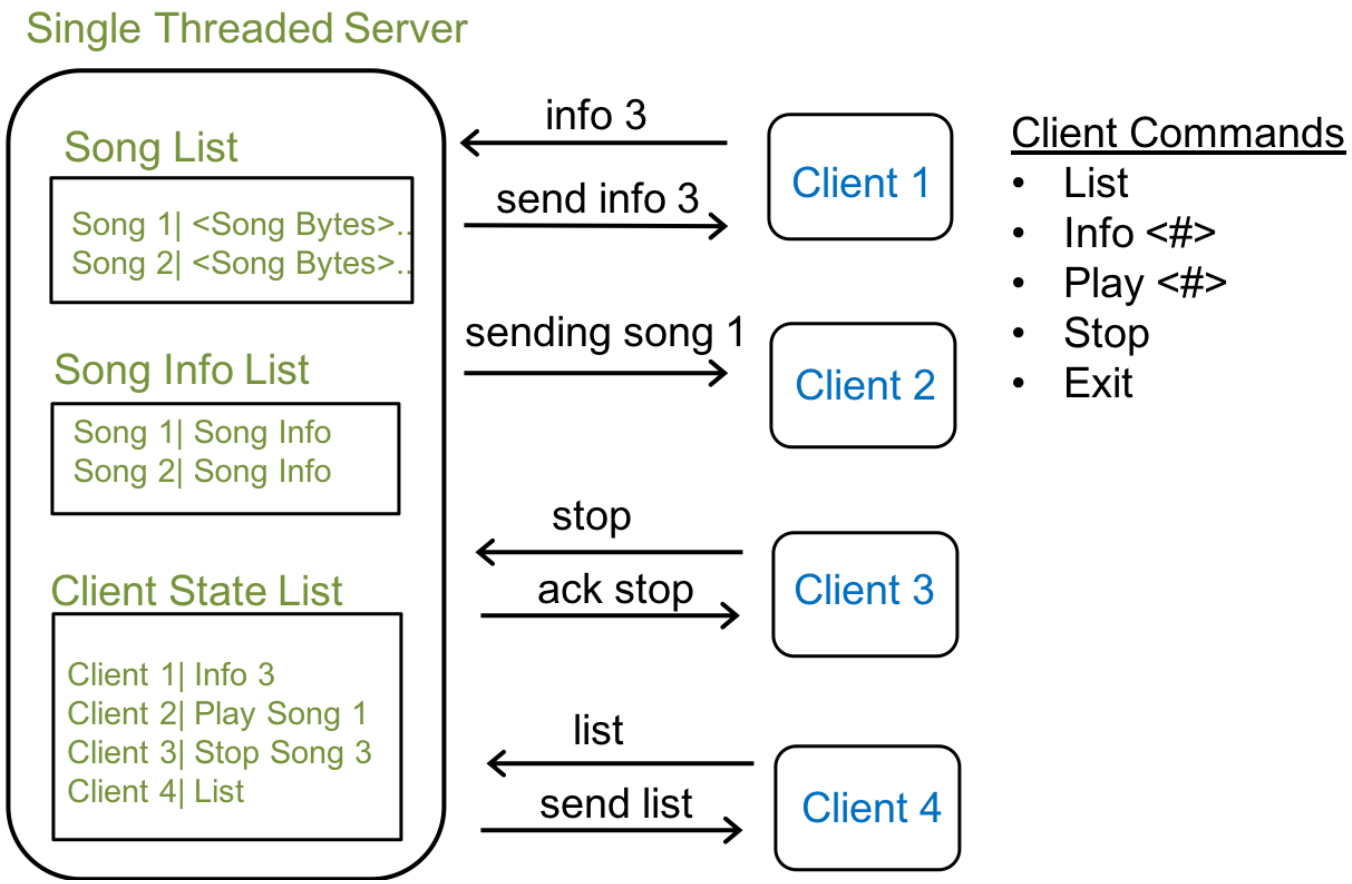
Due: Monday, Nov 16th, 11:59 PM

Let’s look at what we are implementing…

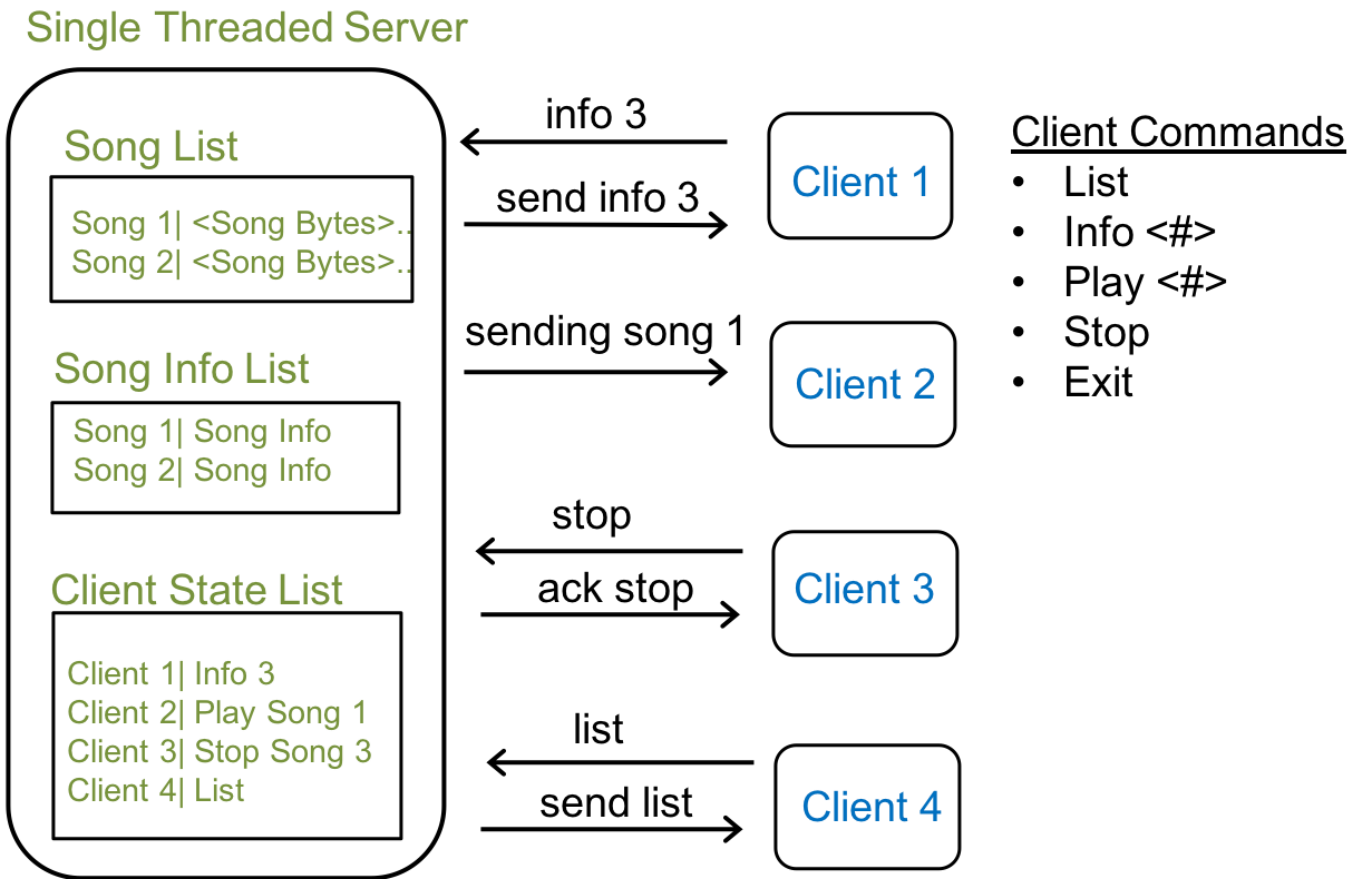
Weekly Deliverables

- Week 1: Protocol Description
- Week 2:
  - Be able to accept clients at the server.
  - Be able to exchange text (list and info) requests and replies between multiple concurrent clients and the server.
  - Be able to exchange music with one client, initially, and then multiple clients.
- Week 3
  - Be able to interleave music and text inputs

This week I’m not expecting you to write any code…



- Single threaded server that can handle multiple concurrent clients using select()
- Multi-threaded client
- Client and server can interleave playing song data with other commands.



- Single threaded server that can handle multiple concurrent clients using select()
- Multi-threaded client
- Client and server can interleave playing song data with other commands.

You cannot send the whole song at once, so you can interleave song data and other types of messages!

On the server side…
Overview of select()

1. Add file descriptors to select read set (sockets we know we want to recv on)
2. Add file descriptors to select write set (sockets we know we want to send on)

On the server side…
Overview of select()

1. Add file descriptors to select read set (sockets we know we want to recv on)
2. Add file descriptors to select write set (sockets we know we want to send on)
3. Select() will block until one of the file descriptors is ready to read or write. Therefore, when select returns, atleast one of the sockets has data in receive buffer (can recv) or there is space in the send buffer (can send). The descriptors that are ready are still in the read set or write set when select returns.

On the server side…
Overview of select()

1. Add file descriptors to select read set (sockets we know we want to recv on)
2. Add file descriptors to select write set (sockets we know we want to send on)
3. Select() will block until one of the file descriptors is ready to read or write. Therefore, when select returns, atleast one of the sockets has data in receive buffer (can recv) or there is space in the send buffer (can send). The descriptors that are ready are still in the read set or write set when select returns.
4. Loop over send set and call send once on each descriptor
5. Loop over read set and call recv once on each descriptor

On the server side…
Overview of select()

1. Add file descriptors to select read set (sockets we know we want to recv on)
2. Add file descriptors to select write set (sockets we know we want to send on)
3. Select() will block until one of the file descriptors is ready to read or write. Therefore, when select returns, atleast one of the sockets has data in receive buffer (can recv) or there is space in the send buffer (can send). The descriptors that are ready are still in the read set or write set when select returns.
4. Loop over send set and call send once on each descriptor
5. Loop over read set and call recv once on each descriptor
6. Continue steps 1-5 in a loop

How to use per-client state + select()

Per-client data stored in a struct

```
clients[socket_fd]
0: connected = 0
1: connected = 0
2: connected = 0
3: connected = 1
4: connected = 1
5: connected = 0
…
9: connected = 1
…
```

How to use per-client state + select()

Per-client data stored in a struct

```
clients[socket_fd]
0: connected = 0
1: connected = 0
2: connected = 0
3: connected = 1
4: connected = 1
5: connected = 0
…
9: connected = 1
…
```

If client is in a state where you want to see if you receive something, add it to the read set

READ SET

3, 4, 6, 9, 10

Is there other state you need to know, to know when to add client to the read set?

How to use per-client state + select()

Per-client data stored in a struct

```
clients[socket_fd]
0: connected = 0
1: connected = 0
2: connected = 0
3: connected = 1
4: connected = 1
5: connected = 0
…
9: connected = 1
…
```

If client is in a state where you want to see if you receive something, add it to the read set

READ SET

3, 4, 6, 9, 10

Is there other state you need to know, to know when to add client to the read set?

If client is in a state where you have something to send, add it to the write set

WRITE SET

3, 4, 10

Is there other state you need to know, to know when to add the client to the write set?

When select() returns, only the sockets that are recv will be in the read set and only sockets able to send will be in the write set.

Per-client data stored in a struct

```
clients[socket_fd]
0: connected = 0
1: connected = 0
2: connected = 0
3: connected = 1
4: connected = 1
5: connected = 0
…
9: connected = 1
…
1023: connected = 0
```

READ SET	WRITE SET
3, 4, 6, 9, 10	3, 4, 10

After calling `select(READ_SET, WRITE_SET)` returns..

READ SET	WRITE SET
9, 10	4, 10

Now we can call recv on sockets 9, 10 and call send on sockets 4, 10

When select() returns, only the sockets that are recv will be in the read set and only sockets able to send will be in the write set.

READ SET	WRITE SET
3, 4, 6, 9, 10	3, 4, 10

After calling select(READ_SET, WRITE_SET) returns..

We are using TCP sockets so we know send and recv may not send or receive everything in one call? Should I call send and recv in a loop here?

READ SET	WRITE SET
9, 10	4, 10

Now we can call recv on sockets 9, 10 and call send on sockets 4, 10

Some questions to answer in protocol design document:

- Format of messages
- Per-client struct data and what it means
- How a client will move from one state to another
- Can your server keep track of how much song data you've sent so far?
- Can your client/server interleave messages and song data?
- Ok, if your design document changes over time!

Signup for a 15-minute time slot to talk to me about your protocol

We’ll talk about starter code and implementation details next week…