Design Document

Data Communications - Assignment 2

Eric Tsang, A00841554, 40

Table of Contents

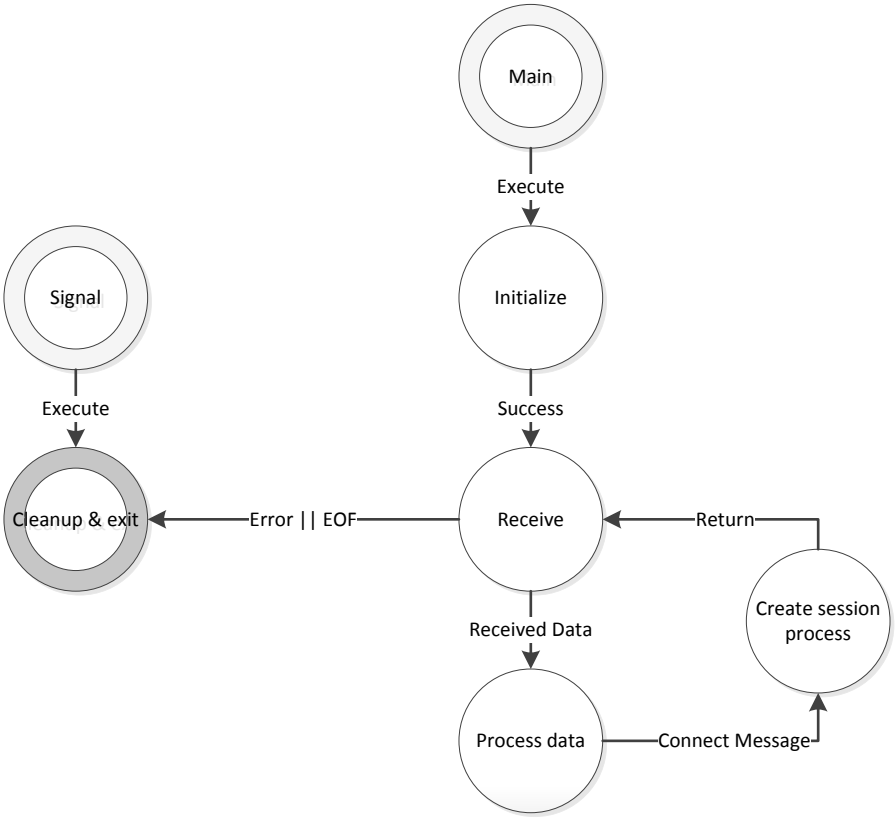
January 10, 2015

State Diagrams

This section contains a finite state machine for each of the processes in the program. These processes include:

- **Server**; creates and destroys the message queue, and forks off new session processes whenever a new client process connects.
- **Session**; verifies the client's request, and then attempts to server the client. if any errors occur, it informs the client, and then exit.
- **Client**; attempts to connect with the server, upon successful connection, it sends its request, and prints out data from the message queue.

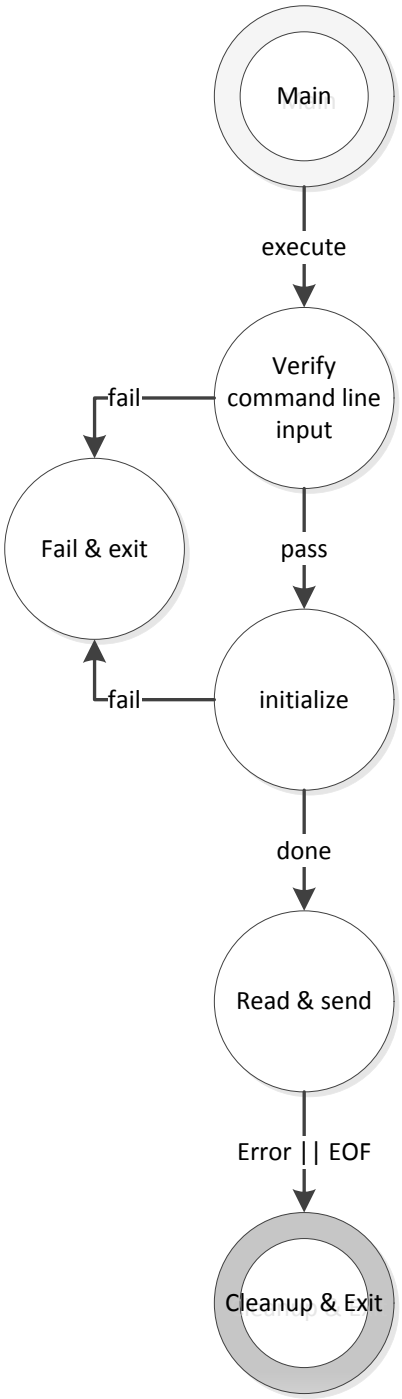
Server



Here is a brief description of each state, and what they do:

- **Initialize**; creates a new message queue, and sets up the signal handler.
- **Receive**; reads from the message queue, expecting to receive connection messages.
- **Process data**; parses the received message, and handles it as needed, or passes the message to a message handler.
- **Handle Connect Message**; a new session process is created to communicate with the client process.
- **Cleanup & exit**; the message queue is removed.
- **Signal**; when the process receives an interrupt, we immediately transition to the cleanup & exit state.

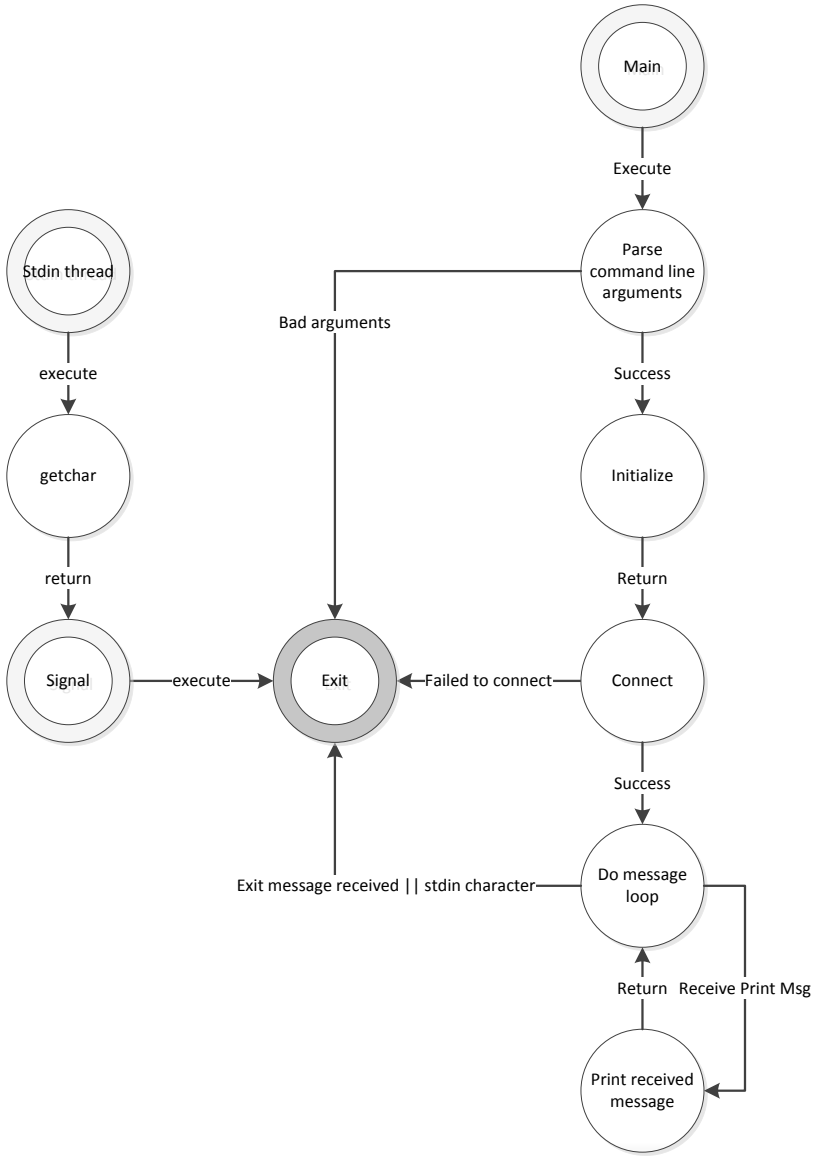
Session



This is the session process's state transition diagram:

- **Verify command line input**; the session is passed the client's input when it begins. in this state, the client's priority parameter is checked. if the parameters are invalid, we transition to the fail & exit state.
- **Initialize**; the client's input is valid. in this state, we set the priority of the session process, get the server message queue, and open the requested file. if the file fails to open, we transition to the fail & exit state.
- **Fail & exit**; here, the session sends a fail message describing the nature of the error to the client through the message queue, and then exits.
- **Read & send**; in this state, the session process reads from the open file, and send it to the client process through the message queue. if an error occurs, or we reach the end of the file, we transition to the cleanup and exit state.
- **Cleanup & exit**; in this state, we close the file descriptor, and exit the process.

Client



This is the client state transition diagram:

- **Parse command line arguments**; in this state, we parse the command line arguments. if it fails, then we move to the exit state, and print a usage message.
- **Initialize**; the message queue is obtained, and the stdin thread is started.
- **Connect**; the connection request structure is constructed, and sent to the server through the message queue. if this fails, we transition to the exit state.
- **Do message loop**; here, the client continuously reads from the message queue, and prints it to the screen. if the client reads a stop message, it transitions to the exit state.
- **Stdin thread**; this thread waits for something from standard in, and once it gets it, it triggers the SIGINT signal.
- **SIGINT signal**; when this signal is caught, we immediately transition to the and exit state.
- **Exit**; in the exit state, we close the file descriptor if it's open, and then exit the process.

Pseudocode

In this section, we have the pseudo code for the 3 main processes.

Server

This section contains the pseudo code for the core procedures that should be in the server module.

Initialize

The following pseudo code should be run before the receive loop procedure begins.

```
1  set signal handler to the cleanup & exit procedure
2  make a system call to make a message queue
```

Receive Loop

The following pseudo code is for dequeueing from the server type messages, parsing and handling them. This cycle should repeat until something goes wrong.

```
1  loop until break...
2      read from the message queue
3      if EOF or an error is encountered or unknown message type
4          break out of the loop
5      if message is a connection request
6          parse connection request
7          start a new session process to deal with client
```

Cleanup & Exit

The following pseudo code should be executed before the process terminates. It should release the system resources that was obtained in the initialize procedure.

```
1  remove the message queue
```

Session

This section contains the pseudo code for the core procedures that should be in the session module.

Initialize

This procedure is used to obtain system resources needed by the read & send procedure.

```
1  set the signal handler to run the cleanup & exit procedure
2  process priority is set
3  existing message queue is obtained
4  client specified file is opened
5  send this process's reference to the client process
6  if any errors occurred in the above steps, exit
```

Read & Send

This procedure reads from the opened file, and sends its content to the client process through the message queue managed by the server process.

```
1  loop until break...
2      read from the opened file (from initialize)
3      send file contents to client process through message queue
4      if EOF was encountered, or error occurs, break out of loop
```

Cleanup & Exit

This procedure releases the resources obtained in the initialize function.

```
1  close the file opened from the initialize procedure
```

Client

This section contains the pseudo code for the core procedures that should be in the client module.

Initialize

This procedure obtains the system resources necessary for the process to function.

```
1  set the signal handler to execute the cleanup & exit procedure
2  get existing message queue
3  begin the exit on character procedure on a thread
4  if anything goes wrong with the above steps, end the process,
   and display the reason for program termination
```

Connect

This procedure sends a connect message to the server, then immediately transitions to the message loop.

```
1  construct the connect message...
2  add the priority, and file path to the connect message
3  send the message to the server process through the message
   queue via type one
```

Message Loop

This procedure reads from the message queue, and handles each message, like WndProc!

```
1  read message from the message queue repeatedly
2  switch on the message type...
3      message type is DATA
4          print it to the screen, including null characters
5      message type is PRINT
6          print the message to the screen
7      message type is STOP_CLIENT
8          break out of the loop
9      message type is SESSION_REFERENCE
10         remember the session's reference
11     message type is unknown
12         execute to cleanup and exit
```

Cleanup & Exit

Informs the session that the client is terminating, so it can terminate as well.

```
1  signal the session that the client is exiting
2  terminate the process
```

Exit on Character

This function is run on a separate thread, in parallel with the process's "main thread".

```
1  wait for a character from the standard input stream
2  execute cleanup & exit
```