CS-4323 Progress Report

Group I

1. Work Distribution

Caleb:
- Part 6, Command history

Collin:
- Part 5, Shell commands in background

Ethan:
- Part 7, Client-server communication

Kazi:
- Part 3 & 4, Shell interface and command execution

2. Group Work Completed

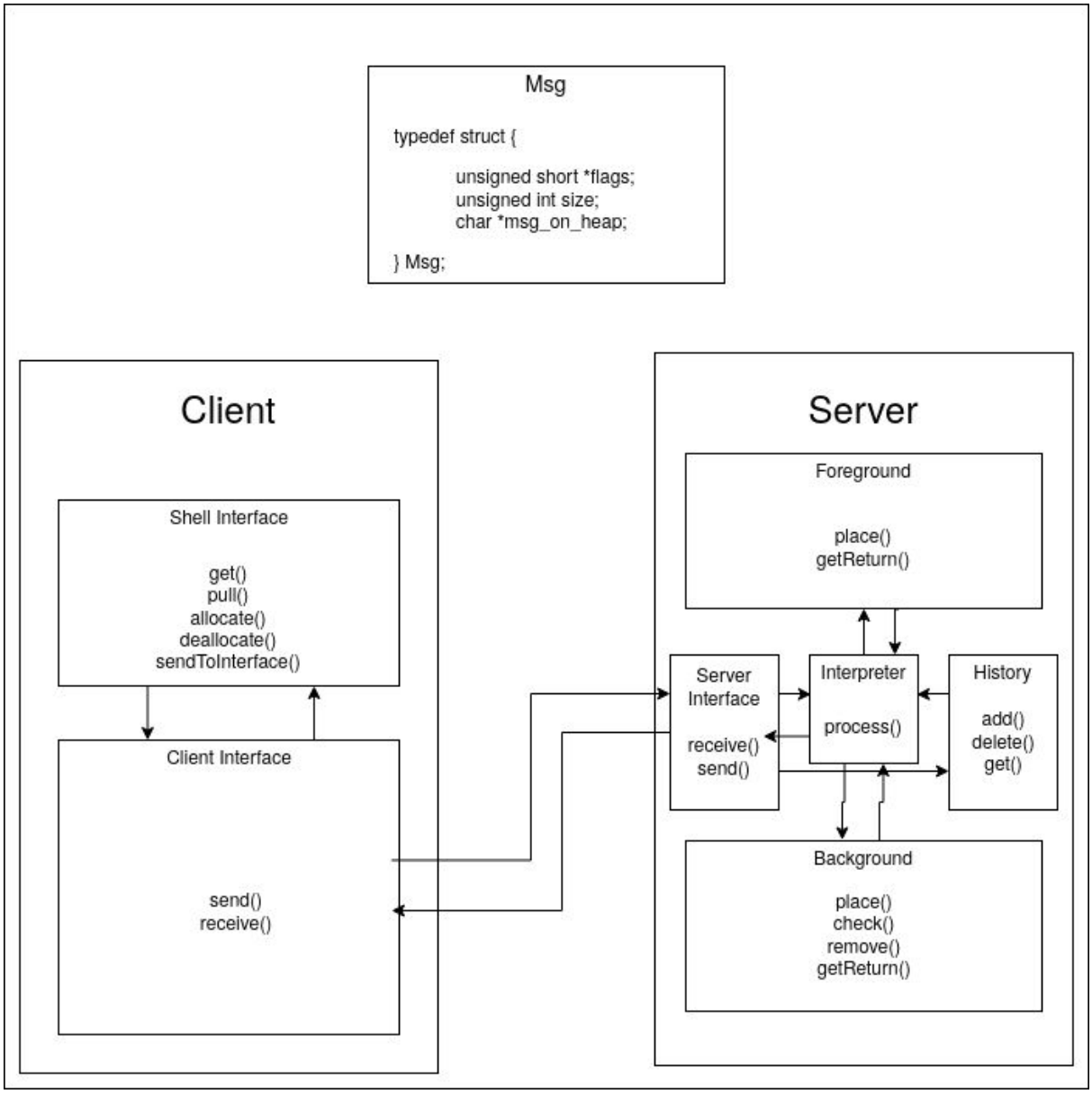

Figure 1: Block diagram created in October 7 meeting.

Between the 3 meetings, we have distributed the workload, scheduled meeting times, created individual work completion dates, and designed the overall hierarchy of the project.

Meeting 1, Ethan, Collin, Caleb, Kazi: Distributed workload, scheduled meetings, individual work deadlines

Meeting 2, Ethan & Collin: Organized the hierarchy of the project, separated all parts into server or client side, determined which section interacts with other sections.

Meeting 3, Ethan, Collin, Kazi: Progress report overview, code implementation

3. Individual Amount Completed

The primary goal of the first week was to determine workload and design the hierarchy and flow of the project. As such, individual work and code is not expected to be delivered until 10/14. Some members may work on code before this point, however this is on an individual basis until it is pushed and accepted into the master branch of our git repository.

Caleb:

Collin:

Designed hierarchy of the project, as well as communication lanes between the sections. Created block diagram, set up GitHub repository, added branch for each member, and organized most meetings.

Ethan:

Basic socket client-server communication designed
Designed hierarchy of the project, as well as communication lanes between the sections

Kazi:

Designed layout for the shell interface, including startup, user input, directory location, and command execution.

4. Work Left

Group:

Caleb:

Design storage for previous commands to act as a reference for history

Collin:

Implement dynamic array to contain background processes. Design method of status check of all processes in array. If exited, pass return back to client-server interface and restructure array as needed. Optionally use doubly linked list to manage processes as this would simplify asynchronous exit handling and reduce time complexity of restructuring list.

Ethan:

Design message passing system to tell the interface what to display.

Implement client-server sockets and design message passing in code.

Kazi:

Multi-pipe command execution.

5. Issues And Solutions

a. Members not showing up
   i. Split work among whoever finishes their part first
   ii. Otherwise, split evenly
b. Unclear parts of assignment
   i. Message professor
c. Unclear how to approach problem individually
   i. Have group discussions to tackle any issues

6. Meeting Notes

October 4, 2020

Attendees:
Collin
Kazi
Ethan
Caleb

Work Distribution:

-Kazi
- part 3 and/or part 4

-Caleb
- part 6

-Ethan
- part 7

-Collin
- part 5

Notes

- Get clarification on part 7
- Working with git server. Branches for each person + main branch
- Try to wrap the code into a function with specific input/output that can be executed from main

---

Schedule

- Try to have individual component complete by October 14 (Wed)
- 1.5 weeks to integrate and fix bugs

Meetings

- Sunday @ 7:30 p.m.
- Wednesday @ 9:30 p.m.

October 7, 2020

Attendees:

Collin
Ethan

Notes

- Utilize Msg struct across code:

```
typedef struct {
   unsigned short type_of_msg; (command, return, etc)
   unsigned int size_of_msg;
   char* msg_allocated_on_heap;
}
```

- 2 processes (client & server)

ON CLIENT PROCESS

- Prompt Interface
  - Reads/writes to stdin/stdout
  - Allocates command string on heap
  - Deallocates returns from heap
  - Passes/receives to Client Interface
- Client Interface
  - Reads/writes to Server Interface (via socket)
  - Passes/receives to Prompt Interface

ON SERVER PROCESS

- Server Interface
  - Reads/writes to Client Interface (via socket)
  - Passes to History
  - Passes/receives to Command Interpreter
- Command Interpreter
  - Processes commands for special characters (pipe, ampersand, etc)
  - Passes/receives to Foreground

- Passes/receives to Background
- Receives form History
- History
- Make a copy of command (potentially make duplicate heap allocation)
- Received from Server Interface
- Passes to Command Interpreter
- Background
- Spawn process to execute command in background
- Passes/receives to Command Interpreter
- Foreground
- Execute command in foreground
- Passes/receives to Command Interpreter

- Any function that will block either process needs to be vetted by team
- Prefer use of flags/polling over blocking calls
- Default path found at /etc/environment. Can be sourced there.

Questions

Bash Shell Commands

- Do we only need to implement what's in part 4?
- How do we interpret the various redirects
- How will we know to run a process in the background?
    - If ampersand -> it isn't in part 4. What are expectations?
- Can we have a couple examples?

Goals for next meeting

- Have definitions for all functions to be called by other people
- Enough detail to create .h file

- Once these definitions are created, they shouldn't be changed w/out group consensus
- This is for the functions used by other people
- Do whatever you want with the functions only used by you

October 11, 2020

Attendees:
Collin
Ethan
Kazi

Notes

- Work on progress report. Due tonight!