

# Project 1 Summary

## Challenges

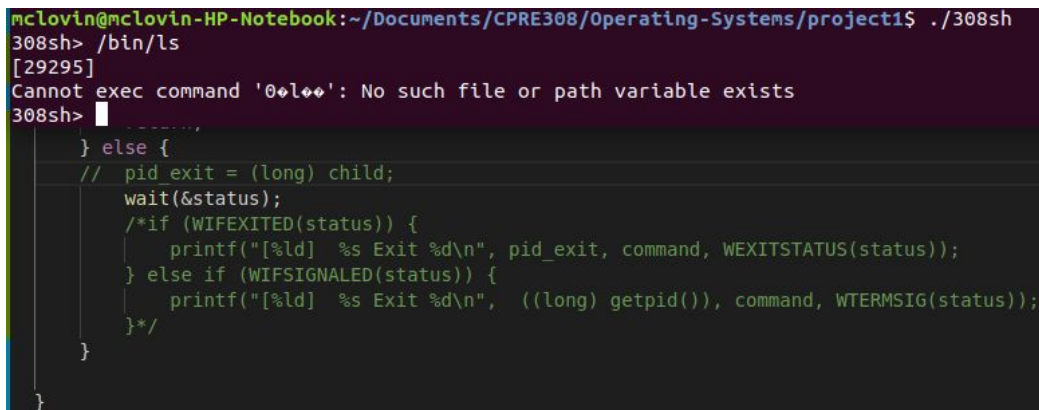
The main challenges I faced while completing this project was remembering how memory allocation works in C. I spent a little extra time reviewing these concepts, as I enjoy the many applications of C and wanted to have nice code that I could come back to and review in the future. The two most difficult things to deal with during this project were the following:

1. Dynamically passing my tokenized strings (in the form of “pointer to pointer to char” which is easier understood as a string array) from the parse function to the main and then when necessary to the executable function.
2. General memory allocation and overwriting of strings in the executable function. E.g. a common unicode character that would print instead of my intended command would be the black diamond with a white question mark character.

To overcome the first challenge, in my parse function I used calloc, to allocate memory for the maximum number of arguments I would need which is in the line below:

- `char** args = calloc(3, (MAXLINE + 1) * sizeof(char));`

To overcome the second issue, in hindsight I realize that incrementally testing the code is what was throwing me off on this one. I based my handling of the parent / child processes when the `execvp()` function is called on the code we did in lab 2. I am still curious as to why this error happens but I have documented it for future references. Notice the difference between when the code is commented vs. when the code is not commented.



```
mclovin@mclovin-HP-Notebook:~/Documents/CPRE308/Operating-Systems/project1$ ./308sh
308sh> /bin/ls
[29295]
Cannot exec command '0?l?': No such file or path variable exists
308sh>
} else {
// pid_exit = (long) child;
wait(&status);
/*if (WIFEXITED(status)) {
    printf("[%ld] %s Exit %d\n", pid_exit, command, WEXITSTATUS(status));
} else if (WIFSIGNALED(status)) {
    printf("[%ld] %s Exit %d\n", ((long) getpid()), command, WTERMSIG(status));
}*/
}
```

Fig 1. Doesn't work

```
mclovin@mclovin-HP-Notebook:~/Documents/CPRE308/Operating-Systems/project1$ ./308sh
308sh> /bin/ls
[29503]
308_Project1.pdf  308sh_backup.c  308sh.o          include.h  strings.c  test.c
308sh             308sh.c         High-Level-Shell-Design.pdf  Makefile   test       test.o
[29503] /bin/ls Exit 0
308sh>

} else {
    pid_exit = (long) child;
    wait(&status);
    if (WIFEXITED(status)) {
        printf("[%ld] %s Exit %d\n", pid_exit, command, WEXITSTATUS(status));
    } else if (WIFSIGNALED(status)) {
        printf("[%ld] %s Exit %d\n", ((long) getpid()), command, WTERMSIG(status));
    }
}
```

Fig 2. working

## Design

I broke the program down into 5 simple helper functions outside of the main. They are getShellName, parse, builtIn, executable, and checkIfBgp. All are self explanatory, checkIfBgp is meant to check for an ampersand and return a boolean true if the processes is a background process and otherwise return false. This logic still needs to be completed.

# High-level Shell Design Summary

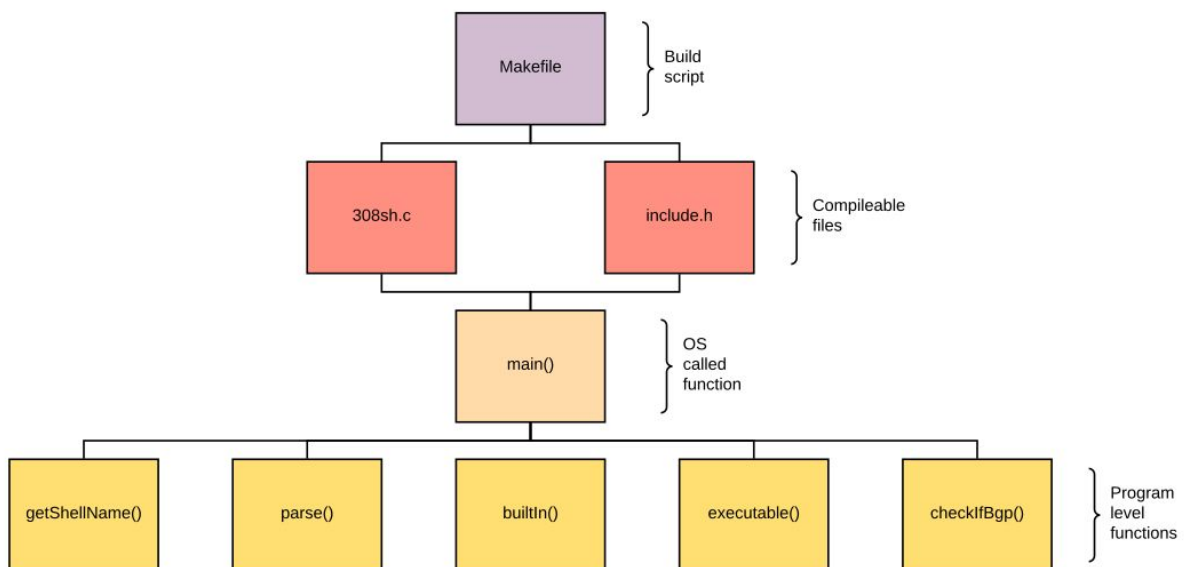


Fig 3. Design Summary

## **Functionality**

I was able to break things up and simplify things by taking the time to parse everything into my string array, which I am very excited about! The basics of this project were very similar to project 2. We use fork to spawn a child process for an executable which allows the main process to keep running. We then use the wait block to wait for the process to terminate just like in 3.8 of lab 2.