Small Shell Project Archive

Team 71
Haoxiang Dai, Ryan Stachura, Ezra Robert Taylor

Abstract

The Small shell is a lightweight Unix-like shell with customized features developed by C language. Small Shell attempts to help you work by simplifying commands and reducing the amount of unnecessary information displayed to the screen. This tool is intended to help those who are unfamiliar with Linux OS's learn the ropes and comprehend the fundamentals of this type of system. On the other hand, this tool is also targeted towards interested programmers who can easily add their own custom commands/instructions to provide greater, or more specific, functionality.

Table of Contents

Introduction

- Who requested it?

  Given our uncommon circumstance, we approached Mr. Appasamy with the idea for this project.

- Why was it requested?

  We requested this project as it served as the best middle ground between our collective knowledge base. We felt most comfortable with this over other projects that required background knowledge and/or unfamiliar programming languages.

- What is its importance?

  It serves to show how a program with many different capabilities can be constructed as a collective. Since we can conceptually split the large-scale program into smaller sections which we can work on individually, but will all be able to communicate with each other and the program itself..

- Who was/were your project partner(s)?

  Mr. Bharat Appasamy is our project partner.

- Who are the members of your team?

  All three of us, Haoxiang Dai, Ryan Stachura, and Ezra Robert Taylor, were of equal importance to the overall team. Each of us contributed code to the shell. Most of the time was spent discussing how we wanted to approach a goal.

- What were their roles?

  Each of us are Software Developers.

- What was the role of the project partner(s)? (I.e., did they supervise only, or did they participate in doing development)

  Mr.Bharat: supervisor, consultant.

- How did the changes in on-campus courses due to Covid affect your deliverables (e.g., you were limited in how much time you could spend working with essential hardware or in an on-campus lab)?

  Luckily for us, our project of only three months didn't require any specific hardware outside of our personal computers and the internet. Unfortunately, we believe that physical meetups produce a higher work flow and greater success as opposed to virtually. Covid limited us to the virtual world in order to cooperate on this project.

- How do you recommend the next team use this final documentation to pick up where you left off? (Refer readers to the release notes—that's where you'll write up the remaining work to do).

  We gladly recommend anybody who wishes to add more features to Small Shell to read through our documentation so they know how it functions and can properly contribute by creating additional features.

Copy of the Vision & Scope (Milestone 2) Paper

Background

The programming language C has been around since 1972 and is used across the Earth to date. Understanding one of the most fundamental languages inside and out can only help. On top of this, Unix based systems are also an extremely important part of today's society. Therefore, we will be combining them in such a way that we can show our understanding of each.

Vision Statement

This is a very time sensitive project with specific goals. Due to the time constraints, we will not be adding extra features unless all goals have been met. There is no extra expense needed nor any additional constraints that could hold us back. Each member of the team will do their job in order to complete this project on time.

Success Measures

We will be measuring our success on not only time spent each week on the project but also goals achieved in a timely manner. Each person on the team will be assigned their own tasks in order for us as a team to complete a goal. Each team member will be measured by how much time they spent in one week specifically on writing code. These time measurements should come out to around at least eight hours each week. With these measurements of everyone on the team and the complete progress of the project, we can use this information to confirm that we will be able to reach our final goal on time.

Prioritized Project Constraints

Our main constraint with this project is time, since we are completing a 3-month project. Because of this, we will prioritize the essential features of our project such as changing directories, using foreground and background processes, interrupt signal processing, input/output redirection, etc. In order to release by the end of this term, our resources have been limited to C and Linux-based systems/servers.

Stakeholders

Operating system learner:

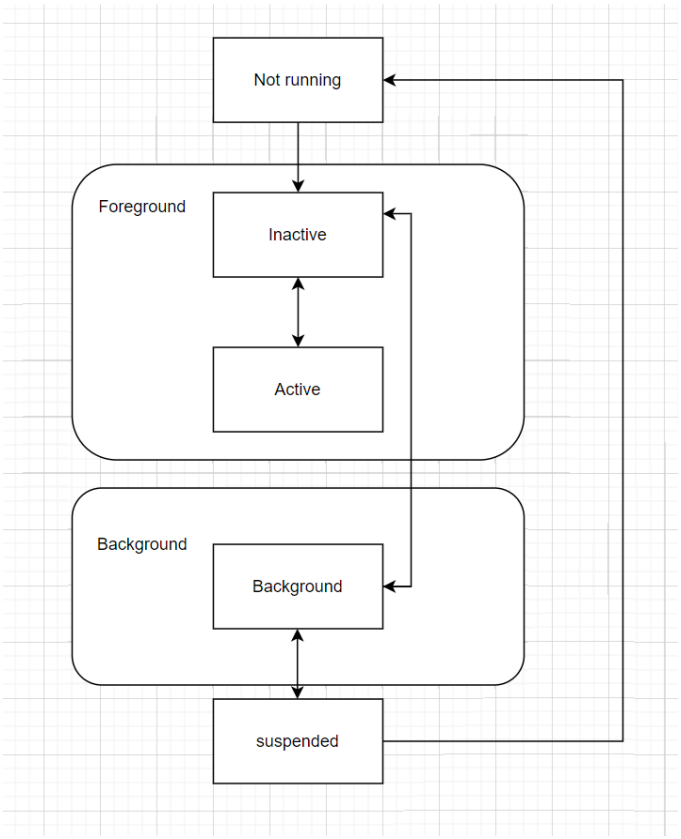
This project provides the most basic features of an operating system. The operating system learner can easily pick up those features and designs.

Risk

Risk	Likelihood	Impact	Mitigation Strategy	Early Detection	Consequence
All the paperwork and documentation needed for the project not getting done. Other teams have had three terms to complete, as we only have one.	Unlikely	Medium	To mitigate this, we have a plan of action to complete this project as fast as possible. When this project is complete in this way, it will give us some extra time in the end to complete all necessary documentation needed for project and expo demonstration.	Weekly plan updates result in an iteration plan that goes beyond the deadline.	Should the mitigation strategy fail to prevent/avoid the risk, then our project will not look as nice without documentation.

Scope

1.1 Process Flow



1.2 User Stories

As a user, I want the system to have a cd command that allows me to change my current position to anywhere I want.

As a user, I want to be able to check the status of my current programs by using a status command.

As a user, I want the system to have the most basic commands: ls, pwd, kill.

As a user, I want to be able to have a foreground only mode.

As a user, I want to be able to modify the command.

As a user, I want to be able to see a different color of text in my system.

1.2.1 Epics

As an operating system learner, I want this system to clearly show the information with a good format.
As a designer, I want to implement the features by using least API
As a programmer, I want all of the code be well-commented
As a programmer, I want the code can be run on multiplatform

1.3 Iteration Plan and Estimate

Each week we will be having a sprint meeting to talk about what needs to be done this week and if every has put in their time from the previous week. Each person has their own goal for the end result of the project. As long as everyone puts in their time each week, there should be no problem finishing this project in time for the expo.

1.4 Solution Architecture

C was chosen as the language to implement this project in because of how unrestricted it is. Unlike other common languages, C will do just about anything you tell it to do. Of course this can be a bad thing too, but part of this project is to show our understanding of C -- to show what we have learned. Linux is the operating system of choice because of how well it interacts with C. On top of this, one of the goals is to create entire processes, whether it be in the foreground or background. C provides us with the *fork()* function which allows us to manage these processes systematically.

Copy of your Retrospective (Winter Term)

*Ours are not from Winter term given our unique circumstance.

Haoxiang:

Sprint 1:

What did I work on in this sprint?

I wrote the documentation for our project to specify the features and functions. I held a group meeting with Ryan and Taylor to brainstorm more features, and our sprint plans. I implemented the github repo with code review, and teached my teammates how to use it. I implemented the starter code and Readme.

What went well?

We have more thoughts about the file lock features, command modification. Also, I successfully get in touch with our group members, I think it’s an improvement of our current situation.

What didn't go well/Are there any blockers?

I think we can do better on communication. We should have more conversations in groups.

What are we doing to remove the blockers/resolve issues?

I have tried to call Ryan and Taylor in person instead of text to discuss the project, and it did improve the communication problem a bit.

What will I be working on for the next sprint?

I will implement the basic commands with Taylor of our shell. Also, design the mechanism for file lock. I will also connect with Ryan and Taylor about the expo page.

Sprint2:

What did I work on in this sprint?

I have updated the sprint retrospective documentation. I have updated the features design and scope documentation with file lock mechanism. I have written the description for our expo page, and made the thumbnail of our smallshell. I have updated Github to wipe private information like group71, Oregon state university. I have implemented some basic commands on GitHub.

What went well?

Our project is on the right track now, we have implemented something real, and finished the expo page this week. Ryan and I worked on the expo page together and Taylor worked on the code of basic comments. We worked closer and better than previous weeks.

What didn't go well/Are there any blockers?

We need to start the project archive as soon as possible since we have much less time than other groups. Our project does not have a high level design document, and I will try to add.

What are we doing to remove the blockers/resolve issues?

Do it ASAP

What will I be working on for the next sprint?

Implement the file lock feature. Add a high level design document. Try to meet with people to discuss the project archive document.

For sprint 3:

Sprint Retrospective/report from each team member to be sent every week (up to 1 page):

What did I work on in this sprint?

I finished up the high level design for the project and divided the functions into different feature section.

I implemented the basic commands with Taylor.

I submitted my lock&unlock feature design and started to implement it.

What went well?

We are on the right track for the implementation.

What didn't go well/Are there any blockers?

Ryan was disconnected with us for a while because of his illness, and we have a hard time on discussion.

What are we doing to remove the blockers/resolve issues?

I will shoot Ryan messages in person, and try to talk to him about the project, reduce his workload.

What will I be working on for the next sprint?

I will finish up the lock/unlock feature, and changeCmdName feature.

I will start to work on the user_guide.

I will update the previous documentation with teammates.

I will finish up the main loop for our shell

I will write up the test script for all features.

For sprint 4:

Sprint Retrospective/report from each team member to be sent every week (up to 1 page):

What did I work on in this sprint?

I finished up the high-level design, lock&unlock design documentation.

I finished the test script and tested all features.

I finished the changeCmdName feature, and the lock&unlock is still buggy.

I merged the newest branch to github, and discuss the code with Taylor.

I drafted out the user guide, and discussed it with Taylor.

What went well?

We finished up most of the components and the code works well.

What didn't go well/Are there any blockers?

The time is too short for us. We need to do all the things quickly before the regular deadline of CS463.

What are we doing to remove the blockers/resolve issues?

We will increase our work time per week.

What will I be working on for the next sprint?

I will finish up the user guide, pack all things needed, and send them to Bharat, and Kirsten.

I will debug my lock&unlock feature.

I will organize our repo and make it clean.

For sprint 5:

Sprint Retrospective/report from each team member to be sent every week (up to 1 page):

What did I work on in this sprint?

I participated on the writing of our project archive for part 5, and 9.

I did the debug my lock&unlock feature.

I re-organize our repo and make it clean.

What went well?

We finished up most of the components and the project archive looks good. Everyone is showed up on writing even it’s final week.

What didn't go well/Are there any blockers?

The time is too short for us, and I have other finals to learn, which makes hard to debugging the new features.

What are we doing to remove the blockers/resolve issues?

We will increase my work time per week.

What will I be working on for the next sprint?

This is the last sprint!!

Ryan Stachura:

- What did I work on in this sprint?
  I started work on the main C file that will be used to compile our shell. So far, I've added the main outline of operations. This includes the general structure and flow of events. There also exists a check for background processes that have terminated. I have also worked on the sprints.
- What went well?
  Setting up the code wasn't terribly demanding. We've also started communicating much more effectively with each other, i.e. via email, discord, etc.
- What didn't go well/Are there any blockers?
  Initially, our goals were a little hazy but we have focused them since. We also struggled with communication prior to sprint 2.
- What are we doing to remove the blockers/resolve issues?
  We have increased our communication frequency by having our own meeting times on a weekly basis. We also expanded our means of communication (again, email/slack/discord/etc).
- What will I be working on for the next sprint?
  For the next sprint, I will be focusing on adding a specific element/section to our code. This section hasn't been decided upon yet, but each of us will be completing a section of our own. This includes, but is not limited to, the change-directory command, status command, or possibly the encryption/decryption commands.

Screenshot/Image from Virtual Expo page

Project Description:
The Small shell is a lightweight Unix-like shell with customized features developed by C language. In reality, people used operating systems such as Linux, Windows, macOS to communicate with their hardware devices, and the small shell is an operating system kernel with the least features, which is suitable for low-performance devices.
In the small shell, the system calls, the mechanisms by which a user program asks the OS to perform services are supported, including process control, file management, information maintenance, communication, protection. It contains the basic commands for users to communicate with their devices as Unix does. The supported commands are: pwd(print out the working directory), cd(move users current working directory to a different one), ls(Display the files in a given directory), mkdir(create a directly), rmdir(remove a directly), rm(Remove files or directories if used recursively), mv(Move or rename files and directories), cat(Dump the character data from given file to the terminal).

Beyond the basic Unix commands, the small shell allows users to modify their commands by using chCommand(change command) to modify their commands. For example, if you feel mkdir is obscure and hard to memorize, you can use chCommand to change command mkdir to createDIR or anything that seems right for you.

The small shell has a file lock feature to ensure the security of your files. The lock command allows users to lock their files with a password. This command will encrypt the target file and promote a password from the user, and change the file suffix to .lock. After applying the lock command to a target file, users can still open/cat this file but get encrypted content. Only when the user uses the unlock command to this file and enters the correct password, the file will be decrypted and switched back to the original suffix.

Project Team Member(s):
Ryan Stachura
Haoxiang Dai
College of Engineering Unit(s):
Electrical Engineering and Computer Science
Project Key Words:
Unix
Shell
Command prompt

Project Documentation

Index of this user project documentation:
- **User guide**
- **High-level design**
- **Lock/unlock design**

-User guide:

Introduction:

GithubLink:
https://github.com/DHX98/small-shell

The Small shell is a lightweight Unix-like shell with customized features developed by C language. In reality, people used operating systems such as Linux, Windows, macOS to communicate with their hardware devices, and the small shell is an operating system kernel with the least features, which is suitable for low-performance devices.

In the small shell, the system calls, the mechanisms by which a user program asks the OS to perform services are supported, including process control, file management, information maintenance, communication, protection. It contains the basic commands for users to communicate with their devices as Unix does. The supported commands are:

- pwd(print out the working directory),
- cd(move users current working directory to a different one),
- clear(to clear the screen),
- ls(Display the files in a given directory),
- mkdir(create a directly), rmdir(remove a directly),
- rm(Remove files or directories if used recursively),
- mv(Move or rename files and directories),

- cat(Dump the character data from a given file to the terminal),
- > (dump data to any certain file),
- status(check the status of the shell),
- kill(to kill the program
- chCommand(modify the command name),
- lock (encrypt file and promote password from user),
- unlock (decrypt file and promote password from user).
- exit(quit the shell)

Beyond the basic Unix commands, the small shell allows users to modify their commands by using chCommand(change command) to modify their commands. For example, if you feel mkdir is obscure and hard to memorize, you can use chCommand to change command mkdir to createDIR or anything that seems right for you.

chCommand(modify the command name)

The small shell has a file lock feature to ensure the security of your files. The lock command allows users to lock their files with a password. This command will encrypt the target file and promote a password from the user, and change the file suffix to .lock. After applying the lock command to a target file, users can still open/cat this file but get encrypted content. Only when the user uses the unlock command to this file and enters the correct password, the file will be decrypted and switched back to the original suffix.

-WorkFlows:

How to run and environment:

$make

Run make on any linux environment with makeFile(version higher than 3.82) and gcc (version higher than 4.8.5)

```
GNU Make 3.82
Built for x86_64-redhat-linux-gnu
Copyright (C) 2010  Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

```
gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-44)
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

$./smallShell

```
[daihao@flip1 ~/capstone$] make
gcc -o small_shell small_shell.c
[daihao@flip1 ~/capstone$] ./small_shell
:
```

1. pwd (print out the working directory),

```
: pwd
/nfs/stak/users/daihao/capstone
```

2. cd (move users current working directory to a different one),

```
: cd capstone
: ls
a.out  encryptedFile  junk  junk2  makefile  RSA  rsa.c  small_shell  small_shell.c  testScript
: cd ..
: ls
112650    cs160f18                   CS444_final          jossss         testdir13023   testdir238615
134442    CS162                      CS444_lab2           lab-1-DHX98    testdir13642   testdir241753
139033    CS261                      CS445                l_k            testdir15328   testdir25658
22222     CS290                      CS475                mail           testdir214136  testdir31493
32886     CS290-Server-Side-Examples final-project-DHX98  newCS444       testdir214484  testdir31838
72762     CS325                      fker                 node_modules   testdir21520   testdir5237
bin       CS331                      hpeesof              npm-debug.log  testdir230438  tmp
box       CS340                      hw2                  perl5          testdir232664  tnl
capstone  CS344                      jiujiuwo             public_html    testdir234229  week1
CS160     CS444                      jos                  testdir$$      testdir236859  Windows.Documents
```

3. ls (Display the files in a given directory),

```
: cd capstone
: ls
a.out  encryptedFile  junk  junk2  makefile  RSA  rsa.c  small_shell  small_shell.c  testScript
:
```

4. rm (Remove files or directories if used recursively),

```
: ls
a.out  encryptedFile  junk2  makefile  RSA  rsa.c  small_shell  small_shell.c  testScript
: rm junk2
: ls
a.out  encryptedFile  makefile  RSA  rsa.c  small_shell  small_shell.c  testScript
:
```

5. mv (Move or rename files and directories),

```
: mv a.out RSA
: ls
encryptedFile  makefile  RSA  rsa.c  small_shell  small_shell.c  testScript
: cd RSA
: ls
a.out
:
```

6. cat (Dump the character data from a given file to the terminal)

```
: cat encryptedFile
a.out
2333encryptedFile
2333junk
2333junk2
2333p2testscript
2333RSA
2333rsa.c
2333smallsh
2333smallsh.c
2333ranranranranranranranranran2333:
```

7. > (dump data to any certain file),

```
: ls > junk
: ls
encryptedFile  junk  makefile  RSA  rsa.c  small_shell  small_shell.c  testScript
: cat junk
encryptedFile
junk
makefile
RSA
rsa.c
small_shell
small_shell.c
testScript
:
```

8. status (check the status of the shell),

```
: status
exit value 0
:
```

9. kill (to kill the program

```
: kill -SIGTSTP

Usage:
 kill [options] <pid|name> [...]

Options:
 -a, --all              do not restrict the name-to-pid conversion to processes
                        with the same uid as the present process
 -s, --signal <sig>     send specified signal
 -q, --queue <sig>      use sigqueue(2) rather than kill(2)
 -p, --pid              print pids without signaling them
 -l, --list [=<signal>] list signal names, or convert one to a name
 -L, --table            list signal names and numbers

 -h, --help     display this help and exit
 -V, --version  output version information and exit

For more details see kill(1).
:
```

10. changeCmdName (modify the command name),

```
: changeCmdName cdd xx
can not find this cmd: 'cdd'
: changeCmdName cd changeDir
your command: 'cd' was successfully changed to 'changeDir'
your command 'cd' will not work anymore
: ls
encryptedFile  junk  makefile  RSA  rsa.c  small_shell  small_shell.c  testScript
: cd ..
: ls
encryptedFile  junk  makefile  RSA  rsa.c  small_shell  small_shell.c  testScript
: changeDir ..
: ls
112650    cs160f18                  CS444_final          jossss         testdir13023   testdir238615
134442    CS162                     CS444_lab2           lab-1-DHX98    testdir13642   testdir241753
139033    CS261                     CS445                l_k            testdir15328   testdir25658
22222     CS290                     CS475                mail           testdir214136  testdir31493
32886     CS290-Server-Side-Examples  final-project-DHX98  newCS444       testdir214484  testdir31838
72762     CS325                     fker                 node_modules   testdir21520   testdir5237
bin       CS331                     hpeesof              npm-debug.log  testdir230438  tmp
box       CS340                     hw2                  perl5          testdir232664  tnl
capstone  CS344                     jiujiuwo             public_html    testdir234229  week1
CS160     CS444                     jos                  testdir$$      testdir236859  Windows.Documents
:
```

11. lock (encrypt file and promote password from user),

```
: lock junk
encryptedFile
junk
makefile
RSA
rsa.c
small_shell
small_shell.c
testScript
your file: (null) is locked!
 -use unlock to unlock it
:
```

12. unlock (decrypt file and promote password from user).

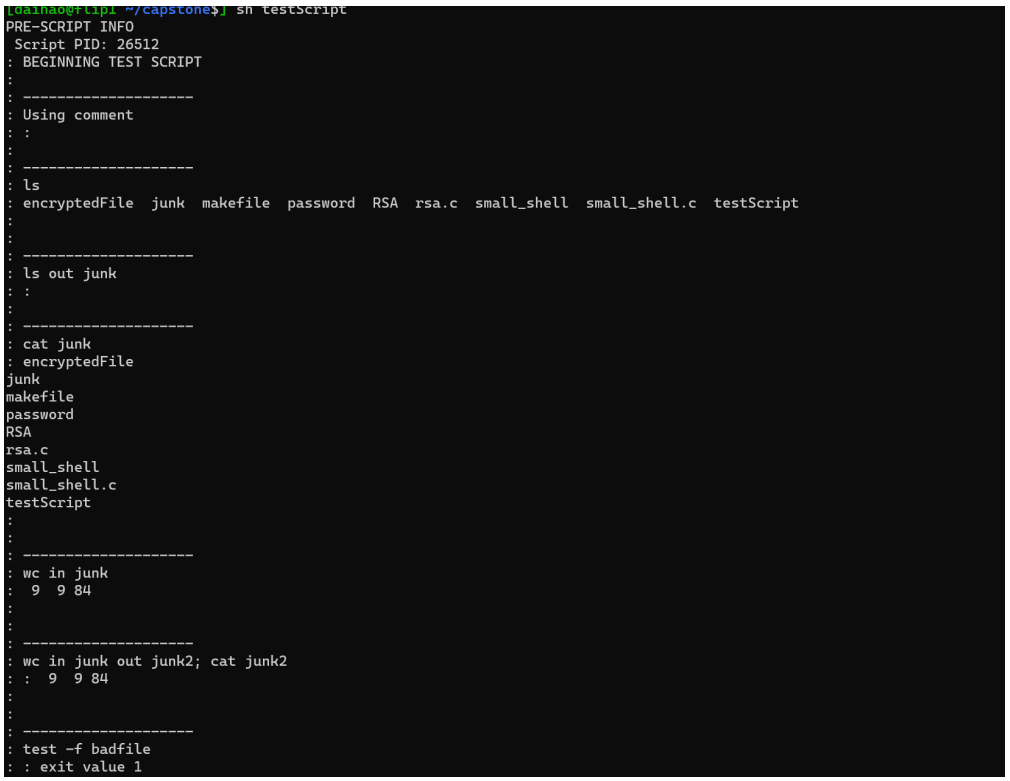
```
: unlock wrongPassword
password wrong, please try again
: unlock lol
password correct.
Enter your fileName:
junk
file junk
 is unlocked
:
```

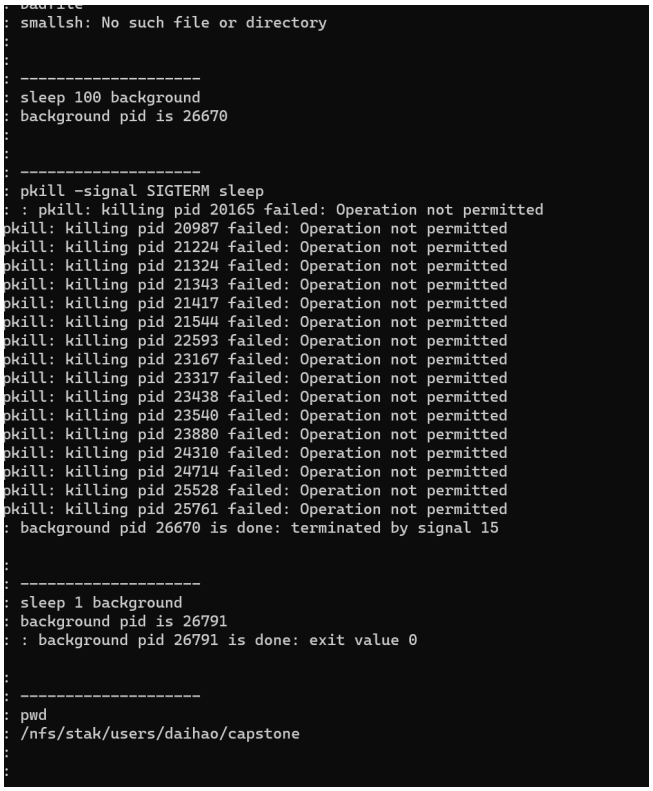
13. exit (quit the shell)

```
: exit
[daihao@flip1 ~/capstone$]
```

-Test

To to the test for all commands:
$sh testScript





If you run the shell on your own device, the kill will disappear.

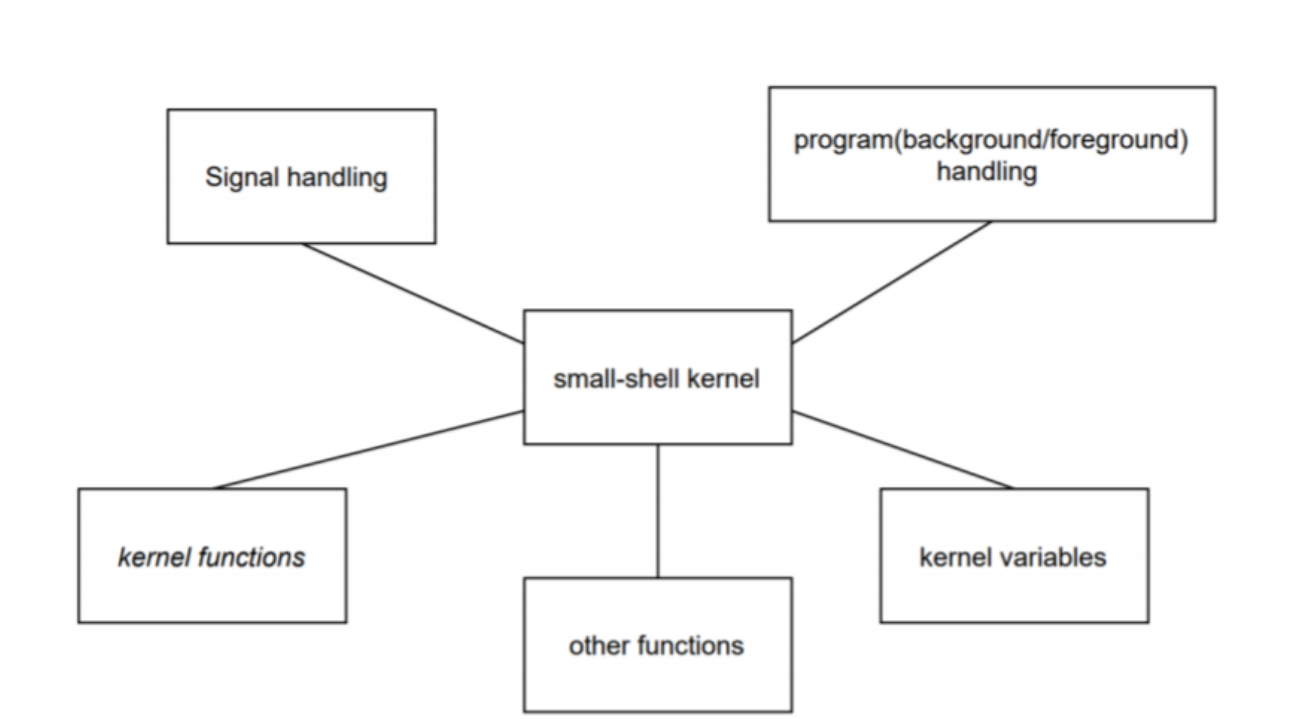
```
: background pid 26670 is done: terminated by signal 15

:
: ----------
: sleep 1 background
: background pid is 26791
: : background pid 26791 is done: exit value 0

:
: ----------
: pwd
: /nfs/stak/users/daihao/capstone
:
:
: ----------
: cd
: :
:
: ----------
: pwd
: /nfs/stak/users/daihao
:
:
: ----------
: mkdir testdir26513
: :
:
: ----------
: cd testdir26513
: :
:
: ----------
: pwd
: /nfs/stak/users/daihao/testdir26513
: ----------
: Entering foreground-only mode (& is now ignored)
: : Fri May 28 23:46:24 PDT 2021
: : Fri May 28 23:46:29 PDT 2021
```

Structure:

High level design for Small-shell The kernel architecture:



Kernel consisted by the features shown below:

Signal handling functions
void catch_TSTP(int signo);
signal handler for SIGTSTP, toggle fg when called

.

program(bg/fg) handling functions
void sh_kill_zombies(); void sh_add_bgs(long pid);
check and clear zombies add background process pid to array

kernel functions
int sh_cd(char **args) int sh_exit(char **args) int sh_status(char **args) int sh_changeCmdName(char **args) int sh_lock(char **args) int sh_unlock(char **args)
change dir command build in exit command build in status command build in changeCmdName command buildin lock command buildin unlock command buildin

kernel variables
pid_t sh_pid int num_bgs = 0 int ter = 0 int num_bgs = 0 int ter = 0 int fg = 0 long sh_bgs[MAX_BGS];
parent process pid number of background process check if process terminated number of background process check if process terminated store background process pid foreground mode switch store background process pid

other functions:

void sh_loop();
char *sh_read_line();
char **sh_get_args(char *line);
int sh_lanuch(char **args);

Running the kernel loop Handling user input by line Handling user input sperate by space Trigger for all build commands

---

List of Recommended Technical Resources for Learning More

- What websites were helpful? (Listed in order of helpfulness.)
  - Man7.org/Linux
  - Crypto-It.net

- Were there any people on campus who were really helpful?

  Justin Goins, an instructor for Operating Systems I and II, was a very helpful resource when trying to understand how to program a similar system, especially in C.

Conclusions and Reflections

- What technical information did you learn?

  Ryan Stachura: Throughout this term, I've learned how to use GitHub in a more technical way by using a repository as a team. This mainly includes but isn't limited to branching and approving push/pull requests.

  Haoxiang Dai: In this tough term, I've learned to implement the RSA encryption, and signal handling in the operating system. I've also learned how to manage a team and set up the workloads. I also learned how to make a thumbnail and a good-looking design chart.

  Ezra Robert Taylor: I learned some more technical skills with writing in C and some new libraries.

- What non-technical information did you learn?

  Ryan Stachura: Some non-technical information I learned includes working as a team, communication skills, how to deal with unexpected circumstances, etc.

  Haoxiang Dai: I have learned the workflow as a team leader, to actively communicate with teammates, and how to do the oral report for my code in meetings. I've also improved my technical writing.

  Ezra Robert Taylor: I learned how to properly work as a team with sprint meetings and learned about different work methods with a team

- What have you learned about project work?

  Ryan Stachura: Project work can have many unexpected issues that arise. Whether it be a technical issue or a group work issue, there are always unforeseen circumstances. It is best to communicate clearly and concisely what the issue is, how we might go about solving it, and when it needs to be corrected.

  Haoxiang Dai: I've learned how to manage the team and balance the actual work and demands. I learned the importance of attending meetings and responding promptly. Since the term is special because of covid, we have to meet online and could not get an efficient response between team members sometime.

  Ezra Robert Taylor: Something I learned about project work is that there is a lot more design and communication needed to make it all come together.

- What have you learned about project management?

  Ryan Stachura: There are many different aspects to consider when attempting a project. Management of a project is somewhat difficult, especially with such a short timeframe.

  Haoxiang Dai: I've learned how to lead the team and set up the workflow, workload. I've also learned how to lead the discussions about our project.

  Ezra Robert Taylor: I learned that you have to actually manage the project sometimes, you can't just let things roll and hope it works out.

- What have you learned about working in teams?

  Ryan Stachura: Teamwork needs to be managed carefully and meticulously. It is extremely helpful to split the work up amongst all the team members. Most importantly, communication is key; frequent communication keeps everyone in the loop and up to date.

  Haoxiang Dai: I've learned how to communicate with teammates and project partners. We need to have efficient communication to make the team successful.

  Ezra Robert Taylor: I've learned that when working with teams, if everyone is motivated and committed to the project then your team should work well together

- If you could do it all over, what would you do differently?

  Ryan Stachura: I would develop more connections within the class and within other teams so that I could have more perspectives on the scope of what it takes to complete a project.

  Haoxiang Dai: I would start my work early each sprint, so that I can better cooperate with my teammates.I would also like to set up some ground rules such as response within 2hours on weekdays.

  Ezra Robert Taylor: If I could do it again, I wish I could have taken less classes so that I had more time for this project.

Future Direction

As a user, I want non coding people to be able to understand the documentation.
As a user, I want this to be an open source project
As a user, I want people to be able to customize the current functions
As a user, I want to have more features on this shell, such as changing text colour.
As a user, I want this shell to be able to run in multiple environments, not only on Linux.
As a user, I want to have an executable version instead of source code and makefile.
As a programmer, I want to have a docker setting to run and test the code in different environments.

Appendices

1. Essential Code Listing:
   This is the skeleton implementation small-shell:

```
void sh_loop();
char *sh_read_line();
char **sh_get_args(char *line);
int sh_execute(char **args);
int sh_lanuch(char **args);
int sh_cd(char **args);
int sh_exit(char **args);
int sh_status(char **args);
int sh_changeCmdName(char **args);
int sh_lock(char **args);
int sh_unlock(char **args);

void sh_add_bgs(long pid);  //add background process pid to array
```

```
void sh_kill_zombies();     //check and clear zombies
void catch_TSTP(int signo); //signal handler for SIGTSTP, toggle fg when called

char *buildin_args[] = { "cd", "exit", "status", "changeCmdName", "lock","unlock"};
int (*buildin_func[])(char **) = {&sh_cd, &sh_exit, &sh_status, &sh_changeCmdName,
&sh_lock,&sh_unlock};
pid_t sh_pid;              //parent process pid
int num_bgs = 0;           //number of background process
int ter = 0;               //check if process terminated
int fg = 0;                //foreground mode
long sh_bgs[MAX_BGS];      //store background process pid
```

If you want to add more features,
you can add the command name into here.

```
char *buildin_args[] = { "cd", "exit", "status", "changeCmdName", "lock","unlock"};
```

then, you might need to add the feature function into this array:

```
int (*buildin_func[])(char **) = {&sh_cd, &sh_exit, &sh_status, &sh_changeCmdName,
&sh_lock,&sh_unlock};
```

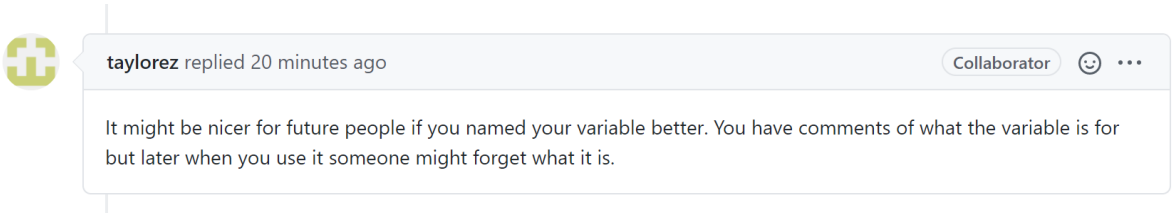
If you want to maintain the same structure of our original implementation. Just add your separate functions into this function.h as below:

C functions.h

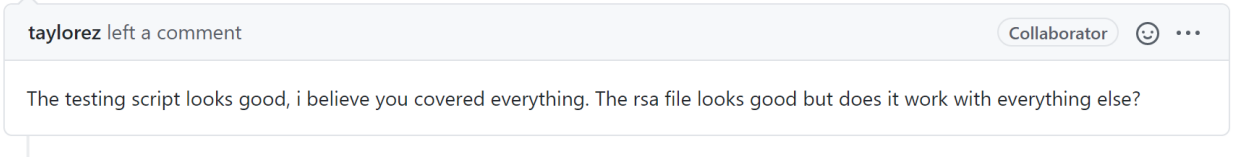
```
15
16  void call_cd(char** array, int count);
17  void call_status(char** array, int count);
18  void call_mkdir(char** array, int count);
19  void call_wc(char** array, int count);
20  void call_cat(char** array, int count);
21  void call_comment(char** array, int count);
22  void call_ls(char** array, int count);
23  void call_exit();
24
25  #endif
26
```

If you want to change the encryption algorithm, you can change the code from rsa.c

Critisism:



Already changed some variable names and added more comments on the latest commit.



Already updated the test script on the latest commit.