CAOS Assignment 2

#include Files:

- #include <sys/wait.h>
  - waitpid() and associated macros
- #include <unistd.h>
  - chdir()
  - fork()
  - exec()
  - pid_t
- #include <stdlib.h>
  - malloc()
  - realloc()
  - free()
  - exit()
  - execl()
  - EXIT_SUCCESS, EXIT_FAILURE
- #include <stdio.h>
  - fprintf()
  - printf()
  - stderr
  - getchar()
  - perror()
  - fgetc()
  - WIFSIGNAL
  - WIFEXITED
- #include <string.h>
  - strcmp()
  - strtok()

- strcat()

The command shell take all five (5) internal commands (– ‘cd’,

‘echo’, ‘history’, ‘pwd’ and ‘exit’.) and take all five (5) external commands (‘ls’, ‘cat’, ‘date’, ‘rm’ and

‘mkdir’.)

How does shell work?

the workflow of the shell will look like this:

1. Startup the shell.
2. Wait for user input.
3. Parse user input.
4. Execute the command and return the result.
5. Go back to 2.

The shell is the parent process. This is the main thread of our program which is waiting for user input.

Errors faced 1 -

However, we cannot execute the command in the main thread itself, because of the following reasons:

1. An erroneous command will cause the entire shell to stop working. We want to avoid this.
2. Independent commands should have their own process blocks. This is known as isolation and falls under fault tolerance.

To be able to avoid this, we use the system call fork , it creates a copy of the current process. The copy is known as the child and each process in a system has a unique process id (pid) associated to it.

1. By invoking fork we are creating a new branch in our program. Fork creates a copy of the current process and creates a new one out of it. The resulting system call returns the process id of the child process.
2. Immediately after the fork call has succeeded, both the child and the parent process (the main thread of our code) are running simultaneously.

The getpid system call returns the current process id.

sleep is not used as:

1. We cannot guarantee that whatever it is that you are waiting for will complete its execution within those n seconds.
2. What if whatever it is that we are waiting for finishes much sooner than n seconds? In that case you are idling unnecessarily.

To execute commands I have used excel().

Error faced 2 –

While writing code for history command, the file pointer was at the end of the file due to which I wasn't able to read the file.

So I used fseek()

Now in our main function, we invoke readline to read an input from the user, and pass it to get input we just defined above. Once the input has been parsed, we call fork and call execl in the child process.

Assumption-

We have assumed that the user will not do piping.

fork –

I have assumed that the all commands work perfectly

But If the OS runs out of memory or reaches the maximum number of allowed processes, a child process will not be created and it will return -1. We add the following to our code:

```
...
while (1) {
      input = readline("unixsh> ");
      command = get_input(input);

      child_pid = fork();
      if (child_pid < 0) {
            perror("Fork failed");
            exit(1);
      }
...
```

malloc - It can fail if the OS runs out of memory. We should exit the program in such a scenario

Test Case

```
diptanshu@diptanshu-G3-3579:~/

==============================
            Simple C Shell
------------------------------
           Diptanshu Mittal :
==============================


diptanshu@diptanshu-G3-3579:~/
diptanshu@diptanshu-G3-3579:~/
bin   dev  initrd.img     lib64 mnt   root  snap  tmp vmlinuz
boot  etc  initrd.img.old lost+found opt   run   srv   usr vmlinuz.old
cdrom home lib    media proc  sbin  sys   var
diptanshu@diptanshu-G3-3579:~/
bin   dev  initrd.img     lib64 mnt   root  snap  tmp vmlinuz
boot  etc  initrd.img.old lost+found opt   run   srv   usr vmlinuz.old
cdrom home lib    media proc  sbin  sys   var
diptanshu@diptanshu-G3-3579:~/

/

diptanshu@diptanshu-G3-3579:~/
diptanshu@diptanshu-G3-3579:~/

/home

diptanshu@diptanshu-G3-3579:~/
history



mkdir dfg
history
ls
ls -l
[3~
cd /
ls
ls
pwd
cd home
pwd
history
diptanshu@diptanshu-G3-3579:~/
Wed Aug 28 22:26:47 IST 2019
diptanshu@diptanshu-G3-3579:~/
mkdir: cannot create directory ‘abc’: Permission denied
diptanshu@diptanshu-G3-3579:~/
bash: cd: : No such file or directory
```

```
diptanshu@diptanshu-G3-3579:~/
diptanshu@diptanshu-G3-3579:~/
diptanshu@diptanshu-G3-3579:~/
 2018232_DiptanshuMittal       exampleShell    'JAVA PROGRAMS'
 2018232_DiptanshuMittal.zip   history     lsh-master
 abc       idea-IC-192.6262.58   simple-c-shell-master
diptanshu@diptanshu-G3-3579:~/
a
diptanshu@diptanshu-G3-3579:~/
diptanshu@diptanshu-G3-3579:~/
```

Sources-

https://disqus.com/by/indradhanushgupta/

geeksforgeeks

https://brennan.io/2015/01/16/write-a-shell-in-c/

C The Complete Reference