

Discussion Session

02.19.2025

P3 Overview

- Building wsh
- external command execution
- built-in commands
- variable substitution
- pipe
- command substitution

fork & exec

```
if (pid == 0) {
    // Child process
    // Prepare the arguments for "ls -al"
    char *args[] = {"ls", "-al", NULL};

    // Execute "ls -al" using execv
    execv("/bin/ls", args);

    // If execv returns, it means it failed
    perror("execv");
    exit(EXIT_FAILURE);
} else {
    // Parent process
    // Wait for the child process to complete
    int status;
    waitpid(pid, &status, 0);

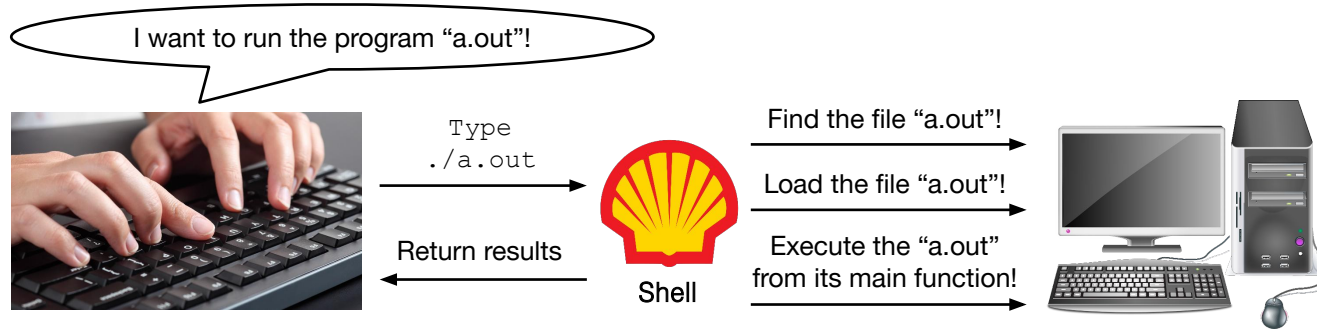
    if (WIFEXITED(status)) {
        printf("Child process exited with status %d\n", WEXITSTATUS(status));
    } else {
        printf("Child process did not exit normally\n");
    }
}
```

pipe, dup2

- pipe()
 - `int pipe(int pipefd[2]);`
 - will allocate two file descriptors, which is the write end and read end.
 - will be connected
- dup2()
 - `int dup2(int oldfd, int newfd);`
 - will copy oldfd into newfd
 - newfd exists -> closed

Shell Review

Shell: the program initiates commands that users type

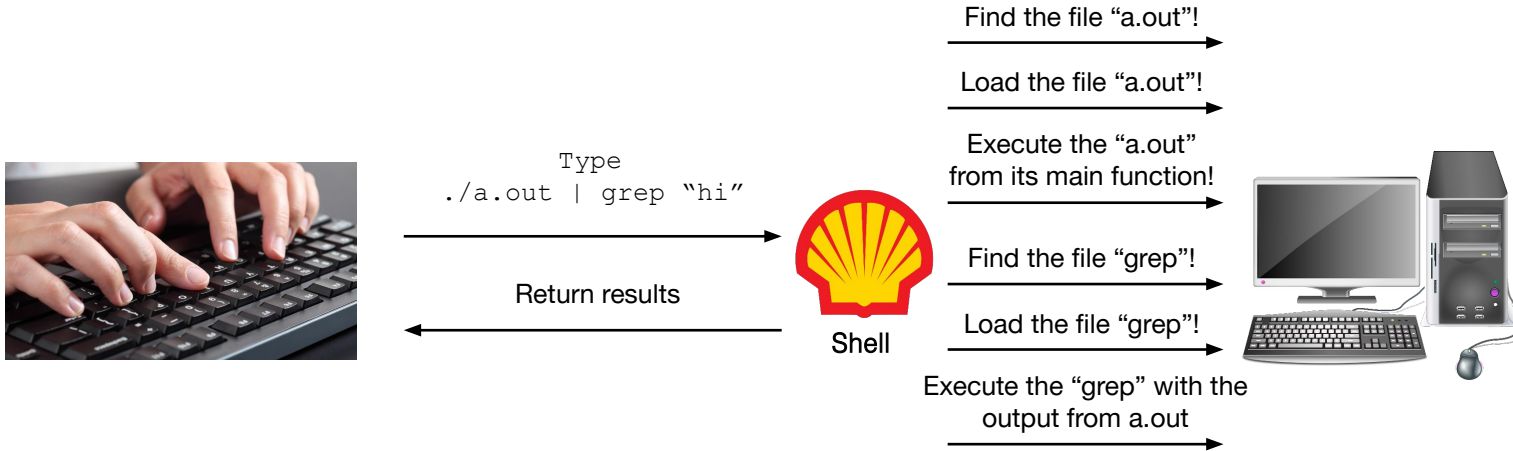


- Receives a string that contains the command from the user
- Interprets the string
- Lets the computer work on initiating the command that the user requested

There is Operating System (Kernel) between the shell and the computer

Shell Review

- Powerful features like pipe



Shell Review

- Shells are really complicated:
 - Bash manual, 196 pages
 - zsh manual, 480 pages

Shell features and OS support

Feature	Interact with OS?
Start new process	Yes
Output redirection	Yes (file descriptors)
History	No
Env variables	No
Local variables	No
Pipe	Yes
Language	No
Job control	Yes (Signals)

Mem Leak

- Allocated memories **should be freed!**
- But should not be **freed more than once!**
- However, we might have a mistake
- Therefore, use a tool!
 - Valgrind
 - Address Sanitizer (ASan)

Valgrind

- A program that tracks allocation / deallocation / memory references
- Let's try

```
$ sudo apt install valgrind
```

In case when your computer do not have valgrind yet

```
$ valgrind ./my_prog
```

You can also discover tremendous functionalities of valgrind using **valgrind -h**

Address Sanitizer

- To enable, build with `-fsanitize=address`
- Then run the compiled program.

```
=====  
==43313==ERROR: LeakSanitizer: detected memory leaks
```

```
Direct leak of 100 byte(s) in 10 object(s) allocated from:
```

```
    #0 0x7f69f14c9a06 in __interceptor_calloc  
    ../../../../src/libsanitizer/asan/asan_malloc_linux.cc:153  
    #1 0x55ce0e3041ec in main (/home/user/discussion_material/week3/a.out+0x11ec)  
    #2 0x7f69f11ee082 in __libc_start_main ../csu/libc-start.c:308
```

```
SUMMARY: AddressSanitizer: 100 byte(s) leaked in 10 allocation(s).
```

Address Sanitizer catches...

- (Global, Stack, Heap) overflow
- Double free
- Use after free
- Memory Leak

Valgrind VS ASan

- Advantages of asan

- Non-heap bugs: Stack overflow, Global overflow, ...
- Much faster: 2x Asan, 20x Valgrind
- Multi-threaded support

- Disadvantage

- Re-compilation required (i.e., Source code is required)
 - Valgrind can detect memory bugs in compiled libraries.
 - But asan only can detect bugs if it is compiled with the option
- Cannot detect uninitialized memory