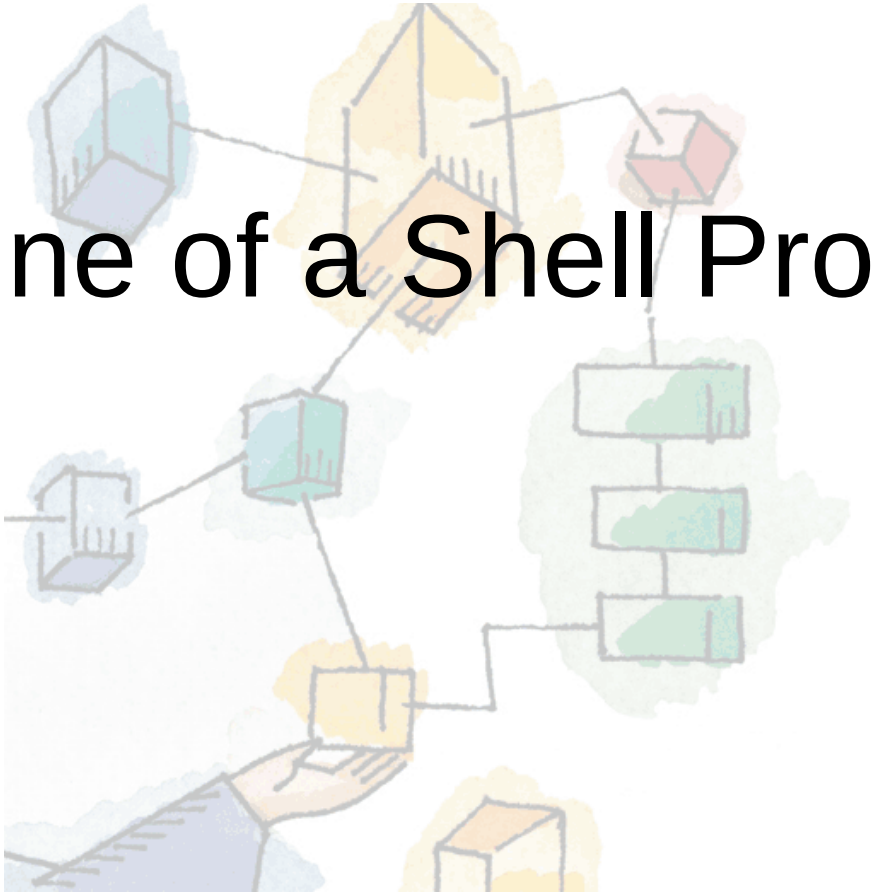
Operating Systems:
Internals and Design Principles, 6/E
William Stallings



Outline of a Shell Program

Outline of a Shell

```
// functions in italics are to be written
for (;;) {
   parse_input_line(arg_vector);
   if built_in_command(arg_vector[0]) {
      do_it(arg_vector);
      continue;
   } // built-in command
   pathname = find_path(arg_vector[0]);
   create_process(pathname, arg_vector);
   if (interactive())
      wait_for_this_child();
} /* for loop */
```

- The real shells are more complicated because they also handle I/O redirection, pipes (as in **`ls -alg | more`**), command aliasing, wildcard characters (such as **`*`**) and so forth.

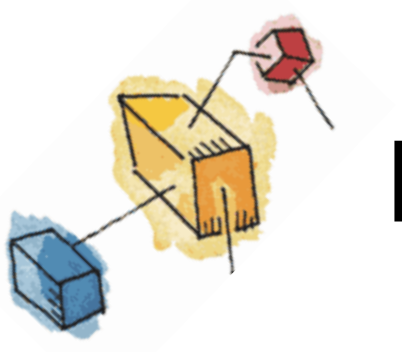
Outline of a Shell

- *With fork(), execve(), and wait(), our shell program thus becomes:*

```
for (;;) {
   parse_input_line(arg_vector);
   if (built_in_command(arg_vector[0])) {
      do_it(arg_vector);
      continue;
   } //built_in command
   pathname = find_path(arg_vector[0]);
   if ((pid = fork()) == 0) {
      // code executed by child
      //put here I/O redirection
      execve(pathname, arg_vector, envp);
      exit(1); /* in case execve fails */
   } //child process
   if (interactive())
      while (wait(0) != pid);
} // main for loop /
```

Outline of a Shell

- Separating the creation of the new process (**fork()**) and the loading of the new program (**execve(...)**) allows the shell to do the I/O redirection within the new process before the new program is loaded
- The **exit(1)** library call is only executed if the **execve(...)** system call fails:
  - without it the child process would start executing the parent code.



Expanded UNIX Shell

```
for (;;) {
    parse_input_line(arg_vector);
    if built_in_command(arg_vector[0]) {
       do_it(arg_vector);
       continue;
    } //built_in command
    pathname = find_path(arg_vector[0]);
    if ((pid = fork()) == 0) {
       // code executed by child
       if (!ispiped) {
           if (stdout_redirect) {
              // open target file and create it if needed
              fd = open(outfile, O_WRONLY | O_CREAT, 0644);
              close(1); // close stdout/
              dup(fd); // dup into stdout
              close(fd); // good practice
           }
           if (stdin_redirect) {
              fd = open(infile, O_RDONLY);
              close(0); // close stdin/
              dup(fd); // dup into stdin
              close(fd); // good practice
           }
           execve(pathname, arg_vector, envp);
           exit(1); /* in case execve fails */
       }
       else exec_pipe(……);
    } //child process
    if (interactive())
       while (wait(0) != pid);
} // main for loop /
```

```
exec_pipe(……) {

    int pd[2];
    pipe(pd); //creates the pipe
     ...
    if ((pid = fork()) == 0) {
       // child does producer/
       close(1); // close stdout
       dup(pd[1]); // dup into stdout
       close(pd[0]); close(pd[1]); //
    goody
       execve(leftsidecommand, p_argv,
    p_envp);
       exit(1); // should execve fail
    } //* end of child
    // parent does consumer
       close(0); // close stdin
       dup (pd[0]; // dup into stdin
       close (pd[0]); close(pd[1]); //
    goody
       execve(rightsidecommand, c_argv,
    c_envp);
       exit(2); // if execve fails
    }
} // main for loop /
```

UNIX Shell

```
for (;;) {
   parse_input_line(arg_vector);
   if built_in_command(arg_vector[0]) {
      do_it(arg_vector);
      continue;
   } //built_in command
   pathname = find_path(arg_vector[0]);
   if ((pid = fork()) == 0) {
      // code executed by child
      if (!ispiped) {
          if (stdout_redirect) {
             // open/create target file
             fd = open(outfile, O_WRONLY | O_CREAT,
                                                0644);
             close(1); // close stdout/
             dup(fd); // dup into stdout
             close(fd); // good practice
          }
          if (stdin_redirect) {
             fd = open(infile, O_RDONLY);
             close(0); // close stdin/
             dup(fd); // dup into stdin
             close(fd); // good practice
          }
```

UNIX Shell

```
            execve(pathname, arg_vector, envp);
            exit(1); /* in case execve fails */
        }
        else exec_pipe(……);
    } //child process
    if (interactive())
        while (wait(0) != pid);
} // main for loop /
```

UNIX Shell

```
exec_pipe(……) {

   int pd[2];
   pipe(pd); //creates the pipe
    ...
   if ((pid = fork()) == 0) {
      // child does producer/
      close(1); // close stdout
      dup(pd[1]); // dup into stdout
      close(pd[0]); close(pd[1]); // goody
      execve(leftsidecommand, p_argv, p_envp);
      exit(1); // should execve fail
   } //* end of child
   // parent does consumer
      close(0); // close stdin
      dup (pd[0]; // dup into stdin
      close (pd[0]); close(pd[1]); // goody
      execve(rightsidecommand, c_argv, c_envp);
      exit(2); // if execve fails
   }
} // main for loop /
```