CS 537 Discussion

8 February, 2023

Agenda

- Project 3
- Introduction to how shell works
- Process creation and management
  - fork()/exec()/waitpid() examples
- I/O redirection
- Reminder: Project 2 is due tonight at 11:59 pm

Project-3 Overview

Shell Processes

```
# The output of the following commands may vary depending on what shell you are running
# Below is from zsh. If you are using bash, it may look different
$ which gcc
/usr/bin/gcc
$ which alias
alias: shell built-in command
$ which which
which: shell built-in command
```

fork()

```c
pid_t pid = fork();
if (pid == 0) { // the child process will execute this
    printf("I am child with pid %d. I got return value from fork: %d\n", getpid(), pid);
    exit(0); // we exit here so the child process will not keep running
} else { // the parent process will execute this
    printf("I am parent with pid %d. I got return value from fork: %d\n", getpid(), pid);
}
```

```
$ ./fork_example
I am parent with pid 46565. I got return value from fork: 46566
I am child with pid 46566. I got return value from fork: 0
```

execv()

```c
pid_t pid = fork();
if (pid == 0) {
    // the child process will execute this
    char *const argv[3] = {
        "/bin/ls", // string literial is of type "const char*"
        "-l",
        NULL // it must have a NULL in the end of argv
    };
    int ret = execv(argv[0], argv);
    // if succeeds, execve should never return
    // if it returns, it must fail (e.g. cannot find the executable file)
    printf("Fails to execute %s\n", argv[0]);
    exit(1);
}
// do parent's work
```

waitpid()

```
$ sleep 10 # this will create a new process executing /usr/bin/sleep
```

```
pid_t waitpid(pid_t pid, int *status, int options);
```

```
pid_t pid = fork(); // create a new child process
if (pid == 0) { // this is child
    // do some preparation
    execv(path, argv);
    exit(1); // this means execv() fails
}
// parent
int status;
waitpid(pid, &status, 0); // wait the child to finish its work before keep going
// continue to handle the next command
```

strsep()

```c
char * strsep(char **stringp, const char *delim); //The first argument is the string that
                                                  //is provided as an input to this function that you
                                                  // to slice, whereas the second argument is the delimiter
                                                  //according to which you wish to slice the provided string.
```

```c
int main(int argc, char *argv[]){
    //strdup save a copy of a string
    char *str = strdup("This is a line");

    //strsep doest not wotk on constant string
    char *tok = strsep(&str, " ");

    while(tok != NULL){
        printf("%s\n", tok);
        tok = strsep(&str, " ");
    }
    free(str);
    return 0;
}
```

Standard I/O

- Standard Output (stdout, file descriptor 1)
  - default place to which programs write
- Standard Input (stdin, file descriptor 0)
  - default place from which programs read
- Standard Error (stderr, file descriptor 2)
  - default place where errors are reported

I/O redirection

- Redirecting standard output (Operator: >)
  - `ls > my_files (redirects the output of ls to my_files)`
  - `ls >> my_files (output of ls is appended to my_files)`
- Redirecting standard input (Operator: <)
  - `wc < my_files (counts the number of words in my_files)`
  - `wc < my_files > wc_ouput (counts the number of words in my_files and redirects it to wc_output)`

I/O redirection using dup2()

```
int dup2(int f_orig, int f_new) ;//duplicates an existing file descriptor.
                                 //It takes two file descriptors as parameters and duplicates the
                                 // first one (f_orig) onto the second one (f_new).
```

I/O redirection using dup2()

```c
//output redirection with dup2()
//sends the output of a command to a file of the user's choice.
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
int main(int argc, char *argv[]){
    int pid, status;
    int newfd;  /* new file descriptor */

    char *cmd[] = { "/bin/ls", "-al", "/", NULL };
    if (argc != 2) {
        fprintf(stderr, "usage: %s output_file\n", argv[0]);
        exit(1);
    }
    if ((newfd = open(argv[1], O_CREAT|O_TRUNC|O_WRONLY, 0644)) < 0) {
        perror(argv[1]);    /* open failed */
        exit(1);
    }
    printf("writing output of the command %s to \"%s\"\n", cmd[0], argv[1]);

    /* this new file will become the standard output */
    /* standard output is file descriptor 1, so we use dup2 to */
    /* to copy the new file descriptor onto file descriptor 1 */
    /* dup2 will close the current standard output */

    dup2(newfd, 1);

    /* now we run the command. It runs in this process and will have */
    /* this process' standard input, output, and error */

    close(newfd); /* close unused file descriptor*/
    execv(cmd[0], cmd);
    perror(cmd[0]);     /* execv failed */
    return 0;
}
```

Pipes

- Redirects the output (stdout) of first command as the input (stdin) of second command.

$ cat file.txt | grep ‘some text’

$ cat file.txt | rev | grep ‘txet emos’

pipe()

```
int pipe(int fds[2]);//fds[0] will be the fd(file descriptor) for the read end of pipe.
                     //fds[1] will be the fd for the write end of pipe.
```

pipe()

```c
/**
 * Executes the command "cat scores | grep bob".  In this
 * implementation the parent doesn't wait for the child to finish and
 * so the command prompt may reappear before the child terminates.
 */
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
int main(int argc, char *argv[]){
  int pipefd[2];

  char *cat_args[] = {"/bin/cat", "scores", NULL};
  char *grep_args[] = {"/bin/grep", "bob", NULL};

  // make a pipe (fds go in pipefd[0] and pipefd[1])
  pipe(pipefd);

  int pid = fork();

  if (pid == 0){
      // child gets here and handles "grep bob"
      // replace standard input with input part of pipe
      dup2(pipefd[0], 0);

      // close unused half of pipe
      close(pipefd[1]);

      // execute grep
      execv(grep_args[0], grep_args);
    }
  else{
      // parent gets here and handles "cat scores"
      // replace standard output with output part of pipe
      dup2(pipefd[1], 1);

      //wait(NULL);
      // close unused half of pipe
      close(pipefd[0]);

      // execute cat
      execv(cat_args[0], cat_args);
      perror(cat_args[0]);

    }
    return 0;
}
```

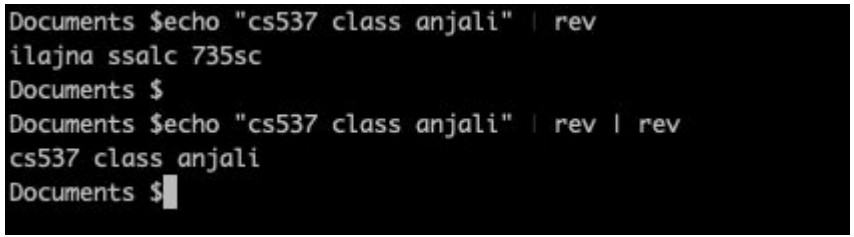
Valgrind demo

Some Unix Utilities

- cat : Outputs the contents of the file

  Ex: cat my_file (prints the contents of my_file)

- rev: Reverse everything - line by line

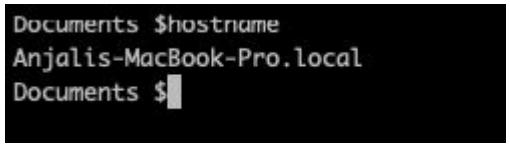


  Ex: Remove last character of line

  $ cat filename | rev | cut -c2- | rev

Some Unix Utilities

- grep: Pattern matching utility.
  - Search for exact match of a string
    - grep "literal_string" filename
  - Search lines ending in “ing”
    - grep “ing$” filename
  - Search string starting with A0,A1…A9
    - grep “^A[0-9]” filename
- hostname - prints name of current host system



- whoami - shows username