COP4610 Project 1

A Simple Shell

Overview

- A simple shell that interprets commands from either user input or a file
  - Interactive vs Batch modes
- Commands are delimited by ‘;’
- All valid commands on a given line of input should be run simultaneously
  - fork() and wait()
- Shells exit when either given the built-in command “quit” or when EOF is reached
  - However, the shell must wait for other commands to finish running

Interactive Mode

- Interactive mode is the default shell behavior; i.e., when the program is run without any arguments
  - $ tinysh
- Interactive mode displays a prompt to the user whenever the shell is ready for more input
  - prompt> *command1; command2; ...*
- When EOF is reached (CTRL+D in this mode) or the “quit” command is entered
  - First, ensure completion of all commands on the same line in which “quit” appears
  - Once all commands have finished execution, exit

Batch Mode

- Batch mode is entered only when the shell is invoked with an argument
  - $ tinysh /path/to/valid/batch/file
- Do not forget that “quit” is a valid command in batch mode
  - When “quit” is encountered in batch mode, do **not** handle any subsequent lines
  - Commands that appear on the same line as “quit” must still be executed
- Batch files will be 0 or more lines of command strings, i.e.
  - $ cat example.batch

    ls -l; echo hello

    echo goodbye; quit; cat /foo/bar

    cat /bar/foo <= NOT EXECUTED

Library Routines: fgets()

- http://linux.die.net/man/3/fgets
- #include <stdio.h>
- **char *fgets(char *s, int num, FILE *stream);**
- Reads characters from stream until
  - num - 1 characters are read
  - Newline is encountered
  - EOF is reached
- Returns str on success
- Returns NULL
  - On error
  - When EOF reached and nothing read

- Example usage

```
int BUFSIZE = 1024;
char buf[BUFSIZE];

if(fgets(buf, BUFSIZE, stdin) == NULL) {
        // error handling goes here
} else {
        printf("buf: %s\n", buf);
}
```

Library Routines: strtok()

- http://linux.die.net/man/3/strtok
- #include <string.h>
- **char strtok(char *str, const char *delim);**
- delim acts as the delimiter for tokens
  - That is, the symbol that will be matched to split a string into tokens
- str should be the string to tokenize on the **first** call to strtok()
  - Subsequent calls wishing to tokenize the same string will then pass NULL in place of str

- Example usage

```
char* token;
char* delim = “+”;

token = strtok(“foo+ bar+ foobar”, delim);
while(token != NULL) {
        // do something with token
        token = strtok(NULL, delim);
}
```

- In the above code, token should take the value of
  - foo, bar, foobar

Library Routines: fork()

- http://linux.die.net/man/2/fork
- #include <unistd.h>
- **pid_t fork(void);**
- Duplicates calling process
  - Functionally, this *child* is an exact duplicate of the *parent*
  - Unique PID
- To the parent, fork() returns either -1 on failure or the PID of the child on success
  - 0 is returned to the child
- Typically, the parent process will wait() for a child to finish execution

- Example usage

```
pid_t pid;
if((pid = fork()) < 0) {
        perror("fork");
} else if(pid == 0) {
        // inside the child, so do work and..
        exit(1);
}
```

Library Routines: wait()

- http://linux.die.net/man/2/wait
- #include <sys/types.h>
- #include <sys/wait.h>
- **pid_t wait(int *status);**
- Waits for a state change in a child of the calling process
  - Termination or start/stop due to a signal
- Allows for child's resources to be released
  - Without wait()ing, child becomes a zombie
- This function blocks until the child's status has changed
  - Returns immediately if the child has already terminated

- Example usage

```
// in a parent process after a fork()

pid_t pid;
int status;

pid = wait(&status);
printf("PID %d exited with status %d\n",
pid, status);
```

Library Routines: execvp()

- http://linux.die.net/man/3/execvp
- #include <unistd.h>
- **int execvp(const char *file, char *const argv[]);**
- file is the name of a file to be executed
  - I.e., a command
- argv[] is the full array of arguments
  - argv[0] holds the filename, and can be used as the file argument
  - argv[] must be terminated by a null pointer
- On success, execvp() does not return
  - The execvp()'d command replaces the current process image
  - Therefore, an error with execvp() has occured if this function is returned from

- Example usage

```
char* args[3];
args[0] = "ls";
args[1] = "-l";
args[2] = NULL;

if(/* successful fork */) {
        execvp(args[0], args);
        printf("execvp() failed\n");
        exit(0);
}
```

Hints

- Depending on your implementation, you may consider keeping track of how many processes were fork()ed
  - You will need to wait() on each of these processes **after** they have all been fork()ed
- Test your program as you go
  - If you write a couple lines of code and think of something that might go wrong
    - Try it out and see if your program breaks
    - Include error handling
      - Waiting to include error handling may cause hours of headache-inducing debugging