# `dsh`: A Diagnostic Shell

Noah Brubaker

February 12, 2016

Course: CSC456 – Operating Systems

Instructor: Dr. Jeff McGough

## Program Description

This program is a simple diagnostic shell that will emulate some of the functionality of the standard Bash shell. The main purpose of this shell is process identification, and to provide a platform for further development.

The program will provide the following features:

- **Prompt:** The prompt `dsh>` will be displayed. This is where the user will enter commands.
- **Intrinsic Commands:** Six shell intrinsic commands will be implemented: `cmdnm, signal, systat, exit, cd, and pwd`. `cmdnm` prints the command that initiated a process. `signal` sends a signal to another process. `systat` displays some information about the system, including version, uptime, memory usage, and CPU info. `exit` will exit the shell nicely. `cd` implements the `chdir` command to change the directory via the relative or absolute path provided. `pwd` prints the working directory.
- **Single Program Command:** Any single command (plus arguments), will be executed by the shell and return any stdout.

## Submission Details

The submission includes a tar-ball, `prog1.tgz`, which contains all files relevant to the program. This includes the source code, makefile, and documentation.

`prog1.tgz` contains:

- `dsh.c`: This file implements the command prompt, command line input, input parsing, and the main event loop for the program.
- `run.c` This file implements the intrinsic commands, as well as `fork/exec` for single commands with arguments.
- `makefile` This file builds the program `dsh` from the source files `dsh.c` and `run.c`.
- `prog1.pdf` This file provides documentation for the program `dsh`, its source files and the makefile.

## Compilation and Usage

The makefile builds the program in the following way:

```
gcc dsh.c -o dsh -g -Wall
```

The program can be run by typing `dsh` is a bash shell.

## Libraries

The source code includes the following libraries.

- `stdio.h`
- `string.h`
- `stdlib.h`
- `signal.h`
- `sys/stat.h`
- `sys/wait.h`
- `sys/time.h`
- `sys/resource.h`
- `sys/types.h`
- `unistd.h`

## Structure and Functions

The general flow of the program has the following format.

### Program structure

```
do
  getInput
  parseInput
  status = handler(input)
while status ≡ 0
```

### Function Descriptions

This section describes all functions implemented in the source code.

**Name:** `dsh_prompt[dsh.c(33)]`

**Description:**
This function recieves command line input for the shell. The storage is dynamically allocated for the input stream in blocks of 256 bytes

**Output:**

`char** input` A pointer to a character array which will store the input taken at the prompt.

**Returns:**

| | |
|---|---|
| `int -1` | Failed to allocate memory for input |
| `int 0` | Function successful took input |
| `int 1` | No input received on commandline |
| `int 2` | Exit command received |

---

**Name:** `parse_input[dsh.c(105)]`

**Description:**
This function parses input gathered from the command line.

**Input:**

| | |
|---|---|
| `char * input` | The input string returned by prompt. |

**Output:**

| | |
|---|---|
| `char *** argv` | A pointer to the new parsed argument list, passed by reference. |

**Returns:**

| | |
|---|---|
| `int argc` | The number of arguments in the input string. |

---

**Name:** `run_command[dsh.c(215)]`

**Description:**
This function takes the argument list from Main and directs it to either the fork/exec code for single functions or to the instrinsic commands.

The first argument is expected to be the command name.

**Input:**

| | |
|---|---|
| `int args` | Number of arguments |
| `char ** arg_list` | List of arguments |

**Returns:**

| | |
|---|---|
| `int ret` | Returns the valued returned by `Run` or `New_Process`. |

---

**Name:** `main[dsh.c(243)]`

**Description:**
This function implements the main event loop for the shell. It waits for the exit command to terminate.

**Returns:**

| | |
|---|---|
| `int 0` | Always returns 0. |

---

**Name:** `cmdnm[run.c(30)]`

**Description:**
This function gets the command that started a process by accessing `/proc/<pid>/comm`.

**Input:**

`char * pid` A character array holding the process identification number.

**Returns:**

`int 0` Successful.

`int -1` Couldn't find process.

---

**Name:** `send_signal[run.c(63)]`

**Description:**
This function sends a signal to a process using the kill command. It checks if the arguments are in the proper ranges, switching them if not.

**Input:**

`char * sig_no` A character array holding the desired signal number.

`char * process_id` A character array holding the process identification number.

**Returns:**

`0` Successful.

`-1` Failed to send signal to process.

---

**Name:** `systat[run.c(93)]`

**Description:**
This function gets some information about the system and displays it for the user in stdout. The specific information it provides is as follows:
-Linux version and system uptime
-Memory Usage: memtotal and memfree
-CPU Information: vendor id through cache size

**Returns:**

`int 0` Successful.

`int neg` Couldn't access directory.

---

**Name:** `cd[run.c(176)]`

**Description:**
This function implements the change directory intrinsic command.

**Input:**

`char * path` The absolute or relative path to the desired directory.

**Returns:**

`0` Successful.

| | |
|---|---|
| `-1` | No such file or directory. |

---

**Name:** `pwd[run.c(203)]`

**Description:**
This function implements the print working directory intrinsic command.

**Returns:**

| | |
|---|---|
| `0` | Always returns 0. |

---

**Name:** `Run[run.c(221)]`

**Description:**
This function directs the program to run the intrinsic commands, checking for correct number of arguments where applicable.

**Input:**

| | |
|---|---|
| `int cmd_num` | Number specifying desired command. |
| `int args` | The number of arguments. |
| `char ** arg_list` | The null-terminated list of arguments. |

**Returns:**

| | |
|---|---|
| `int ret` | The return value of function it calls |
| `int neg` | Wrong number of inputs or similar error. |
| `int 2` | Exit code. |

---

**Name:** `New_Process[run.c(269)]`

**Description:**
Creates a new process to run the given single command received at the command line in the diagnostic shell.

**Input:**

| | |
|---|---|
| `char ** arg_list` | The list of arguments for the given command. |

**Returns:**

| | |
|---|---|
| `int 0` | If fork and exec operations were successful. |
| `int -1` | An error occured. Either couldn't find command or failed to execute it. |

---

## Testing and Verification

This program was tested and verified by trying each required command at separate times. The code was developed in such a way that functionally was continually added to an already functional program. Since each required feature was largely independent of the others, debugging was straight-forward.

Valgrind was used to check for memory leaks. There are no known bugs at the time of submission, however error checking could be a bit more rigorous.