OMG -

let's build a shell in 10 mins

Stefanie Schirmer
@linse

OMG -

let's build a shell in 10 mins

Stefanie Schirmer
@linse

What is a shell?

show prompt + read input

```c
#include <stdio.h>  // printf
#include <string.h> // strlen
#include <stdlib.h> // exit
#include <unistd.h> // syscalls

int MAXLINE = 1024;
char prompt[] = "lnsh> "; // command line prompt

int main(int argc, char **argv) {
  char cmdline[MAXLINE]; // buffer for fgets

  while (1) {

    printf("%s", prompt);

    if ((fgets(cmdline, MAXLINE, stdin) == NULL) && ferror(stdin))
      error("fgets error");

    if (feof(stdin)) {
      printf("\n");
      exit(0);
    }

    // remove trailing newline
    cmdline[strlen(cmdline)-1] = '\0';

    // evaluate command line
    eval(cmdline);
  }
}
```

call parse

```
void eval(char *cmdline) {
  int bg;
  struct command cmd;

  printf("Evaluating '%s'\n", cmdline);

  // parse cmdline into cmd structure
  bg = parse(cmdline, &cmd);
  printf("Found command %s\n", cmd.argv[0]);
  printf("Builtin %d\n", cmd.builtin);

  // -1 means parse error
  if (bg == -1) return;
}
```

command struct

```
// maxargs in struct can't be variable, use define
#define MAXARGS 128

struct command {
  int argc;                 // number of args
  char *argv[MAXARGS];      // arguments list
  enum builtin_t {          // is argv[0] a builtin command?
    NONE, QUIT, JOBS, BG, FG } builtin;
};
```

```c
int parse(const char *cmdline, struct command *cmd) {
    static char array[MAXLINE];          // local copy of command line
    const char delims[10] = " \t\r\n";   // argument delimiters
    char *buf = array;                   // ptr that traverses command line
    char *next;                          // ptr to the end of the current arg
    char *endbuf;                        // ptr to the end of the cmdline string
    int is_bg;                           // background job?

    if (cmdline == NULL) {
        (void) fprintf(stderr, "Error: command line is NULL\n");
        return -1;
    }

    (void) strncpy(buf, cmdline, MAXLINE);
    endbuf = buf + strlen(buf);

    // build argv list
    cmd->argc = 0;

    while (buf < endbuf) {
        // skip delimiters
        buf += strspn (buf, delims);
        if (buf >= endbuf) break;

        // Find next delimiter
        next = buf + strcspn (buf, delims);

        // terminate the token
        *next = '\0';

        // Record token as the next argument
        cmd->argv[cmd->argc++] = buf;

        // Check if argv is full
        if (cmd->argc >= MAXARGS-1) break;

        buf = next + 1;
    }

    // argument list must end with a NULL pointer
    cmd->argv[cmd->argc] = NULL;

    if (cmd->argc == 0)  // ignore blank line
        return 1;

    cmd->builtin = parseBuiltin(cmd);

    // should job run in background?
    if ((is_bg = (*cmd->argv[cmd->argc-1] == '&')) != 0)
        cmd->argv[--cmd->argc] = NULL;

    return is_bg;
}
```

parse():

line -> struct

builtin shell commands

```
enum builtin_t parseBuiltin(struct command *cmd) {
    if (!strcmp(cmd->argv[0], "quit")) { // quit command
        return QUIT;
    } else if (!strcmp(cmd->argv[0], "jobs")) { // jobs command
        return JOBS;
    } else if (!strcmp(cmd->argv[0], "bg")) { // bg command
        return BG;
    } else if (!strcmp(cmd->argv[0], "fg")) { // fg command
        return FG;
    } else {
        return NONE;
    }
}
```

System calls

run programs

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

int main (int argc, char const *argv[]) {
  printf("Hi there\n");

  // executing an ls command
  char* argv2[] = {"ls", "-la", NULL};
  execvp(argv2[0], argv2);

  printf(".. and done!");
}
```

fork + children

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
  pid_t childPid;
  printf("Before fork %d\n", getpid());

  // fork creates a child and returns its process ID
  childPid = fork();

  printf("After fork %d\n", getpid());
  exit(0);
}
```

Fork + execvp

`shell.c`

evaluate for real

```
// updated!!!
void eval(char *cmdline) {
  int bg;
  struct command cmd;

  printf("Evaluating '%s'\n", cmdline);
  // parse line into command struct
  bg = parse(cmdline, &cmd);

  // parse error
  if (bg == -1) return;
  // empty line - ignore
  if (cmd.argv[0] == NULL) return;

  if (cmd.builtin == NONE)
    runSystemCommand(&cmd, bg);
  else
    runBuiltinCommand(&cmd, bg);
}
```

fork + execvp = shell

```
void runSystemCommand(struct command *cmd, int bg) {
    pid_t childPid;
    // FORK !!!
    if ((childPid = fork()) < 0)
        error("fork() error");
    else if (childPid == 0) { // I'm the child and could run a command
        // EXECVP !!!!
        if (execvp(cmd->argv[0], cmd->argv) < 0) {
            printf("%s: Command not found\n", cmd->argv[0]);
            exit(0);
        }
    }
    else { // I'm the parent. Shell continues here.
       if (bg)
          printf("Child in background [%d]\n",childPid);
       else
          wait(&childPid);
    }
}
```

builtin shell commands

```
void runBuiltinCommand(struct command *cmd, int bg) {
    switch (cmd->builtin) {
        case QUIT:
            printf("TODO: quit\n"); break;
        case JOBS:
            printf("TODO: jobs\n"); break;
        case BG:
            printf("TODO: background\n"); break;
        case FG:
            printf("TODO: foreground\n"); break;
        default:
            error("Unknown builtin command");
    }
}
```

```c
#include <stdio.h>  // printf
#include <string.h> // strlen
#include <stdlib.h> // exit
#include <unistd.h> // syscalls

#define MAXLINE 1024
// maxargs used in struct, can't be variable so we have to use define
#define MAXARGS 128

char prompt[] = "linsh> "; // command line prompt

struct command {
  int argc;              // number of args
  char *argv[MAXARGS];   // arguments list
  enum builtin_t {       // is argv[0] a builtin command?
    NONE, QUIT, JOBS, BG, FG } builtin;
};

enum builtin_t parseBuiltin(struct command *cmd) {
    if (!strcmp(cmd->argv[0], "quit")) { // quit command
        return QUIT;
    } else if (!strcmp(cmd->argv[0], "jobs")) { // jobs command
        return JOBS;
    } else if (!strcmp(cmd->argv[0], "bg")) { // bg command
        return BG;
    } else if (!strcmp(cmd->argv[0], "fg")) { // fg command
        return FG;
    } else {
        return NONE;
    }
}

int parse(const char *cmdline, struct command *cmd) {
    static char array[MAXLINE];          // local copy of command line
    const char delims[10] = " \t\r\n";   // argument delimiters
    char *buf = array;                   // ptr that traverses command line
    char *next;                          // ptr to the end of the current arg
    char *endbuf;                        // ptr to the end of the cmdline string
    int is_bg;                           // background job?

    if (cmdline == NULL) {
        (void) fprintf(stderr, "Error: command line is NULL\n");
        return -1;
    }

    (void) strncpy(buf, cmdline, MAXLINE);
    endbuf = buf + strlen(buf);

    // build argv list
    cmd->argc = 0;

    while (buf < endbuf) {
        // skip delimiters
        buf += strspn (buf, delims);
        if (buf >= endbuf) break;

        // Find next delimiter
        next = buf + strcspn (buf, delims);

        // terminate the token
        *next = '\0';

        // Record token as the next argument
        cmd->argv[cmd->argc++] = buf;

        // Check if argv is full
        if (cmd->argc >= MAXARGS-1) break;

        buf = next + 1;
    }

    // argument list must end with a NULL pointer
    cmd->argv[cmd->argc] = NULL;

    if (cmd->argc == 0)  // ignore blank line
        return 1;

    cmd->builtin = parseBuiltin(cmd);

    // should job run in background?
    if ((is_bg = (*cmd->argv[cmd->argc-1] == '&')) != 0)
        cmd->argv[--cmd->argc] = NULL;

    return is_bg;
}

void error(char *msg) {
  printf("Error: %s", msg);
  exit(0);
}

void runSystemCommand(struct command *cmd, int bg) {
    pid_t childPid;
    // FORK !!!!
    if ((childPid = fork()) < 0)
        error("fork() error");
    else if (childPid == 0) { // I'm the child and could run a command
        // EXECVP !!!!
        if (execvp(cmd->argv[0], cmd->argv) < 0) {
            printf("%s: Command not found\n", cmd->argv[0]);
            exit(0);
        }
    }
    else { // I'm the parent. Shell continues here.
      if (bg)
        printf("Child in background [%d]\n",childPid);
      else
        wait(&childPid);
    }
}


void runBuiltinCommand(struct command *cmd, int bg) {
    switch (cmd->builtin) {
        case QUIT:
            printf("TODO: quit\n"); break;
        case JOBS:
            printf("TODO: jobs\n"); break;
        case BG:
            printf("TODO: background\n"); break;
        case FG:
            printf("TODO: foreground\n"); break;
        default:
            error("Unknown builtin command");
    }
}


// We changed this
void eval(char *cmdline) {
  int bg;
  struct command cmd;

  printf("Evaluating '%s'\n", cmdline);
  // parse line into command struct
  bg = parse(cmdline, &cmd);

  // parse error
  if (bg == -1) return;
  // empty line - ignore
  if (cmd.argv[0] == NULL) return;

  if (cmd.builtin == NONE)
    runSystemCommand(&cmd, bg);
  else
    runBuiltinCommand(&cmd, bg);
}

// We already know this ..
int main(int argc, char **argv) {
  char cmdline[MAXLINE]; // buffer for fgets
  char c;

  while (1) {

    printf("%s", prompt);

    if (((fgets(cmdline, MAXLINE, stdin) == NULL) && ferror(stdin))
      error("fgets error");

    if (feof(stdin)) {
      printf("\n");
      exit(0);
    }

    // Remove trailing newline
    cmdline[strlen(cmdline)-1] = '\0';

    // Evaluate command line
    eval(cmdline);
  }
}
```

done!

Thank you!

I'm Stefanie Schirmer
sschirme@gmail.com
@linse on twitter