

CSC-458 Exam #1

Gabriel Hofer

February 26, 2021

Submission Contents

The compressed folder contains three items: `Makefile`, `dash.c`, `exam1.pdf`.

`dash.c` contains the entire diagnostic shell program. `exam1.pdf` contains the writeup and documentation for the program. `Makefile` compiles `dash.c`.

How to Run the Program

Execute the following commands in the terminal to run the program:

```
tar -xzf exam1.tgz
cd exam1
make
./dash
```

Testing & Verification

This program was tested by running the `dash` executable and entering miscellaneous commands. Then, the output from `dash` was compared to the data or contents in the `/proc` directory to verify correctness.

main & REPL

The entry point for the program is the `main` function. `main` immediately calls the REPL (read-eval-print-loop). The REPL reads one line at a time and evaluates the command. If the command name is in the set `{ cmdnm, pid, cd, systat }`, then `dash` calls the function with the same name in the `dash.c` file. If the command isn't in the set, then `fork` and `execvp` are used to execute the command with a child process. The shell is terminated when the `exit` command is entered.

```

void REPL(){
    char * buf=NULL;
    size_t leng=32;
    char a[32], b[32];
    char str[] = "cmdnm\0pid\0cd\0systat\0exit\0";
    char cwd[PATH_MAX];

    for(;;){
        a[0]=b[0]='\0';
        getcwd(cwd, sizeof(cwd));
        printf("%s> ", cwd);
        getline(&buf, &leng, stdin);
        sscanf(buf, "%s%s", a, b);

        if(!strcmp(a, str)) cmdnm(b);
        else if(!strcmp(a, &str[6])) pid(b);
        else if(!strcmp(a, &str[10])) cd(b);
        else if(!strcmp(a, &str[13])) systat();
        else if(!strcmp(a, &str[20])) return;
        else otherwise(buf);
    }
}

void main(){
    REPL();
}

```

cmdnm

Returns the command string (name) that started the process for a given process id. The function looks in the `/proc/<PID>/stat` file for the command string (it's the second element in the file).

```

void cmdnm(char * pid){
    char path[256], scratch[256], nm[256];
    sprintf(path, "/proc/%s/stat", pid);
    FILE * fp = fopen(path, "r");
    if(fp==NULL) return;
    fscanf(fp, "%s%s", scratch, nm);
    printf("%s\n", nm);
    fclose(fp);
}

```

pid

Returns the process IDs for a given command string. The function looks through every `/proc/<PID>/stat` file in the `/proc` directory and prints the name of a directory when the command function parameter matches the second element in a `stat` file.

```
void pid(char * command) {
    char pid[33], command2[256];
    sprintf(command2, "(%s)", command);
    DIR * dir;
    struct dirent *dp;
    if((dir = opendir("/proc")) == NULL){
        perror("Can't open directory\n");
        return;
    }
    while((dp = readdir(dir)) != NULL){
        char path[512], scratch[256], nm[256];
        sprintf(path, "/proc/%s/stat", dp->d_name);
        FILE *fp = fopen(path, "r");
        if(fp==NULL) continue;
        fscanf(fp, "%s%s", scratch, nm);
        if(!strcmp(command2,nm))
            printf("%s\n",dp->d_name);
        fclose(fp);
    }
}
```

systat

Prints some process information to stdout: Linux version, system uptime, memory usage, cpu info. Specifically, the `showfile` function is called 4 times from `systat` to print the contents of 4 different files.

```
void showfile(char * filename){
    printf("%s: \n",filename);
    char * buf=NULL;
    size_t leng=32;
    char path[256];
    sprintf(path, "/proc/%s", filename);
    FILE * fp = fopen(path,"r");
    if(fp==NULL) return;
    while(getline(&buf, &leng, fp) != -1)
        printf("%s",buf);
    printf("\n");
    fclose(fp);
}
```

```

void systat(){
    char nm[] = "version\Ouptime\Omeminfo\Ocpuinfo\O";
    showfile(nm);
    showfile(&nm[8]);
    showfile(&nm[15]);
    showfile(&nm[23]);
}

```

cd

Changes the directory according to the path given using the `chdir()` function.

```

void cd(char * dir){
    printf("%s\n",getenv("HOME"));
    if(strlen(dir)==0){
        printf("inside\n");
        chdir(getenv("HOME"));
    }
    else if(chdir(dir)!=0)
        perror("something went wrong\n");
}

```

fork & execvp

Creates a new child process. The child process executes the command that was entered in into `dash` from `stdin`. The parent process waits for the child process to terminate before proceeding. The PID of the child process is printed immediately after it's created.

```

void otherwise(char * buf){
    int status;
    int pid = fork();
    if(pid!=0) printf("Child PID: %d\n",pid);
    if(pid==0){
        char * name[4];
        name[0] = "/bin/bash";
        name[1] = "-c";
        name[2] = buf;
        name[3] = NULL;
        execvp("/bin/sh",name);
    }
    get_RUSAGE_CHILDREN();
    waitpid(pid,&status,0);
}

```