CS 240

Hw 02

Due October 7th, 2009

Colby Blair

I. Goal/Motivation

The goal of the assignment is to further familiarize ourselves with forking and process handling by building our own unix shell, called tws (for tripwire). This first implementation needs to get input from the user using getchar(), parse commands from the user to a filename and its arguments, and fork to a child. The child needs to recognize it is a child, execute the commands, and return, while the parent waits for the child to complete. Upon completion, the parent starts the next cycle, or exits.

II. Code

See the section at the end of the file.

III. Data

```
myriad360@anubis:~/CS_240/hw02$ ls
tws  #tws.c#  tws.c  tws.c~
myriad360@anubis:~/CS_240/hw02$ ./tws
?: ls
tws  #tws.c#  tws.c  tws.c~
?: ls -talh
total 32K
drwx------ 4 myriad360 myriad360 4.0K 2009-10-07 09:37 ..
drwx------ 2 myriad360 myriad360 4.0K 2009-10-07 03:18 .
-rw------- 1 myriad360 myriad360 1.3K 2009-10-07 03:18 #tws.c#
lrwxrwxrwx 1 myriad360 myriad360   32 2009-10-07 03:18 .#tws.c ->
myriad360@anubis.3600:1254788969
-rw------- 1 myriad360 myriad360 1.3K 2009-10-07 03:18 tws.c
-rwxr-xr-x 1 myriad360 myriad360 9.2K 2009-10-07 03:18 tws
-rw------- 1 myriad360 myriad360 1.7K 2009-10-07 01:42 tws.c~
```

```
?: which ls
/bin/ls
?: exit
myriad360@anubis:~/CS_240/hw02$
```

IV. Data Revision / Conclusion

The most difficult part of the homework continues to be string parsing in C. It has been about a year since I have programmed in C, spending my time in higher level languages. So it is good to get back to my favorite language (because of its efficiency in run time), but I need to start developing a string library of my own. The exact nature of forking actually took me by surprise; I thought the entire program image was duplicated, and the child process started over from the beginning. This later idea is false, however, although I am not sure about the former. The child starts at the point in the code that the parent was. Which can be a good feature, I am not sure if it is the best or not.

Something that I want my shell to do eventually is maintain its own environment variables (like PATH), instead of inheriting it from my calling shell. This wasn't necessary yet, but will be in future implementations I think. I would also like to include flex and bison parsing that I have used in another CS class, but I haven't created parsing trees with the final bison output yet, so that will have to wait as well. The current tws version also need to be modulized, but I had some problems passing variables by reference and with segmentation faults, so my future studying of C will hopefully correct this.

Overall, the homework was a lot simplier than I thought it was going to be. But the devil is always in the details in C, so relearning C is most of my time spent, especially with memory management. It will be a shorter learning process than the previous, and my productivity should increase as the C portion of the class continues.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

#define MAX_BUFFER_SIZE 1024

void input_cmd(char input[MAX_BUFFER_SIZE])
{
  char c;
  int x = 0;
  do {
    c = getchar();
    //putchar (c);
    input[x] = c;
    x++;
  } while (c != '\n');
  input[x] = '\n';
}

int main()
{

  char input[MAX_BUFFER_SIZE];
  char *filename;
  char *argv[MAX_BUFFER_SIZE];

  int status = 0;
  while(status != -1)
    {
      printf("?: ");

      input_cmd(input);

      //parse_cmd(input, filename, argv);
      filename = input;
      int x = 0;
      filename = input;
      argv[0] = input;
      while( !(input[x] == ' ' || input[x] == '\n') )
  {
    x++;
  }
      int y = 1;
      while(input[x] != '\n')
  {
    if(input[x] == ' ')
      {
        input[x] = '\0';
        x++;
        argv[y] = &input[x];
        y++;
      }
    else
      x++;
  }
      input[x] = '\0';
      int argvsize = y + 1;
      y = 0;
      while(y < argvsize)
  {
    y++;
  }
  if(filename[0] == 'e' && filename[1] == 'x'
   && filename[2] == 'i' && filename[3] == 't')
  {status = -1;}
      else
  {
    int pid = fork();
    if(pid == 0)
      {
        execvp(filename, argv);
      }
    else
      {
        //parent
        waitpid(pid, NULL, 0);
      }
  }
    }
return(0);
}
```