Creating A Shell

From OSDev Wiki

This page is a work in progress and may thus be incomplete. Its content may be changed in the near future.

Hello, this is a guide for writing a simple shell. Our shell with need a few things:

```
1. A working bootloader/GRUB
2. An IDT and IRQ handler
3. A GDT
4. A keyboard driver
5. A working text input/output system.
6. A set of string editing functions
```

First we need to make a file, shell.c, this file will contain all our functions and shell managers. The first function we need to write is our initialization function then our actual shell function.

```
/* shell.c */
void init_shell()
{
}

void shell()
{
}
```

Now you need to add "init_shell()" to your setup routine in your kernel.c file or whatever you named it, followed by a while loop running shell until the boolean "exit_status" is true.

Ok, now we need to write a few things. The first is command table, or a way to store our commands. I do this with a structure as follows:

```
/* shell.h */
#define MAX_COMMANDS 100

typedef struct
{
    char *name;
    char *description;
    void *function;
} command_table_t;
```

Now we have a way to store commands, but how do we use this? Well lets add a few variables to shell.c:

```
/* shell.c */
command_table_t CommandTable[MAX_COMMANDS];
```

This goes right above the function definitions. The first is a way to keep track of how many commands there are. The second is our command table. Ok, so now we have a "Command Table", but how to we access it? Well in order to add a command we need a function to do it. Add this below the other functions:

```
/* shell.c */
void add_new_command(char *name, char* description, void *functi
{
    if(NumberOfCommands + 1 < MAX_COMMANDS)
    {
        NumberOfCommands++;

        CommandTable[NumberOfCommands].name = name;
        CommandTable[NumberOfCommands].description = description
        CommandTable[NumberOfCommands].function = function;
    }
    return;
}
```

Ok, now we are getting somewhere. We can add a command. But what about getting command line input? Thats up to our "shell()" function. But before we code that, lets add another variable: "char* input_string". Now for the shell function:

```
/* shell.c */
void shell()
{
    puts("\nMy_Prompt>");
    gets(input_string);

    void (*command_run)(void);
}
```

Ok, now we can get a string. But what about finding what command the user typed? Well we rely on old trusty "findCommand()" function for that one:

```
/* shell.c */
int findCommand(char* command)
{
    int i;
    int ret;
```

```
    for(i = 0; i < NumberOfCommands + 1; i++)
    {
        ret = strcmp(command, CommandTable[i].name);

        if(ret == 0)
            return i;
        else
            return -1;
    }
}
```

Now we need to add a little something right below "gets(input_string)", but above "void (*command_function) (void)":

```
/* shell.c */
int i = findCommand(input_string);

if(i >= 0)
{
    command_function = CommandTable[i].function
    command_function();
}
else
{
    return;
}

return;
```

Now in our "init_shell()" function we need to add a few commands:

```
/* shell.c */
add_new_command("help", "You code this one.", help_command);
add_new_command("", "", empty_command);
```

Now in our shell.h file:

```
extern void add_new_command();
extern void help_command();
extern void empty_command();
```

Now you need to code a void help_command function and a empty_command that manages null input, all it needs is a definition:

```
void empty_command()
{
}
```

An idea is to loop through all the command table and print the command name and description - for the help command.

Whats left?

Add support for filesystem commands, a clear screen command, I don't know. But enjoy developing :)

- wxwsk8er