Lab Assignment 1: Shell and System Calls

I. Code Organization

- Each module is separated to header file and its implementation where any function that will get used by other modules will be defined in the header file and the header file contains the full documentation of the function. An implementation file might contain helper function where their documentation is above their prototype in the implementation file.

- Modules in the project:
  - ***command _parser:*** module responsible about parsing the command and executing it.
  - ***commands:*** module responsible about the terminal own implemented commands.
  - ***environment:*** module responsible about setting up the environment variables in the local variables table.
  - ***file_processing:*** module responsible about handling with the files, meaning opening , closing and returning handlers to them.
  - ***history:*** module responsible about creating the internal history table and managing it and managing the history file.
  - ***shell_constants:*** Only a header which contains some constants used throughout the code.
  - ***signal_handler:*** module responsible about registering and managing the actions taken on a certain signal.
  - ***string_operations:*** small helper module containing some useful string manipulation functions and a scanning string function.
  - ***variables:*** module responsible about maintaining environmental and local variables and organizing them and any operation on them.

II. Main Functions

- lookup_variable :

Implemented in variables.c and defined in variables.h. This function is responsible for looking for the values of the variables used in terminal.

- set_variable :

Implemented in variables.c and defined in variables.h. This function is responsible of adding/updating the variables value in the local table and updating them in the environment if they are environmental variables.

- read_line :

Implemented in string_operations.c and defined in string_operations.h. This function is responsible of reading the input dynamically, increasing the size of buffer that the data is read into until all user input has been read from a user defined data stream.

- start_signal_handlers :

Implemented in signal_handler.c and defined in signal_handler.h. This function is responsible of registering a function handler to handle the sigchld event, taking the suitable action when a child is terminated.

- add_to_history :

Implemented in history.c and defined in history.h. This function is responsible of taking a command and adding it if appropriate to the history log file and the internal history table.

- parse_command :

Implemented in command_parser.c and defined in command_parser.h. This function is responsible of defining the number of parameters, getting a list of parsed parameters having full replaced the variables and processed the parameters as suitable, defining the command type and if it is a background command.

- execute_command :

Implemented in command_parser.c and defined in command_parser.h. This function is responsible of executing the command using the knowledge extracted by parse_command.

- shell_loop :

Implemented and documented in main.c. This is considered the core of the shell as it's where we read the input and begin calling the different modules to process it as suitable.

III. Compilation & Running Guide

- Compilation :
  - Go to the folder of the project <lab1_46>. You should see src folder, include folder and makefile.
  - Open a terminal in this directory.
  - Type the command 'make' and run it.
  - It should end successfully having created two folders in the directory obj and bin.
  - The Shell should be in bin folder.
  - If for any reason you want to clean the directory and compile again, enter the command 'make clean' which deletes all the object files and the output program. Then rerun 'make'.

- Running the shell :
  - After compiling the project, go to the bin folder.
  - Open a terminal in this directory.
  - To run in interactive mode, enter the command './Shell'
  - To run in batch mode, enter the command './Shell <filename>' where filename is the name of the batch file.

IV. Program Features

- Capable of running large scripts due to efficiently releasing allocated memory after being done using it.
- Dynamically growing variables and history table to accommodate any number of variables or history entries.
- Support for double and single quotation.
- Support for ~ and ~username in full capacity.
- Support for export and correctly updating the environment variables or the exported environment variables. (Meaning printenv will run just fine).
- Auto removal of zombie processes.
- Capable of reading large input lines and filtering them only for input of suitable length.
- Support for echo, export, cd & variable assignment commands beside the support for the general linux commands.
- Export supports only exporting one variable at a time either in format of ‘export x’ or ‘export x=10’.
- Support for the history command.
- Support of background and foreground operations.
- Support of mid command comments like ‘ls #this is a comment’ and comments in general.
- Support for batch running mode.
- Flexible makefile that is easy to modify to compile any c project.

V. Sample Runs

1. *Sample run 1 :*

```
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/shell/bin$ ./shell
Shell > pwd
/home/amrnasr
Shell > cd MyFiles/projects/OS/shell
Shell > pwd
/home/amrnasr/MyFiles/projects/OS/shell
Shell > mkdir ../lab1_46
Shell > cd ..
Shell > ls
lab1_46  shell
Shell > cp -r shell/src lab1_46/src
Shell > cp -r shell/include lab1_46/include
Shell > ls lab1_46
include  src
Shell > cp shell/makefile lab1_46/
Shell > ls lab1_46
include  makefile  src
Shell > cd lab1_46
Shell > ls include
command_parser.h  commands.h  environment.h  file_processing.h  history.h  shell_constants.h  signal_handler.h  string_operations.h  variables.h
Shell > ls src
command_parser.c  commands.c  environment.c  file_processing.c  history.c  main.c  signal_handler.c  string_operations.c  variables.c
Shell > cat makefile
# Final program directory
TARGET_DIR = ./bin
# Final program name
TARGET = Shell
# Header files directory
HEADER_DIR = ./include
# Header files
HEADERS = $(wildcard $(HEADER_DIR)/*.h)
# Source files directory
SOURCE_DIR = ./src
# Source files
SOURCES  = $(wildcard $(SOURCE_DIR)/*.c))
# Object files directory
OBJECT_DIR = ./obj
# Add prefix to the directory, substitue any .c with .o for all .c files.
OBJECTS = $(patsubst $(SOURCE_DIR)/%.c,$(OBJECT_DIR)/%.o,$(wildcard $(SOURCE_DIR)/*.c))
# Compiler to use.
COMPILER = gcc
# Include headers compiler flag.
IHCF = -I $(HEADER_DIR)
# Default action.
all: directories $(TARGET)
# Print any variable.
print-%  :
        @echo $* = $($*)
# Check our directory.
directories:
        # make directory if it isn't available.
        -mkdir -p $(OBJECT_DIR)
        # Make directory if it isn't available.
```

2. *Sample run 2 :*

```
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/shell/bin$ ./Shell
Shell > pwd
/home/amrnasr
Shell > cd MyFiles
Shell > cd projects
Shell > pwd
/home/amrnasr/MyFiles/projects
Shell > cd ~/MyFiles/projects/OS/lab1_46
Shell > pwd
/home/amrnasr/MyFiles/projects/OS/lab1_46
Shell > ls
include  makefile  src
Shell > make
# make directory if it isn't available.
mkdir -p ./obj
# Make directory if it isn't available.
mkdir -p ./bin
# obj/environment.o left side of condition, src/environment.c first operand of the right side
gcc -c -o obj/environment.o src/environment.c
# obj/signal_handler.o left side of condition, src/signal_handler.c first operand of the right side
gcc -c -o obj/signal_handler.o src/signal_handler.c
# obj/main.o left side of condition, src/main.c first operand of the right side
gcc -c -o obj/main.o src/main.c
# obj/command_parser.o left side of condition, src/command_parser.c first operand of the right side
gcc -c -o obj/command_parser.o src/command_parser.c
# obj/variables.o left side of condition, src/variables.c first operand of the right side
gcc -c -o obj/variables.o src/variables.c
# obj/string_operations.o left side of condition, src/string_operations.c first operand of the right side
gcc -c -o obj/string_operations.o src/string_operations.c
# obj/commands.o left side of condition, src/commands.c first operand of the right side
gcc -c -o obj/commands.o src/commands.c
# obj/file_processing.o left side of condition, src/file_processing.c first operand of the right side
gcc -c -o obj/file_processing.o src/file_processing.c
# obj/history.o left side of condition, src/history.c first operand of the right side
gcc -c -o obj/history.o src/history.c
# Compile the final program.
gcc -o ./bin/Shell obj/environment.o obj/signal_handler.o obj/main.o obj/command_parser.o obj/variables.o obj
Shell > ls
bin  include  makefile  obj  src
Shell > ls bin
Shell
Shell > ./bin/shell
No matching command found...
Shell > ./bin/Shell
Shell > exit

Shell > exit
```

3. *Sample run 3 :*

```
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/lab1_46/bin$ ./Shell
Shell > x=5
Shell > echo $x
5
Shell > x=~
Shell > echo $x
/home/amrnasr
Shell > x=~/MyFiles/"no    file"
Shell > echo $x
/home/amrnasr/MyFiles/no file
Shell > echo "$x"
/home/amrnasr/MyFiles/no    file
Shell > echo '$x'
$x
Shell > echo  "~"
~
Shell > ls
Android  Assignment_2.pdf  cs333  CS333F17Lab1.pdf  Desktop  Documents  Downloads  eclipse
Shell > cd ~postgres
Shell > pwd
/var/lib/postgresql
Shell > cd ~amrnasr
Shell > pwd
/home/amrnasr
Shell > echo $y

Shell > echo "ths is    a   successfull "'$TEST'" so keep the space"
ths is    a   successfull $TEST so keep the space
Shell > y=$x/5
No matching command found...
Shell > echo $x
/home/amrnasr/MyFiles/no file
Shell > y="$x"/5
Shell > echo $y
/home/amrnasr/MyFiles/no file/5
Shell >
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/lab1_46/bin$
```