SHELL SIMULATOR

OS ASSIGNMENT

By Nabeel Ahmed

18024

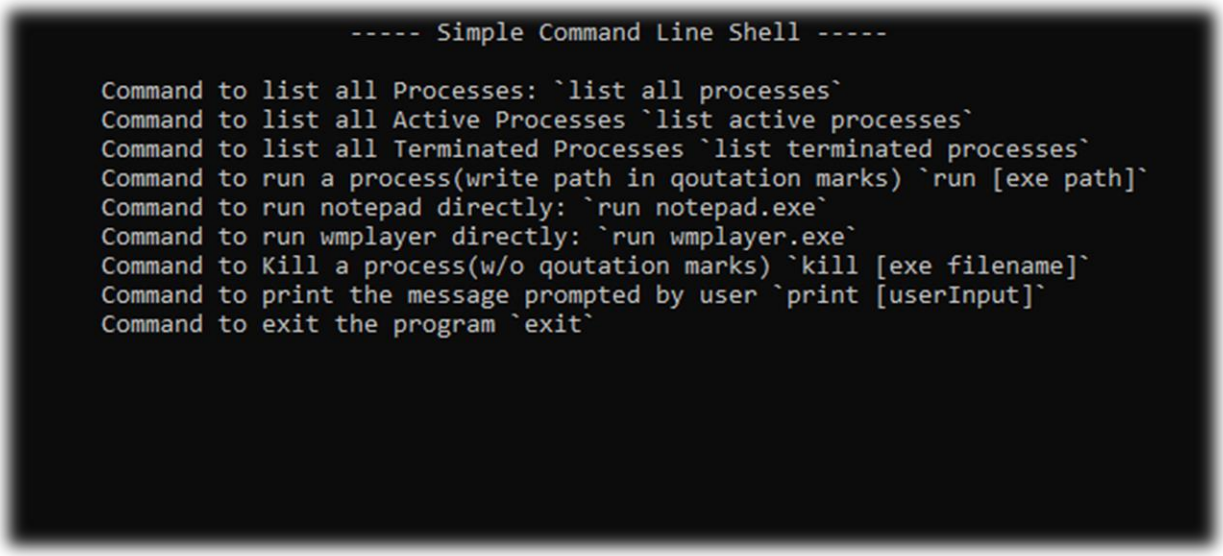


Table of Contents

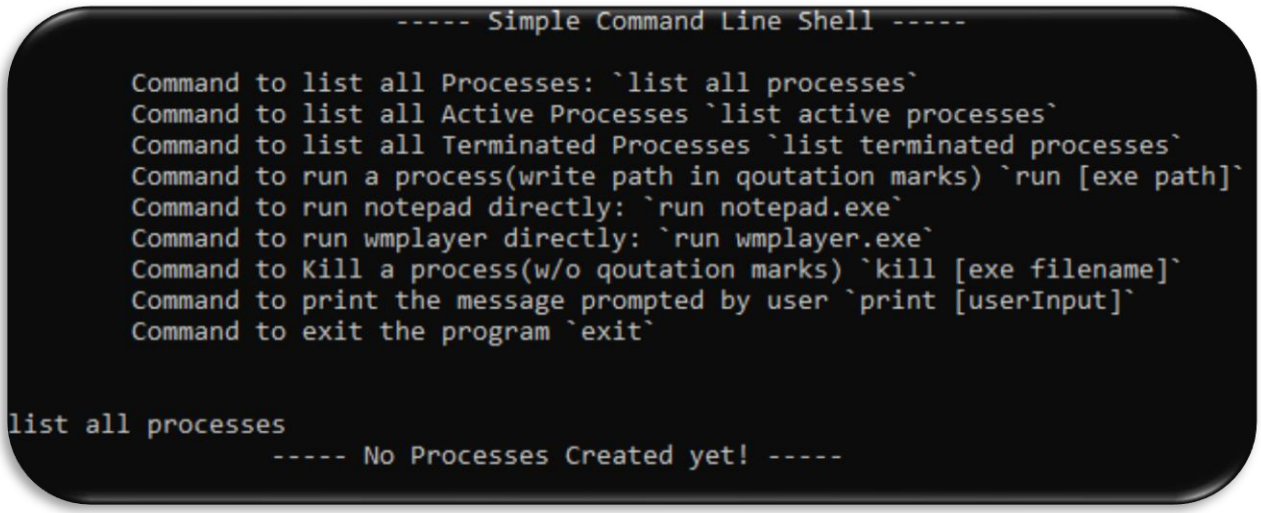
Functionalities of Simple Shell / Help (User Manual)

The shell simulator which I created has the following functionalities:

- `list all processes`
- `list active processes`
- `list terminated processes`
- `run [exe path]`
- run notepad directly: `run notepad.exe`
- run wmplayer directly: `run wmplayer.exe`
- `kill [exe filename]`
- `print [userInput]`
- `exit`

Let us give a look at what each command does.

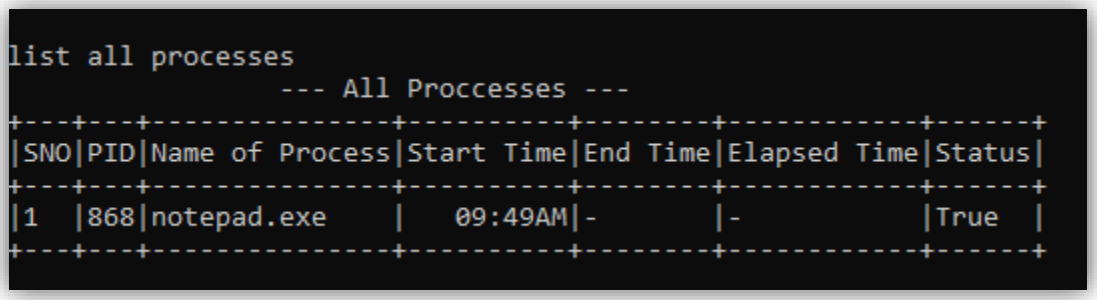
1. `list all processes`**:** This command lists all the processes which you have created, or you have terminated.
    - If no process has been created and user runs this command then it will print a message “No Processes Created yet” as shown below.



    - Similarly, if a process has been created by the user then it will display a table which will have following properties:
        - **Sno**: It is just the counter which keeps track how many processes we have created.
        - **PID**: It is the unique identifier given to each process when user runs the process.

- **Name of Process**: It tells us the name of process we have created eg. If I have created notepad then in this column it will display “notepad.exe”.
- **Start Time**: It displays the time which was formatted to local time to display when this process got created.
- **End Time**: It displays the time which was formatted to local time to display when this process was killed/terminated.
- **Elapsed Time**: It displays the time difference between start and end time formatted nicely in HH:MM:SS.
- **Status**: Finally this displays the status of the process i.e. whether the process is currently active “True” or is inactive “False”.

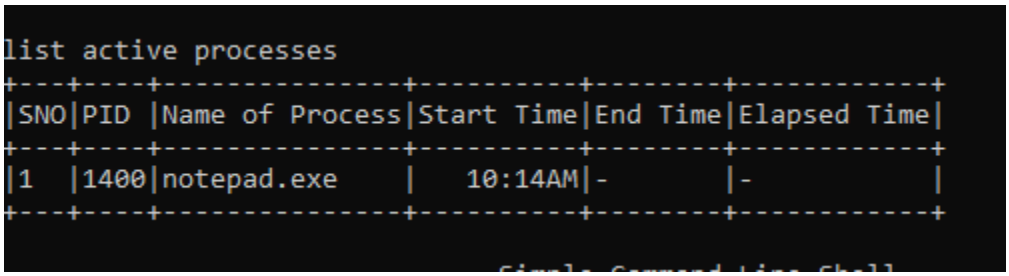
Below is the screenshot of the table which displays that notepad was created, its other characteristics and shows that the process is currently active.



2. `list active processes`**:** This command displays all the processes which were created/run by user but was not terminated.
    - If there are no active processes then a message will be printed that “**No active processes**”.
    - Similarly, if the user has not created any process then it will print “**No Processes Created yet**”.
    - If a process was created by the user and has not been terminated yet, so on running this command it will display the table with following properties:
        - **Sno**: It is just the counter which keeps track how many processes we have created and tells what the counter of the process is.
        - **PID**: It is the unique identifier given to each process when user runs the process.
        - **Name of Process**: It tells us the name of process we have created eg. If I have created notepad then in this column it will display “notepad.exe”.
        - **Start Time**: It displays the time which was formatted to local time to display when this process got created.
        - **End Time**: Since the process is currently running and active hence I have displayed hyphen “-” in this column.

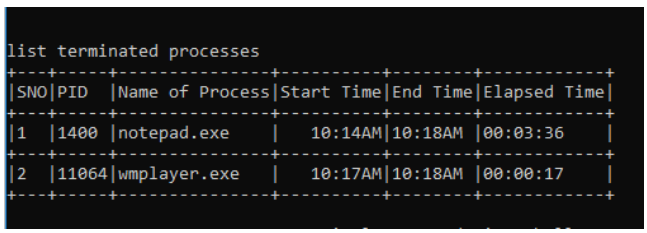
- **Elapsed Time**: Similarly, as process has not been terminated yet so we cannot calculate elapsed time yet therefore hence I have displayed hyphen "-" in this column.

Below is the screenshot of the current active process.



3. `list terminated processes`: This command displays all the processes which were terminated by user.
    - If there are no terminated/inactive processes then a message will be printed that "**No Terminated processes**".
    - Similarly, if the user has not created any process then it will print "**No Processes Created yet**".
    - If a process has been terminated by user, so on running this command it will display the table with following properties:
        - **Sno**: It is just the counter which keeps track how many processes we have created and tells what the counter of the process is.
        - **PID**: It is the unique identifier given to each process when user runs the process.
        - **Name of Process**: It tells us the name of process we have created eg. If I have created notepad then in this column it will display "notepad.exe".
        - **Start Time**: It displays the time which was formatted to local time to display when this process got created.
        - **End Time**: It displays the time which was formatted to local time to display when this process got killed/terminated.
        - **Elapsed Time**: It displays the time difference between start and end time formatted nicely in HH:MM:SS.

Below is the screenshot of the terminated processes.



4. `run [exe path]`**:** This command will call windows API and will create the process which user has specified path of. E.g. *run "C:\Program Files (x86)\Windows Media Player\wmplayer.exe"* , The entire path has to be in **quotation marks**.
    - If the user entered anything else after specifying path, a message will be printed saying *"Too many Arguments. Follow Manual!"* as shown below:

```
run "C:\Program Files (x86)\Windows Media Player\wmplayer.exe" aa
Too many Arguments. Follow Manual!
```

    - So if user entered the command correctly then a new process will be created, and a window of the program will pop up.

5. `run notepad.exe` or `run wmplayer.exe`: In order to provide quick testing of the program I have hardcoded the full directory of notepad and wmplayer executable, so if a user just wants to check functionality he can just pass *'run notepad.exe'* (No need to write in quotations). This will concatenate the name of exe file with the hardcoded path and will call *Windows API to create the process*.
    - Now, there has also been some checks here e.g. if user just tries to run a process whose full path is not already hardcoded. It will print a message to the user that *"Provide complete Path to the Process or refer to the manual ."* as shown below.

```
run calculator.exe
Provide Complete Path of the Process or refer to the manual.
```

    - Similarly, if a user provides more than 2 command line arguments, an error message will be printed *"Too many arguments. Follow Manual"*.

6. `kill [exe filename]`**:** This command will accept ***name of executable file only*** not entire path and will kill all the processes of this type which were created and are currently running/active. E.g. *kill notepad.exe* => This will terminate all instances of notepad that were created by the user and were active that time.
    - There are few edge cases which have been checked here as well.
        - If a user tries to kill the process which was never really created, it will print a message that ***"The process has not been created yet.".***
        - Similarly, if a user to kill already killed process then the program will print the message ***"The process is not active."***

7. `print [userInput]`**:** This command just prints on console whatever user types after print command. E.g. *print hello world* would display ***"hello world"***

```
print hello world
hello world
```

8. `exit`**:** This will terminate the entire program and user will exit from the ***Simple Shell.***

Code Architecture

In the implementation **Simple Shell** I have created global list of Processes and each process has following attributes:

```
// Code for Process structure
struct Process
{                                   //=== Code Explanation =====//
   int num;                         // to maintain serial no. of process.
   PROCESS_INFORMATION pi;  // it contains all the information related to single process.
   string procName;                 // it stores name of the process e.g. notepad.exe.
   bool isActive;                   // it maintains a flag that whether a process is active or not.
   time_t start;                    // it stores the time when this process got created.
   mutable time_t endt;             // it stores the time when this process got terminated.
   mutable time_t elapsedt;         //  it stores the time between start and kill using difftime.
};
// global list decalred
list<Process> procList;
```

- So initially the list is empty, when a user run any process that process is pushed into the list with all its characteristics specified in struct process. On process creation its flag ***isActive will be true***.
- The reason for keeping the Boolean flag **isActive** is to keep track of all active and inactive processes from just a single list.

Table Creation of `all` | `active` and `terminated` processes.

In order to create a well formatted table I have make use of a header file named *"<table.h>"*. It provided the ease of organizing the information in well formatted table.

- For listing all processes [**`list all processes'**] we iterated over the entire process list. If the list is not empty then it displayed the information on the table. However if list was empty a message was printed that "No Process Created yet!"
- Here we have also formatted start time by using ctime library and make use of strftime function to format time in readable format.

If there is a terminated process in the list then we have similarly formatted end time and elapsed time as well

Below is the snapshot of time formatting code.

```
strftime(my_time, sizeof(my_time), "%I:%M%p", localtime(&it->start));
t.add(my time);
```

- For listing all Active processes [`list active processes`] we again iterated over the same list but this time an additional check needs to be done i.e. whether the process is active or not by checking isActive flag. If the process is active then only we displayed the process in the table otherwise we continue to the next process in the list.
    - Similarly, if there are no currently active processes, a message appears saying ***"No Active processes".***
    - Another check which we made was when a user tries to view an active process before creating any process, this prints message on screen that ***"No Processes created yet!".***
    - Here also we did time formatting for just start time for all active processes and end and elapsed time were filled with hyphen as the process is currently running.

- For listing all Terminated processes [`***list terminated processes***`] we again iterated over the same list with an additional check that whether the process is active or not by checking isActive flag. If the process is inactive then only we displayed the process in the table otherwise we continue to the next process in the list.
    - Similarly, if there are no currently terminated processes, a message appears saying ***"No Terminated processes".***
    - Another check which we made was when a user tries to view all terminated process before creating any process, this prints message on screen that ***"No Processes created yet!".***
    - As in list active processes here also we did time formatting but not only for start time but also end time for all terminated processes, we also calculated elapsed time using ***difftime(endt, start),*** which return time for execution of a process. This time was also formatted in HH:MM:SS using **strftime**.

```
int final_time = difftime(it->endt, it->start);
int hour = final_time/HOUR;
int second = final_time % HOUR;
int minutes = second/MIN;
 second = second % MIN;

string h = hour < 10 ? "0" + to_string(hour) : to_string(hour);
string m = minutes < 10 ? "0" + to_string(minutes) : to_string(minutes);
string s = second < 10 ? "0" + to_string(second) : to_string(second);
t.add(it->endt != NULL ? h + ":" + m + ":" + s : "-");
t.add("False");
```

"Code to calculate elapsed time and formatted using strftime"

- For running a process **[`run [exepath]`]** we make use of Windows **createProcess() API** and along with other parameters. We specify the full path of executable file in quotation marks. Then the API does the rest it creates and executes the process. On successful creation of the process we push it into the global list of processes.

```
void createProc(string name)
{
    HANDLE hProcess;
    HANDLE hThread;
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    DWORD dwProcessId = 0;
    DWORD dwThreadId = 0;
    ZeroMemory(&si, sizeof(si));
    ZeroMemory(&pi, sizeof(pi));

    LPCSTR path = name.c_str();



    BOOL bCreateProcess = NULL;
    bCreateProcess = CreateProcess(
        path,
        NULL,
        NULL,
        NULL,
        FALSE,
        0,
        NULL,
        NULL,
        &si,
        &pi);

    if (bCreateProcess == FALSE)
    {
        cout << "Create Process Failed & Error No -" << GetLastError() << endl;
    }

    cout << "Process Created Successfully!" << endl;

    sno++;
    //time (&start);
    struct Process p_struct = {sno, pi, getFilename(name), true, time(0), '-'};
    procList.push_back(p_struct);


    CloseHandle(pi.hThread);
    CloseHandle(pi.hProcess);
```

'Above snapshot shows the function to create Process.'

- For killing a process created already **[`kill exefilename`],** we just specify the exe filename with '.exe' extension. It will loop over the entire process List and kill all instances of the process. This is done by calling Windows API **TerminateProcess().**

Below is the snapshot of the code where it calls windows API and terminate the process. We also need to update the flag in the process list to false as process is not in running state.

```
PROCESSENTRY32 pEntry;
pEntry.dwSize = sizeof(pEntry);
BOOL hRes = Process32First(hSnapShot, &pEntry);
while (hRes)
{
    if (strcmp(pEntry.szExeFile, filename) == 0)
    {
        HANDLE hProcess = OpenProcess(PROCESS_TERMINATE, 0,
                                      (DWORD)pEntry.th32ProcessID);
        if (hProcess != NULL)
        {
            TerminateProcess(hProcess, 9);
            CloseHandle(hProcess);
        }
    }
    hRes = Process32Next(hSnapShot, &pEntry);
}
std::list<Process>::iterator it;
for (it = procList.begin(); it != procList.end(); ++it){
    if(it->procName == filename){
        it->isActive= false;
        it->endt = time(0);
        it->elapsedt = difftime(it->endt,it->start);
    }
```

Creating User Manual

- Finally I created a user manual by using some string manipulations to get user input from command line and after making all necessary checks, calling the required function to execute the task prompted by user.
- All necessary validation checks have also been made incase a user enters an invalid input or enters something my program is not coded to answer.

Screenshots of Program

User manual:

```
                ----- Simple Command Line Shell -----

 Command to list all Processes: `list all processes`
 Command to list all Active Processes `list active processes`
 Command to list all Terminated Processes `list terminated processes`
 Command to run a process(write path in qoutation marks) `run [exe path]`
 Command to run notepad directly: `run notepad.exe`
 Command to run wmplayer directly: `run wmplayer.exe`
 Command to Kill a process(w/o qoutation marks) `kill [exe filename]`
 Command to print the message prompted by user `print [userInput]`
 Command to exit the program `exit`
```

List All Processes:

```
list all processes
              --- All Proccesses ---
+---+-----+---------------+----------+--------+------------+------+
|SNO|PID  |Name of Process|Start Time|End Time|Elapsed Time|Status|
+---+-----+---------------+----------+--------+------------+------+
|1  |7384 |notepad.exe    |   01:05PM|01:06PM |00:00:46    |False |
+---+-----+---------------+----------+--------+------------+------+
|2  |12596|wmplayer.exe   |   01:06PM|-       |-           |True  |
+---+-----+---------------+----------+--------+------------+------+
|3  |5992 |chrome.exe     |   01:06PM|-       |-           |True  |
+---+-----+---------------+----------+--------+------------+------+
```

List All Active Processes:

```
list active processes
+---+-----+---------------+----------+--------+------------+
|SNO|PID  |Name of Process|Start Time|End Time|Elapsed Time|
+---+-----+---------------+----------+--------+------------+
|2  |12596|wmplayer.exe   |   01:06PM|-       |-           |
+---+-----+---------------+----------+--------+------------+
|3  |5992 |chrome.exe     |   01:06PM|-       |-           |
+---+-----+---------------+----------+--------+------------+
```

List Terminated Process

```
list terminated processes
+---+----+---------------+----------+--------+------------+
|SNO|PID |Name of Process|Start Time|End Time|Elapsed Time|
+---+----+---------------+----------+--------+------------+
|1  |7384|notepad.exe    |   01:05PM|01:06PM |00:00:46    |
+---+----+---------------+----------+--------+------------+
```