

LIDO Command Interpreter (build 100) 13/03/2002

Brief Introduction

I was quite unsuccessful browsing through numerous sites looking for a Linux shell to use with DOS. All I found was a DOS shell to use with Linux. I found one though, it was a Korn Linux Shell for DOS, but that program was not what I wanted. It was meant to be used with real mode DOS. I was using the 'so called' DOS in XP.

The name LIDO means LINux DOs. LIDO name concept it from a company called Jobstreet. Jobstreet has an award winning software called LINA. The concept is it sounds like a persons name and I find it cute.

Architecture

LIDO Engine

LIDO uses a simple engine to handle loading, interpretation, and logic of the instructions. The LIDO Engine can simply be used by including the lido_engine.h file in program. A class is used for the engine because:

1. I can encapsulate all my code underneath.
2. Another developer doesn't get confused trying to use the code.
3. Provides data hiding so the engine does not become unstable from outside interference.
4. Provides a simple interface for other developers.
5. Other developers can incorporate the engine into their programs easily.

The LIDO engine lets the program to be easily updated with new instructions for new Linux commands.

LIDO Engine structure

The LIDO engine is coded as the class lido_engine. lido_engine needs four predefined values for it to function properly.

```
#define noc 100           //number of commands
#define noch 80          //length of commands
#define buflen 500      //length of execution buffer
#define nop 10           //number of parameters
```

Number of commands

- o is the maximum amount of commands that is made available
- o #define noc {intval}

Length of commands

- o is the maximum length of each command
- o #define noch {intval}

Length of execution buffer

- o is the length of the execution buffer where the full parsed dos command syntax resides. Its ratio is roughly 1:4 to noch:buflen.
- o #define buflen {intval}

Number of parameters

- o maximum amount of parameters supported for each command
- o #define buflen {intval}

Member functions

lido_engine::lido_engine()

The constructor lido_engine() is used to initialize some of the data members. It is also the function that loads, reads and decodes the instructions from the instruction file. Displays author information and version. Return data type is void and parameters, void. The function reads the first two lines of the instruction file, then it checks whether the particular command contains any parameter. If it does, then it reads the next four lines, two it treats as header lines and the final two as command lines. If the particular command does not contain any parameter then it reads the next two lines as command lines. Finally, it checks for the blank line and continues the cycle until the end of the file.

int lido_engine::exec(char tmp[])

This member function is the one of the most important. It does all the manipulation of the commands and logic. Return data type is integer that returns 0, and parameter is char data type where the Linux command entered by the user is passed.

int lido_engine::cmdprompt()

This function gets the value from the data member, prompt[noch] and changes the command prompt to that value. Return data type integer that returns 0, and does not accept any parameters. So to change the command prompt, you have to change the data member prompt[noch].

int lido_engine::listcmd()

This function displays a list of commands available. It is invoked by the exec() function once the user enters the command 'listcmd'. Return data type is integer that returns 0 and parameter data type is void.

Data members

char

linux[noc] [noch]

Stores all linux commands from the instruction file

dos[noc] [noch]

Stores all dos commands from the instruction file

linuxheader[noc] [noch]

Stores all linux command headers from the instruction file

dosheader[noc] [noch]

Stores all dos command headers from the instruction file

func[noc] [noch]

Stores special function type for each command

prompt[noch]

Command prompt value

tmpheader[noch]

Temporary header storage for manipulation

int

nov[noc]

Stores the number of variables each command contains

cmd_counter

Value for the number of commands loaded

static

static bool enable_doscmd

Switch, whether to enable the use of dos commands in LIDO (1=enable|0=disable)

LIDO Instruction file

The instruction file enables users to update the program easily. Simply by updating to lido.ins file you can give the program new instructions on how to handle new commands. This opens up the door for the LIDO engine to be used to convert commands from different operating systems.

The power and functionality of LIDO is in its instruction file. Many things can done with the instruction file. Although the engine contains a limited amount of built-in functions, but by using the intruction file lots of tasks can be accomplished.

The instruction file concept was used in this project because:

1. Saves function coding time.
2. Here I can just code a few function and build the rest easily by just coding the instruction file. Its concept is something like the generic function concept.
3. Expandability, where new functions can be created by anyone, even though they can't program in C++.
4. New functions can be added without recompiling.
5. Protect my code for distribution

Instruction file format

Each command is separated by a blank line. The instruction file must be in the same directory as the executable file and must be named lido.ins.

example of an instruction file:

```
default
0
wait
pause

chprompt
1
chdir
cd
chdir %1
cd %1

default
2
cp
copy
cp %1 %2
copy %1 %2
```

The first line contains the special function name. There are two special functions so far.

1. *default* – checks for the command in the instruction file and executes it. 0 – nop, parameters supported..
2. *chprompt* – changes the command prompt. Needs at least one parameter. The first parameter passed is the value of the command prompt you want to change to.

The second line contains the number of parameters for the particular command. Ranges from 0 to nop.

The third and fourth lines are a little different. If in line two you specified any value besides 0 then the third line contains the linux command header and the fourth line contains the dos command header of the linux and dos command you are going to specify in the fifth and sixth line. A header is the name of the command without the parameters. For example:

```
linux command : ls %1
linux header  : ls

dos command   : dir %1 %2
dos header    : dir

command       : echo hello world
header        : echo
```

The fifth and sixth line should contain the linux and dos equivalent command. The end.

If in the second line 0 was specified, then the third line contains the linux command and fourth line contains its dos equivalent command. The end

Linux and Dos command line:

To specify the first parameter we put a %1 after the command , second we put %2 and so on.

example:

```
command %1 %2
echo %1 %2
ls %2
dir %1
```

Instruction file format :

line 1: special function name {default | chprompt}
line 2: number of parameters for this command {0 to nop}
line 3: linux header {if line 2 != 0} / linux command {if line 2 = 0}
line 4: dos header {if line 2 != 0} / dos command {if line 2 = 0}
line 5: linux command {if line 2 != 0}
line 6: dos command {if line 2 != 0}
line 7: blank line

Pros

- ✓ The engine has been coded quite efficiently from my point of view, although there are still some changes to be made.

- ✓ No low level coding so can be compiled in other platforms
- ✓ ANSI C++ coding so can be compiled in other platforms

Bugs, Problems & Comments

I have encountered some problems mainly because lack of coding time.

- !! Command to change directory doesn't work properly. The reason is that every command executed loads a new instance of the command.com interpreter and destroys it once completed. So if I've executed the change directory command and want to executed another command on that path, I can't because it returns me to the default path where LIDO resides. I'm working on a way to solve this problem by using a history file.

- !! Command parameters have to follow a particular sequence and can't be interchanged. This problem can be easily rectified by further string manipulation but has to wait for the next release. For example,


```
cmd %1 %2      (works)
cmd %2 %1      (doesn't give you the result you want, but still works)
```

- !! Vulnerable to buffer overflow problems for commands and the buffer.

- !! No low-level coding

FUTURE OF LIDO

I plan to distribute the LIDO Engine as Open Source code.

LIDO has slight potential to be used as training software or simulation software. It can enable small institutions that don't have enough resources, be it hardware or money, to simulate certain amount of the Linux shell for use by their students. People who also want to train using Linux command can do so.

I also plan to release instruction files to simulate other platforms like, VMS, Unix, CP/M and so on. But before all of this, I want to properly code the engine and fix some bugs.

LIDO USER MANUAL

Commands available

Note - to display help or options for a command type the command name followed by /?.

- ls : list directory
- ls /a : shows hidden files
- ls /R : shows all files
- pwd : shows current working directory
- cp : copy file
- mv : move or rename file
- rm : delete or erase file
- ed : unix editor
- echo : echo to screen
- cat : display ASCII file contents
- mkdir : make directory
- rmdir : remove directory
- clear : clear screen
- grep : search function

Redirectors and pipes

- | : pipes can be used to redirect output to another program
- | more : to pause long outputs that do not fit the screen
- > : output redirection to device and files
- < : input redirection from devices and files

SOURCE CODE (lido.h)

```

//LIDO Engine build 100
//Copyright 2002 nmr

#include <iostream>
#include <fstream>
#include <string>
#include <stdlib.h>

using namespace std;

#define noc 100 //number of commands
#define noch 80 //length of commands
#define buflen 500 //length of execution buffer
#define nop 10 //number of parameters

class lido_engine
{
private:

    //declarations of all the main variables needed for the engine
    char linux[noc][noch],dos[noc][noch],linuxheader[noc][noch],dosheader[noc][noch];
    char prompt[noch],tmpheader[noch],func[noc][noch];
    int nov[noc],cmd_counter;
    static bool enable_doscmd;

public:

    lido_engine();
    int exec(char tmp[]);
    int cmdprompt();
    int listcmd();

};

bool lido_engine::enable_doscmd=0; //enable or disable dos commands (1=enable|0=disable)

lido_engine::lido_engine()
{

    strcpy(prompt,"lido>"); //set the default command prompt display to "root>"
    cmd_counter=0;
    int tmp2=0;
    ifstream file1; //initializing file stream

    file1.open("lido.ins",ios::in); //open file stream for input and output

    for(int i=0;i<noc;++i) //initializing nov 0 to 99 to value 0 else error occurs later on
        nov[i]=0;

    for(;!file1.eof();++tmp2) //extract data from file text.txt used

```

```

interpretation
//for dynamic
{
    file1.getline(func[tmp2],noch);

    file1>>nov[tmp2];
    file1.get();

    if(nov[tmp2]!=0) //if parameter for command exist then extract header
    {
        file1.getline(linuxheader[tmp2],noch);
        file1.getline(dosheader[tmp2],noch);
    }

    file1.getline(linux[tmp2],noch);
    file1.getline(dos[tmp2],noch);
    file1.get();

    cmd_counter++;
    //cout<<dos[tmp2]<<" "<<linux[tmp2]<<"
"<<nov[tmp2]<<endl<<cmd_counter<<endl; //debug001
    }

    cout<<"LIDO Command Interpreter build 100"/*\nNavyn Muhammed Rajoo"*/<<endl;
//copyright
    cout<<"Send bugs to - nmr@phreaker.net"/*Copyright 2002 nmr\n"*/<<endl; //do not
touch
    cout<<"type 'listcmd' for command list\n"<<endl;
}

int lido_engine::exec(char tmp[])
{
    char param[nop][noch],buffer[buflen];
    bool through_checking=0;

    for(int z=0;z<noc;++z)
    if(strcmp(tmp,linux[z])==0)
    {
        system(dos[z]);
        cout<<endl;
        through_checking=0;
        break;
    }
    else
        through_checking=1;

    if(through_checking==1)
    {
        if(strcmp(tmp,"listcmd")==0)
        {
            listcmd();
            through_checking=0;
        }
    }
}

```

```

    }

    if(through_checking==1)
    for(z=0;z<noc;++z) //z is the
index of the command working on
    {

        if((nov[z]!=0)&&(tmp[strlen(linuxheader[z])]==' '))
        {
            strncpy(tmpheader,tmp,strlen(linuxheader[z]));
            tmpheader[strlen(linuxheader[z])]='\0';

            if(strcmp(tmpheader,linuxheader[z])==0)
            {
                if(nov[z]==1)
                {
                    for(int
y=strlen(linuxheader[z])+1,x=0;y<=(int)strlen(tmp);++y,++x)
                        param[0][x]=tmp[y];

                    if(strcmp(func[z],"chprompt")==0)
                    {
                        strcpy(prompt,param[0]);
                        cout<<endl;
                    }
                    else
                    {
                        strcpy(buffer,dos[z]);
                        buffer[strlen(dosheader[z])+1]='\0';
                        strcat(buffer,param[0]);

                        system(buffer);
                        cout<<endl;
                    }
                }
            }
            else if(nov[z]!=1)
            {
                int x=strlen(linuxheader[z])+1,y=0,w=0;

                for(;tmp[x-1]!='\0';++x,y++)
                {
                    if(tmp[x]!=' ')
                        param[w][y]=tmp[x];
                    else
                    {
                        param[w][y]='\0';

                        y=-1;
                        w++;
                    }
                }
            }
        }
    }
}

```

```

    }

    strcpy(buffer,dos[z]);
    buffer[strlen(dosheader[z])+1]='\0';

    for(int i=0;i<nov[z];i++)
    {
        strcat(buffer,param[i]);
        strcat(buffer," ");
    }

    //buffer[strlen(buffer)+1]='\0';

    system(buffer);
    cout<<endl;
}

    through_checking=0;
    break;

}
/*else
{
    cout<<"fucked";
}*/

}
else if(tmp[0]=='\0')
{
    cout<<endl;
    through_checking=0;
    break;
}

}

if((through_checking!=0)&&(enable_doscmd==1)) //enable the use of dos commands
{
    system(tmp);
    cout<<endl;
}
else if((through_checking!=0)&&(enable_doscmd==0))
{
    cout<<tmp<<". "<<tmp<<" does not exist\n"<<endl;
}
}

```

```

        return 0;
    }

    int lido_engine::cmdprompt()
    {
        char tmp[noch];
        for(;;)
        {
            cout<<lido_engine::prompt;
            cin.getline(tmp,noch);
            lido_engine::exec(tmp);
        }

        return 0;
    }

    int lido_engine::listcmd()
    {
        for(int i=0;i<cmd_counter;++i)
            //if(nov[i]!=-1)
                cout<<linux[i]<<"\t\t"<<dos[i]<<endl;
            //else
                //cout<<linuxheader[i]<<"\t\t"<<dosheader[i]<<endl;
        cout<<endl;

        return 0;
    }
}

```

SOURCE CODE (lido.cpp)

```

//LIDO Command Interpreter

#include "lido.h"

int main()
{
    lido_engine data1;

    data1.cmdprompt();

    return 0;
}

```