

WineWorks : GUI Enhancements for 'wine'

Motives and Objectives

In a recent study conducted by Google Inc., of all the operating systems that were used to access google.com, only 1% were Linux systems. Hence though Linux is fast increasing in popularity it still has a long way before it gets the level of usage it deserves. One reason for this is that there are many people who are proficient with a certain program on a different OS, and that application is not available in Linux yet. Also, with Linux-Windows multiboot systems rapidly increasing, it becomes tedious to reboot everytime the desired program belongs to the other OS. Something had to be done...

'Wine' is a package (www.winehq.com) that ships with current Linux systems that solves, to a great extent, the problems stated above. 'Wine' is a Windows(TM) emulator for Linux based systems. It allows execution of programs designed for most versions of Windows and DOS from within Linux. The wine developers have already implemented most of the Win32 API, including the latest DirectX and OpenGL support. It is truly a marvel of programming. However, currently wine is still very much a command line tool. For a novice Linux user, the setup of wine's configuration file can be daunting. Further wine does not automatically detect or mount the Windows partitions onto any directory, hence most Windows programs do not run by default. Also wine does not recognize the native Windows links (.LNK files) and thus clicking on .LNK files is useless. Windows users would also love to have their Windows Start Menu on their Linux desktop as well! Something had to be done...

Project WineWorks

This is where my project comes in. The purpose of this project is making wine a pleasure. My project is mainly in the form of an application called WineWorks which is a control center for all aspects of wine from a single point. WineWorks is primarily aimed at systems with both Windows and Linux installed on different partitions. WineWorks accomplishes the following:

=> Make the setup of wine's configuration file for a user (\$HOME/.wine/config) much easier through a user-friendly dialogs. This includes mounting the required Windows partitions automatically so that wine can find the necessary DLLs etc.

=> Add support for native Windows link (.LNK) files. Now just clicking on the link from within Konqueror or typing `./linkname.lnk` from a shell fires up the appropriate Windows executable with wine.

=> Add a Windows like "Start Menu" to the KDE Panel (kicker). This gives users easy access to all their Windows programs and users need not find the executable. This required the implementation of the .LNK file support.

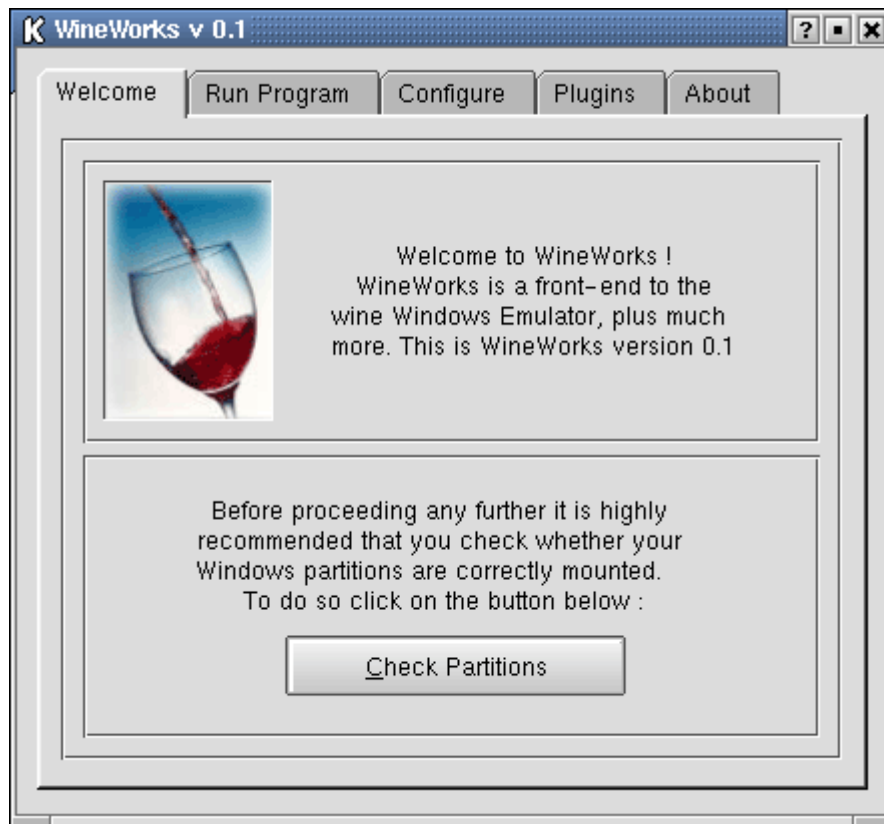
=> Attempt to autodetect settings for the configuration file.

=> Add commonly used context-sensitive menus in Windows (such as "Set as Wallpaper" for an image file).

Implementation and Results

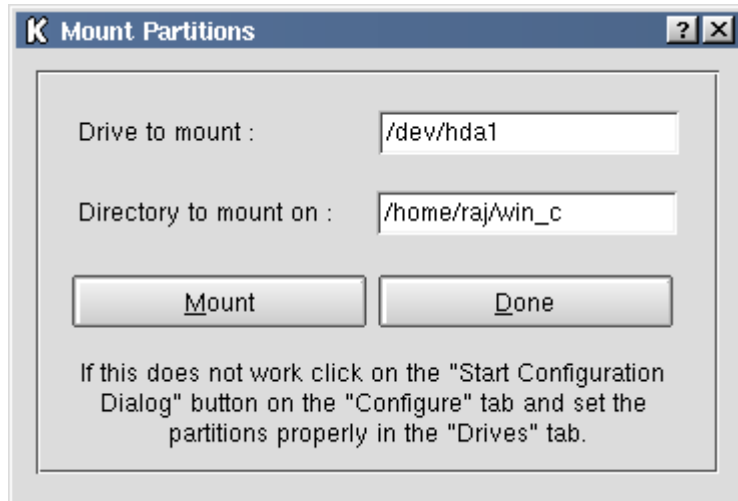
As far as programming in the Linux environment is concerned, I had absolutely no experience with GUI programming, in spite of being quite familiar with g++ or system calls or bash. Hence I knew that I had to start from scratch. I chose to work on the KDE platform using the free Qt Libraries (version 2.3.0) and the KDE Libraries (version 2.1.1) on a Red Hat Linux 7.1 system. The WineWorks project was developed using the KDevelop IDE 1.4 which has excellent integration with the Qt-Designer (used to create the user interface for my project). The excellent documentation of the Qt and KDE libraries coupled with resources from the web (see the references section) speeded up my progress.

My project results in an executable : `/usr/bin/wineworks`. Now there are two modes in which this application can be run. First, without any arguments : this starts up the normal GUI window, as shown next :



Second, with command line arguments : the argument can be a .exe file or a .lnk file. In the latter case the executable itself (for a.exe) or the executable pointed to (for a .lnk) is executed.

My entire project being dependent on the wine rpm, will not run without wine being installed. On the GUI startup, I check if the Windows partitions referred to in the wine configuration file ~/.wine/config refer to valid partitions, otherwise I mount the partitions accordingly (see below), asking for the root password if the user is not root.



On to the more challenging aspects. First I wanted to add a menu similar to the Windows "Start Menu" to kicker, the KDE Panel. Initially I toyed with the idea of writing a full fledged kicker applet by inheriting from the KPanelApplet class and adding some boilerplate factory functions and .desktop files. But I discarded the idea as soon as I discovered the elegant DCOP mechanism for Inter-Process Communications in KDE. This allows any program to issue a function call to another program through a central DCOP Server which routes these calls. Firing up the DCOP browser (kdcop), I found exactly what I was looking for : a kicker function with the following prototype...

```
void addBrowserButton(QString startDir);
```

Relieved of the need to reinvent the wheel, I looked up the dcop Interface manuals in the kdelibs help files and added the following code to the slot for my "Enable Windows Start Menu" Button (see the next image)

```
extern DCOPClient* thisClient;

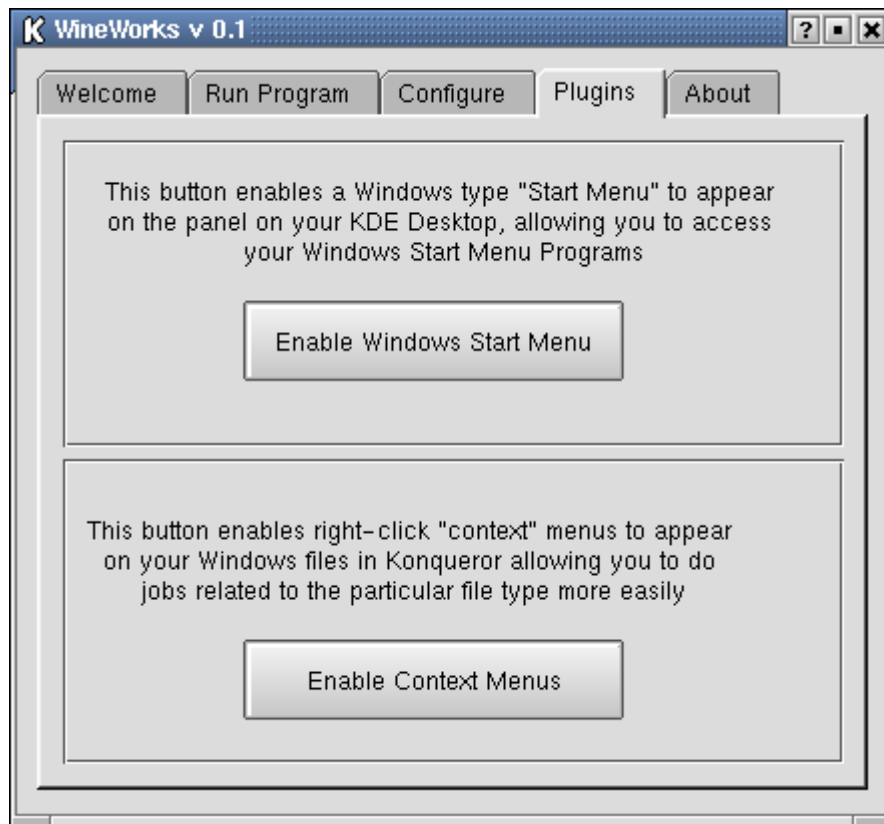
/* This variable was initialised in the main subroutine as follows :
KApplication a;
//more code...
thisClient = a.dcopClient();
*/

//more code

QString s = windowsFolder + "/windows/Start Menu";

//Send DCOP Service request via DCOP Server to 'kicker', the KDE Panel...

if(thisClient==NULL)
    qDebug("OOPS : thisClient is NULL!!");
else if(!thisClient->send("kicker", "Panel", "addBrowserButton(QString)", s))
    QMessageBox::critical(this, "WineWorks", "DCOP handling error!");
```



That solved the problem. This presented two more problems. First, in my Windows Start Menu on the KDE Panel, most entries were .LNK files, clicking on which produced nothing. Second, if I opened a terminal and typed, say `./Calculator.lnk` (found in `%WINDIR%\Start Menu\Programs\Accessories`), bash refused to run it (quite obviously). I thus wrote a C function called `processLink()` that extracts the exe referred to by a .LNK file (which works like the UNIX 'strings' utility). WineWorks now can obviously run the executable. Now, to solve the first problem, I wrote a .desktop file to install into the default KDE mime directory (usually `$KDEDIR/share/mimelnk`) to add a mime type for the .LNK files, thus making them recognizable by Konqueror.

The solution to the second was not so trivial. To make any shell recognize .lnk files, I needed to make the kernel recognize it and pass it on to wineworks. This could be done in 2 ways :

1. Develop a kernel module and insert (insmod) it.
2. Use the `/proc` filesystem to my advantage.

I decided to use the second solution as it was relatively easier than the first. I read `/usr/share/doc/kernel-doc-2.4.2/filesystems/proc.txt` and found that to register a new binary format the `/proc/sys/fs/binfmt_misc` directory provided the kernel support. To register a format I needed to simply issue the following command :

```
echo :name:type:offset:magic:mask:interpreter:>/proc/sys/fs/binfmt_misc/register
```

In my case it would be

```
echo :link:E::lnk::/usr/bin/wineworks: > /proc/sys/fs/binfmt_misc/register
```

and to make it survive reboots, I needed to add a script containing the previous command to `/etc/rc.d/rc3.d`.

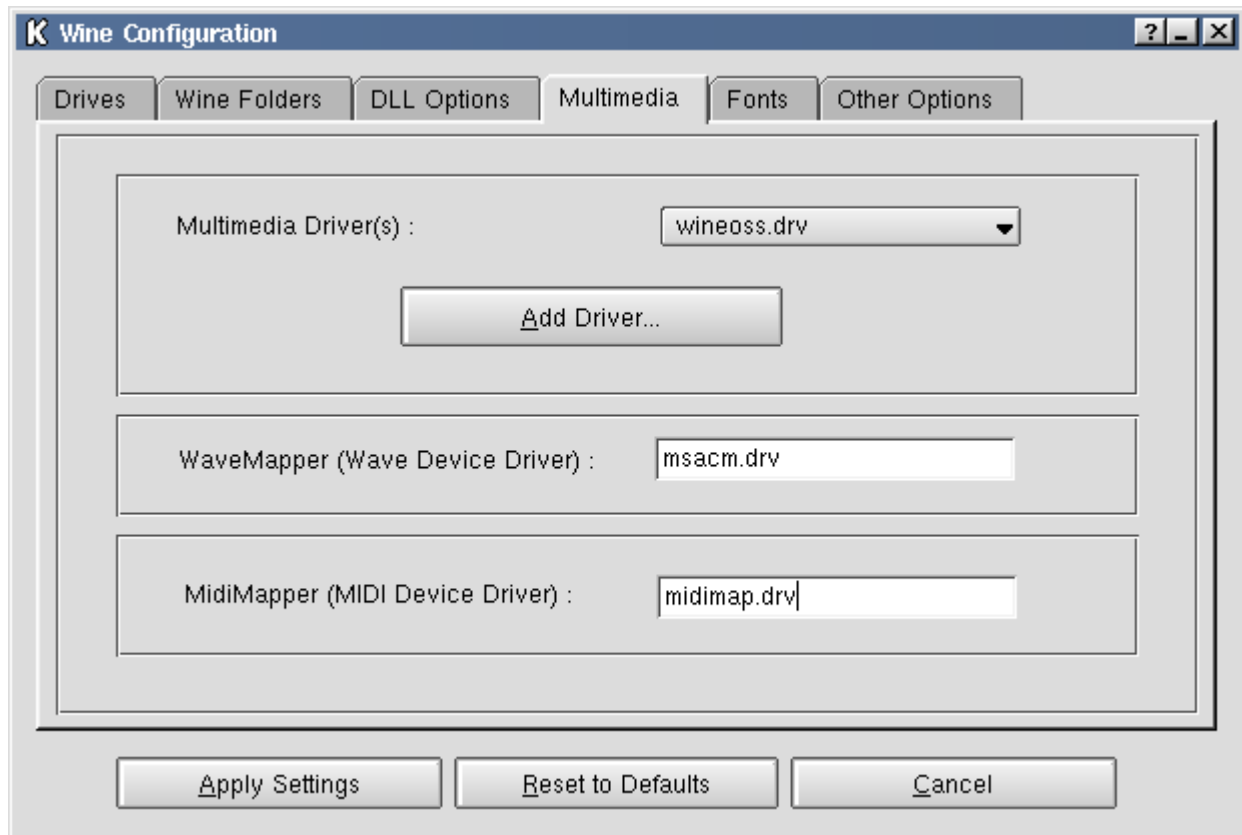
To add context sensitive menus, known as servicemenus in KDE jargon, for certain filetypes, I had to install a `.desktop` file in the appropriate location. For instance, for the "Set As Wallpaper" for jpeg files, I would write a file as...

```
[Desktop Entry]
ServiceTypes=image/jpeg
Actions=SetWallpaper

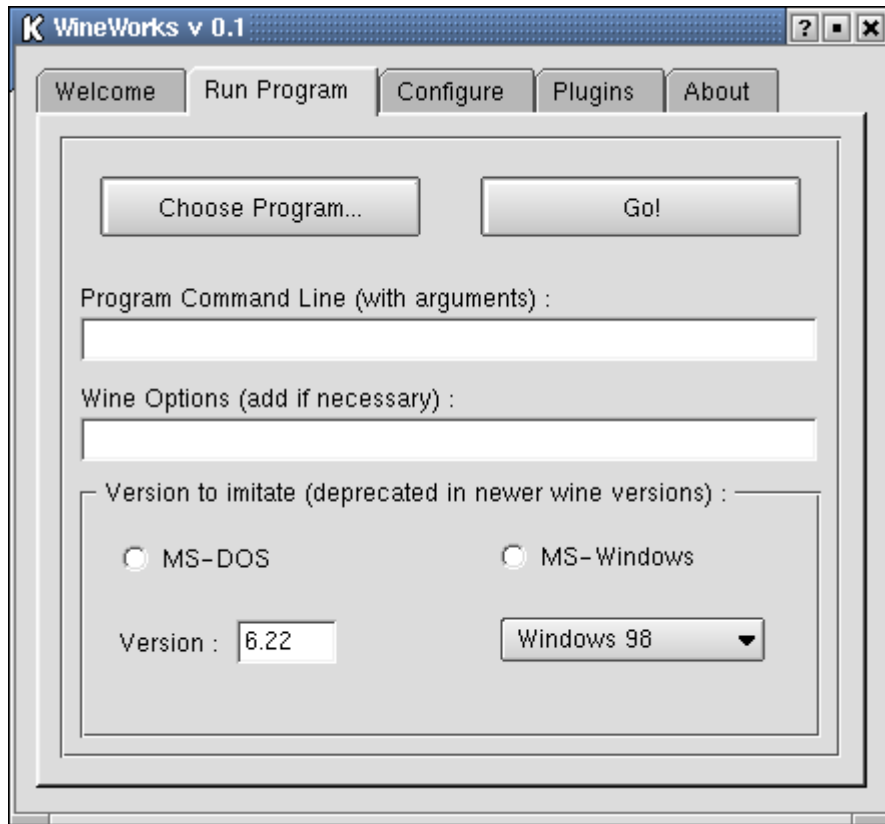
[Desktop Action SetWallpaper]
Name=Set As Wallpaper
Icon=background
Exec=dcop kdesktop KBackgroundIface setWallpaper %U 5
```

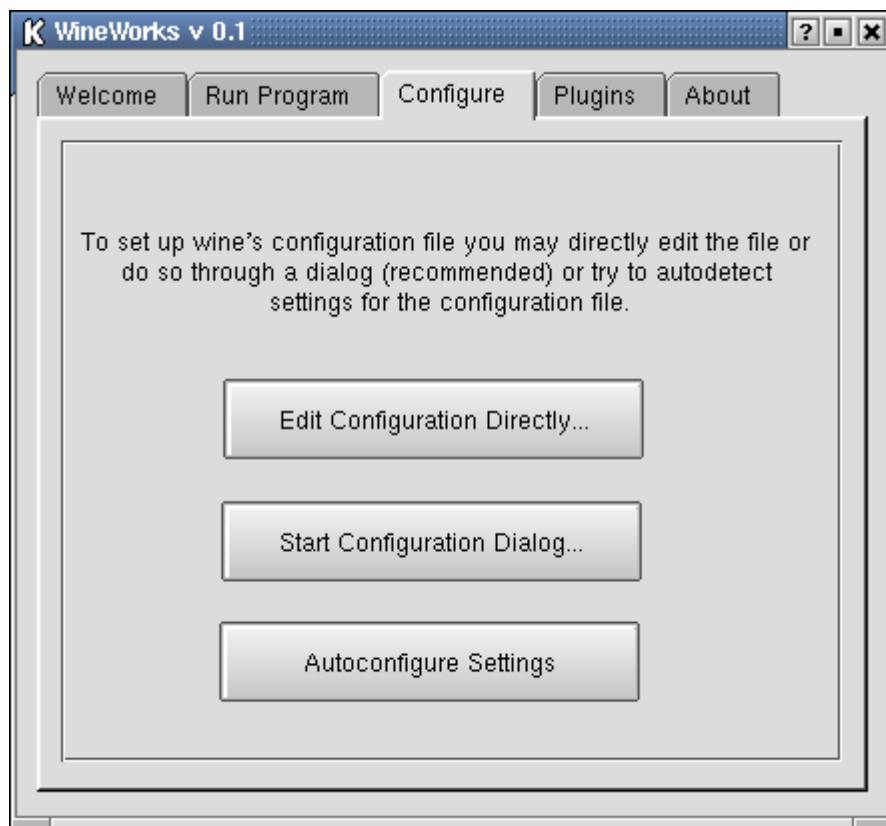
...and install it to `$KDEDIR/share/apps/konqueror/servicemenus/SetWallpaper.desktop`. In this case it is DCOP to the rescue again as the `setWallpaper` function is already available. If a function is not available, we must derive from the class `DCOPObject` to create a DCOP interface.

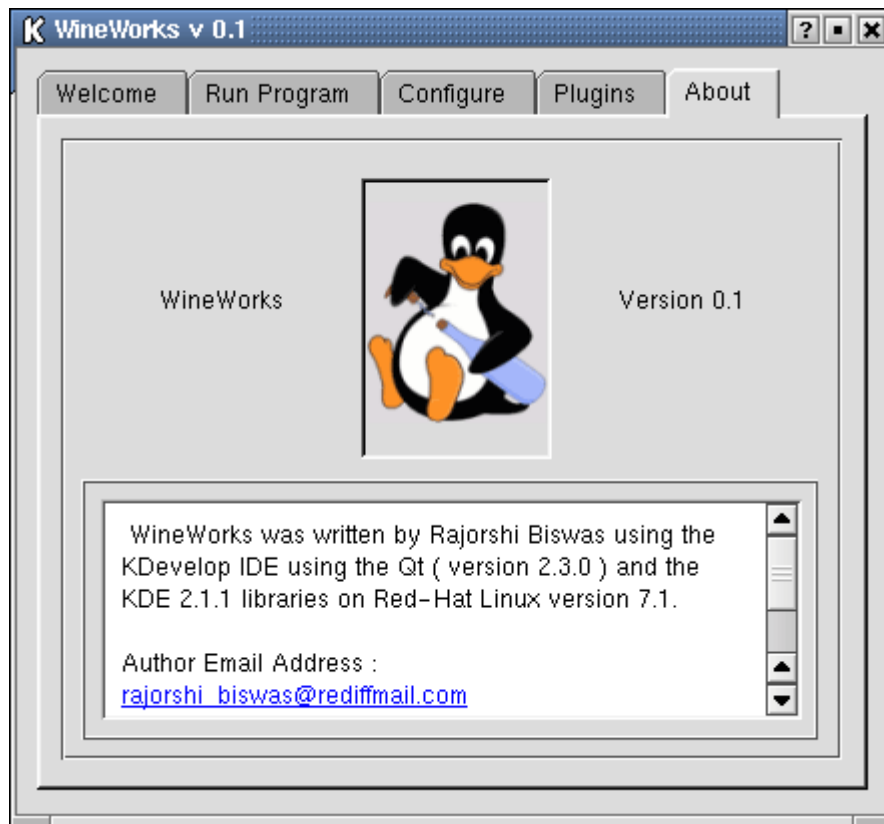
The rest of the project includes a separate dialog for setting up the configuration file easily, a file dialog to enable the user choose the `.exe` or `.lnk` to run, with optional arguments etc. (see the next sample image).



The other tabs of the main window are shown next.







A few important parts still need to be implemented :

1. The Autoconfiguration of the partitions needs to be done. (Probably I could use the `popen()` system call to capture the output of `/sbin/fdisk -l` to get the partition information).
2. Better error handling is desired, as wine crashes frequently.
3. Enhancement of the context-menus.
4. Add Windows partitions to `/etc/fstab` to mount them automatically after a reboot.

Conclusion

I think that WineWorks will make the wine program much more popular, since now there is no need to manually do tedious jobs such as mounting the Windows partitions or editing the configuration file. Even the ubiquitous `.LNK` files behave as they should.

The results of this project, currently amounting to about 5000-6000 LOC, are truly satisfying, but it can be improved a lot more. I am working on its improvement right now, and I hope that it will be a really useful product for the open source community and especially for

those new to Linux. I shall release my open source project to the public soon at <http://sourceforge.net> . I shall consider myself successful if can make people new to Linux realize the joy of Linux.

References :

1. ' wine' manuals from (/usr/share/doc/wine) www.winehq.com and the manpages.

2. ' wine' documentation from the following sources :

www.westfalen.de/witch/wine-HOWTO.txt

www.winehq.com/dev.shtml

www.la-sorciere.de/wine/index.html

3. Qt-Designer Manual and the Qt-Class References.

4. KDevelop Manuals and the KDE Libraries Documentation.

5. Various tutorials from <http://developer.kde.org>

6. Various tutorials from <http://dot.kde.org>

7. The comp.emulators.ms-windows.wine newsgroup.

8. "Creating KParts Components" 1 and 2 and other tutorials from IBM developerWorks : Linux : Education - Tutorials.

9. /proc filesystem documentation :
/usr/share/doc/kernel-doc-2.4.2/filesystems/proc.txt

Note :

The reader is requested to contact me without hesitation for clarification of queries and/or obtaining the sources/rpm of my application at the email address given below or also at rajorshi@vsnl.net .

Rajorshi Biswas
rajorshi_biswas@rediffmail.com