Different Python Versions

S. de Graaf

Circuits and Systems Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
Delft University of Technology
The Netherlands

Report EWI-ENS 09-01
Jan 12, 2009

Last revision: Jan 14, 2009.

1. INTRODUCTION

This year started for me with a new PC on my desk. The new PC is a DELL Precision T3400, which contains an Intel Core2 Duo CPU (E8400 @ 3.0 GHz). It comes also with the new Linux SUSE 11.1 distribution.

Thus, it was needed to setup my space development environment again. I will now also do my development work more on the local disk, because this is faster and more independed of the local network and the remote machines.

I think that i can say, that my old environment (hardware, software and Linux version) were good compatible with my new environment. Thus, an old space distribution can work with-out any problem. Also the existing 3rdparty linux-2.6-i686 software and GNU compiler (3.4.6) can still be used with-out any problem.

There is however a problem to use the local GNU compiler (4.3.2). But, for other things, it is maybe possible to use more local tools and libraries. However, not all libraries are delivered with a static version.

Thus, it must also be possible to use the newer local python version. Thus, i have given it a try. Below you can find the results of the Dutch jury.

For example, the 3rdparty version of python gives the following result:

```
% python
Python 2.4.3 (#1, May 16 2007, 14:49:48)
[GCC 4.1.2 20061115 (prerelease) (SUSE Linux)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Traceback (most recent call last):
  File "/etc/pythonstart", line 7, in ?
    import readline
ImportError: No module named readline
>>>
```

The local SUSE 11.1 version of python gives the following result:

```
% python
Python 2.6 (r26:66714, Dec  3 2008, 06:05:48)
[GCC 4.3.2 [gcc-4_3-branch revision 141291]] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

To use the local version of python read the following section. And find out what is different and what has to be changed to the sources.

2. NOTES

When you log-on to your system or open a new terminal window, your login shell (tcsh) reads your ".cshrc" setup file. It sets-up your (the user) environment (the path, aliases, etc.). For the space developer, it can also execute "cconf". Normally, it executes "cconf" when going to "cacdmode".

Because "cconf" is a python script, you must first make the choice which python you want to use. In case you want to use the 3rdparty version, you must first set the 3rdparty path in your ".cshrc" setup file.

I have changed file "~/.cacd/setup/setup2.tcsh" to modify the PATH environment variable. Note that the 3rdparty "bin/fallback" directory don't need to be added when a local version of vim is used.

The "setup2.tcsh" file executes also the "setup.tcsh" file. I use the "setup.tcsh" file from "~/.cacd/setup". This file modifies the PATH environment variable and adds the 3rdparty path. Note that the 3rdparty path is not realy correctly set, unless CACD_ARCH is correctly symbolic linked to CACD_ARCH-HOST. This important fix is done by my local (private) 3rdparty setup. Note that "cconf" removes incorrect added path directories.

When the 3rdparty "bin" path is not set, you are using the local version of python and gets the following messages:

```
% cacdmode
/users/simon/CACD/src/cmake/cconf:10:
      DeprecationWarning: the md5 module is deprecated;
      use hashlib instead
  import md5
/home/simon/CACD/src/cmake/std.py:12:
      DeprecationWarning: The popen2 module is deprecated.
      Use the subprocess module.
  import popen2
Changed PATH variable.
  ...
```

You can remove "import md5" from "cconf", but then you get the same message from file "std.py" (where md5 is used).

The following changes must be applied to the old python code to let it work with the new (local) python version:

```
% cd /home/simon/CACD/src/cmake
```

```
% diff cconf cconf-new
10c10
< import md5
---
> import hashlib
```

```
% diff std.py std.py-new
8c8
< import md5
---
> import hashlib
12c12
< import popen2
---
> import subprocess
123c123
<     m = md5.new()
---
>     m = hashlib.md5()
130c130
<     m = md5.new()
---
>     m = hashlib.md5()
349,351c349,350
<     child = popen2.Popen3(command, 1) # capture stdout and stderr from command
<     child.tochild.close()             # don't need to talk to child
<     outfile = child.fromchild
---
>     child = subprocess.Popen(command, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
>     outfile = child.stdout
353c352
<     errfile = child.childerr
---
>     errfile = child.stderr
```

I don't know of the last two code changes are correct. It is a little bit too complicated what the "popen2.Popen3" call realy does. And how the streams must be redirected. But above code seems to be working.

```
% diff c3p_module.py c3p_module.py-new
12c12
< import md5
---
> import hashlib
15c15
< import popen2
---
> import subprocess
```

```
% diff cmake_module.py cmake_module.py-new
12c12
< import md5
---
> import hashlib
15c15
< import popen2
---
> import subprocess
```

```
% diff ctools.py ctools.py-new
10c10
< import md5
---
> import hashlib
```

```
% diff utils.py utils.py-new
7c7
< import md5
---
> import hashlib
11c11
< import popen2
---
> import subprocess
```

The following change has nothing to do with another python version. But seems to be needed for compiling dali etc. with new X11 libraries.

```
% diff system.py-bak system.py
134,135d133
<         if "x86_64" in context.sheet["platform"]:
<             node.use_libs(["-lxcb", "-lxcb-xlib"])
137a136
>         node.use_libs(["-lxcb", "-lxcb-xlib"])
```

The "xcb" and "xcb-xlib" libraries must be added (like i had done before for the 64 bits platform). I changed also the order of the lines. It is now added after library "Xau" and library "Xdmcp".

Thus, this are the results of the Dutch jury.