Fedora compatibility settings

Fedora moves fast, and the latest versions of things are not always compatible with previous versions. It is difficult enough to keep your own programs up to date, but sometimes you need to use a piece of software which doesn't work any more due to the rapid changes in compilers and system libraries. Here are some tips and workarounds.

Gcc 3.4

The default compiler is now GCC 5.1 (4.9 in Fedora 21). However, Gcc 3.4 is still available (on request, NOT by default) through the commands gcc34 and g++34. However, those packages will not be maintained forever, so it may be time to figure out why you or your code thinks these older versions are needed. Besides, it will not be possible to link code compiled with gcc34 to libraries compiled with gcc 4.x, so if your program requires external libraries, you should use the current compiler version.

If you want the old compiler to use g77 (Fortran 77) instead of gfortran (Fortran 90/95), you may be better off using gfortran, with some compatibility options, if required. Mostly, use `-std=legacy` to suppress warnings about obsolete code. In special cases, other options may be required, but the internet is full of descriptions on how to proceed.

Various other versions of GCC are also available using environment modules. See `module avail` for a listing.

Glibc problem detection

The current C library (Glibc >= 2.5) is quite good in detecting illegal operations from a program, such as buffer overflows, freeing a bit of allocated memory twice, corrupted pointer lists etc. It will then abort the program with a message starting with **glibc detected:** followed by an explanation of the problem. Usually, this is a good thing. The alternative would be to let the program go on with whatever it was attempting to do, in an uncontrolled way, perhaps resulting in a system crash, data corruption or security hole. However, if you cannot fix the problem in the program, you may want to switch the problem detection off. Glibc's malloc() function call can be controlled using the environment variable MALLOC_CHECK_ . A value of 0 means: no check, so you get the old (and unpredictable) behaviour back. A value of 1 just prints warnings but doesn't prevent the program from running, and 2 gets you back to the current default behaviour to print the error message and abort.

Let's rephrase this to make the point clear: the **glibc detected:** message is not the problem, and MALLOC_CHECK_=0 is not the solution, it is just a workaround to bypass a check in the C library. The real solution is to debug your code and to make sure it doesn't cause the error that glibc detected!

`See man malloc for more details.`

Changes in system implementation

Some bits of behaviour of the Linux system have changed over the last couple of years. Where possible, the program setarch can be used to select a different “architecture” for the execution of the program. Syntax:

```
setarch i686 [setarch-option] [program with program options]
```

The option you probably need most is -L to switch to another method of virtual memory allocation. E.g. iraf seems to need this (and it is thus included in our iraf startup scripts, but you may have to make such a setting yourself when you install iraf on your laptop, or if you call iraf tasks from outside of iraf). See man setarch for more details.