Migration of ECMWF's Operational Meteorological System to UNICOS on the Cray Y-MP

J. K. Gibson and Otto Pesonen

European Centre for Medium-Range Weather Forecasts

Shinfield Park, Reading, U.K.

The ECMWF Meteorological Operational System (EMOS) contains approximately 150 separate Cray jobs, some of which are run up to 30 times each day. With the decision to acquire the Cray Y-MP, it became necessary to migrate this system to run under the UNICOS operating system.

First, it was considered important to define the conventions to be followed. They included proposals for:-

* user names and privileges
* file names and permissions
* operational libraries
* temporary files
* script conventions
* script and function utilisation

These were written into an internal document which has subsequently been updated many times, and which will eventually be made available as a Meteorological Bulletin. Possibly the most difficult part of this exercise was the definition of a typical operational script. The EMOS system relies heavily on being able to intercept all errors, and to report an ABORT state back to the SMS (Supervisor-Monitor-Scheduler) management system, and hence to the attention of operators and analysts. UNIX systems, being designed for interactive use, tend to report errors as messages, then to continue with the next command; thus, the trapping of errors in a batch script requires careful planning.

Next, the individual scripts for each job were written, the required codes migrated, and each job step carefully tested. Since much of the source code had been adapted over the years to isolate as far as possible operating system dependent features, and since UNICOS supports the same FORTRAN compiler and segment loader previously used on the Cray X-MP, many codes required little more than re-compilation. Some specialised features, such as the fields data base and some aspects of the MARS system, required code to be re-written and re-implemented; in particular, the MARSINT interpolation package was re-written to enable future support for higher resolution models.

Cray undertook the task of modifying the SMS system they had written, in co-operation with ECMWF, for the VAX/VMS system to operate under UNICOS. Once this was delivered, the necessary jobs to run data assimilation cycles and a 10 day forecast were tested, and when working assembled as a test suite under the new SMS. These were run on a daily basis from early October. As further jobs were migrated, these were added, resulting in an almost complete operational suite by late October.

The EMOS system relies heavily on MARS and ECFILE, and it was essential that these components be available as early as possible. Thus, from an early stage a MARS system was made available to internal users, so that it could be thoroughly debugged by the time it would be required for operations.

By the 7th November sufficient confidence had been gained to move the delivery of the operational dissemination products to those generated on the Y-MP. Although there were some initial problems, every effort has been made to achieve as smooth a transition as possible, and to make the changes as transparent as possible for Member States users. Additionally, a series of planned "disasters"

were staged, while the X-MP was still available, including the complete destruction of the Y-MP file base, to ensure that procedures for recovery were checked out, and amended where necessary. These provided much useful information which was subsequently applied to ensure the future robustness and reliability of the service.

Annex. Recommendations for ECMWF Meteorological Operational System UNICOS Scripts

The following is an extract from the conventions drawn up for the migration of the ECMWF Meteorological Operational System (EMOS) to UNICOS. It is reproduced in the belief that it may be of general interest.

A standard login script is used for each EMOS job; it contains 3 sections:

1) SMS initialisation sequence
2) definition of environmental variables
3) definition of standard search paths, libraries, etc.

The SMS initialisation sequence, additionally, establishes a number of variants with respect to the date/time, stored in environment variables EMOS_TIME_DIFF_etc., where etc. includes M3, M6, M9, M12, M18, M24, P6, and P24. These contain yyyymmddtttt values corresponding to the base time minus or plus the increment indicated (eg, EMOS_TIME_DIFF_M24 contains the yyyymmddhh for 24 hours before the base time).

In addition, the environment variables include:

SMS_NODE	SMS_MONTH	EMOS_MONTH
SMS_SUITE	SMS_DAY	EMOS_DAY
SMS_FAM	EMOS_SUBS	EMOS_TIME_DIGITS
SMS_TASK	EMOS_BASE	EMOS_TIME_STEP_H
SMS_YEAR	EMOS_YEAR	EMOS_TIME_STEP_M

Those values not directly presented via SMS micros are assembled from such information within the login script; the algorithms used differentiate between old style family names and new style family names, and assemble the correct

information accordingly. Note that CAPITAL letter are used for global environment variables, lower case letters for variables local to a particular script.

Since the basic UNIX system can be unfriendly and cryptic in appearance, the usefulness of the system is based upon the local operating environment and the accepted strategies. When many people access the same files and use the same programs and scripts, it becomes very important that the users know how file names and environment variables are defined, and how to get help from the computer.

When new tools are made, they should be documented immediately and made available for other users. In order to automate the documenting system, UNIX provides some tools, such as "man", but additional tools need to be build locally. Programs (FORTRAN, C or SHELL) can be made self-documented; that is, the documentation is written as comments, then extracted and stored for the man command. This is a logical extension of the DOCTOR type conventions already used throughout ECMWF for FORTRAN source code (Gibson, 1982). Since UNICOS man command does not provide full search facilities, there should be separate man pages to guide the users. The same manual pages should be available for workstations using either the file server or automatic distribution.

Every effort should be made to provide neat, well documented scripts. Each script should have at least the following information at the beginning of the file:

* script name (possibly full path name)
* function
* interface - how called
  - environment variables
  - exit value(s)
* externals - other scrips, functions, programmes called
* method
* external documentation
* date of creation and last modification (modifier ID)

The body of each script shall be divided into sections, and, if necessary, sub-sections, following the header described above. The only additional items of documentation that should be necessary within the script are the section and sub-section headings; however, documentation should be included to describe any unusual or obscure statements (these should be few and far between, as they are to be avoided wherever possible).

Scripts shall be given names which are meaningful, rather than cryptic. It does not matter if the names are long - they are intended for batch, not interactive use. The name should indicate clearly what the script is doing. This makes other scripts more readable. The time spent searching for information concerning what some cryptically named script actually does is often far greater than the time saved by writing short names.

UNICOS 5 limits file names to 14 letters. Use them all! Don't add .sh (for Bourne shell scripts) as an appendage - it wastes 3 letters and is not necessary. For example:-

exss - MEANINGLESS AND BAD

extractsubstr or extr_sub_str - makes more sense!

There are two types of variables in scripts, local and global (or external). By convention the local variables should be in lower case and EXTERNAL variables in UPPER CASE. This makes it easy to read the script, since it is obvious which is which. Variable names are not limited in length; so use names that really mean something, for example:

string_length or

str_len

NOT

sl.

Loop counters can be short, "i" is often used in "for" loops.

Attention should be given to the effectiveness of the use of functions defined within an include file. A set of useful functions will be maintained in

/ec/emos_sms/.functions

Bourne shell provides an easy way of creating a simple task to be executed INSIDE the current process by the function definition mechanism. This is similar to the C-shell alias mechanism. It is not obvious when to use a script, a function, eval or a program. The following guidelines are derived from Otto Pesonen's experience of UNIX programming. In most cases it is advisable to begin by writing a script, then if necessary later move to a programme.

A function should be used:-

* when the task is relatively small
* when the task should alter the calling process (environment

variables)

* when the task is called many times

NOTES:

1) Remember to use return instead of exit to terminate the function.

2) Test functions properly!

A script should be used:-

* when the task is complex
* when the task is new, or rarely used from other scripts
* when the task only needs to return exit status, and does not alter the caller's environment

"eval" should be used:-

* when the script (or program) needs to alter the caller's environment, such as computing an environment variable from given parameters where the use of a function is not acceptable;
* this is the only way to pass arguments to another script and still alter the caller's environment.

NOTE:

When "eval" is used, the referenced script needs to output commands for the shell to execute. This often requires scripts which are expected to be used as targets of eval to contain two levels of logic; resulting scripts can look vary clumsy, and should be avoided where possible.

The . (dot) command should be used:-

* for startup files
* to include a file

NOTE: arguments cannot be used!

A C programme should be used instead of a script:-

* when the task needs character handling or something that is very difficult using script(s) (eg mathematics)
* when the task is complex
* when the shell script takes too much time to run or gets too complicated

NOTE:

The overhead of starting a small script is more than that of starting a C programme.

Functions should be subject to similar naming conventions to scripts. Externally defined names should be in UPPER CASE and local names (defined in the current script) in lower case. Names should be readable and they may be as long as required - not limited to 14 characters!

When the script terminates itself due to a missing file etc, it should abort giving the reason why, and also trace information concerning how the script was called. A function is available to handle this. To facilitate the clarity of batch job output, and to assist in locating run-time errors, it is necessary for a script to output information of the progress it has made. The use of set -v and set +v for this purpose are rather crude. A more elegant method is to use the "echo" command, or by echoing comments from the current shell; alternatively a simple function (info) is available to output additionally a trace of all the names of scripts called. Such trace information can enhance the value of such output, assisting the identification and location of problems. It is recommended that the "info" function be used on entry to each script, and that an "abort" function be specified to be called to present traceback and additional

information in abnormal situations. It is also recommended that a script header sequence, and a section header for each section of the script about to be executed, be passed to the log.

The functions should be gathered into a starting (profile) file to be included into scripts, invoking them by the Bourne shell command . (dot).

```
. /some_path/start_funcs
```

Reference:

Gibson, J. K., 1982: The DOCTOR System - a DOCumenTary ORientated programming system, ECMWF Technical Memorandum No. 52