ANDFizing OSF/1 R1.2 commands

Christian Fabre

Open Software Foundation
Research Institute

June 1993

This paper presents a study of ANDFizing OSF/1 R1.2 commands. It describes the various portability issues encountered while ANDFizing the basic `/bin/ls` command.

1. Introduction

This paper presents an investigation of the issues involved in porting OSF/1 R1.2 commands to the ANDF[1] technology.

OSF/1 Release 1.2 (OSF/1 for short) is the latest version of the operating system developed by OSF. Release 1.2 is compliant with IEEE Std 1003.2-1992 [POSIX-SU] and based on the Mach technology. Additionally, OSF/1 Release 1.2 has been successfully tested against the X/Open verification suites (VSX4), verifying source compliance with the base profile of the X/Open Portability Guide Issue 4 [XPG/4-IF][2].

1. ANDF is an architecture- and language- neutral distribution format being developed by OSF and other collaborators around the world. It is based on the TDF technology provided by the Defence Research Agency (DRA) of the UK Ministry of Defense.
2. Note that the API defined by [ANSI-C] is included in [POSIX-IF] itself included in [XPG/4-IF].

Our investigation involved porting commands to the ANDF compiler, delivering them as a set of target-independent '`.j`' files[3], and installing them on two platforms: i386 and mips, both running OSF/1. So far, the investigation has focused on the `/bin/ls` command.

2. *Token libraries to achieve architecture neutrality*

The key feature of ANDF for achieving architecture neutrality is the *token* mechanism. A token is roughly a typed macro: its target-independent definition is given in the C producer headers files, and each platform provides its actual declaration through a target-dependent token library to be merged with the '`.j`' ANDF file before translation.

Porting with ANDF is usually done in 4 phases:

- Compile and run the application on the current platform with the native compiler, hence using the native standard headers and standard libraries. This is just a sanity check.
- Compile and run the application on the current platform with the ANDF compilation chain using the native header files and the native libraries. This phase is to show up any discrepancies between the native C compiler and the ANDF compilation chain.
- Compile and run the application on the current platform with the ANDF compilation chain using ANDF target-independent include files and the target-dependant token library. Currently DRA provides five sets of API headers: ANSI-C, GCC, SYS-V, POSIX, and XPG/3.
- Check that the generated '`.j`' files can be installed and run on other platforms.

3. This document uses the following terminology:
A *producer* is all of the software that is used to produce the ANDF form of an application. The primary component of a producer is a *compiler*, which does the actual translation of source code into ANDF. An *installer* is all of the software used to install an application on a target. The main components of an installer are the *TDF binder* which resolves the target-dependencies by merging the target independent ANDF files with the *token libraries* of the platform, and the *translator* which does the actual translation of target-dependent ANDF into machine-code. The target-independent ANDF files (output of producers) are suffixed by '`.j`'. The target-dependant ANDF files (input of translator) are suffixed by '`.t`'.

For more details about porting C software using the ANDF technology, see [SM/OSF-I-93].

3. Standard APIs and OSF/1

As stated above, the OSF/1 operating system supports various *de jure* APIs: [ANSI-C], [POSIX-IF], [XPG/3-IF] & [XPG/4-IF]. The official API supported by OSF is described in [AES][4]. Besides these *de-jure* APIs, OSF/1 supports partially APIs coming from other branches of the Unix family such as:

- Mach, from Carnegie Mellon University, as OSF/1 is based on the Mach technology;
- BSD, from the University of Berkeley. *e.g.* the sockets;
- System-V, from USL. *e.g.* shared memory facilities and semaphores.

The actual interface *accessible* (as opposed to *supported*) through OSF's `/usr/include` headers and the C libraries is much larger than [AES]. Moreover, as the OSF/1 commands have to access internal structures of the OS, they are more likely to use *unsupported*[5] APIs. *e.g.*: `/bin/ps` has to access the process table of OSF/1's kernel to obtain the status of a process.

4. ANDFizing the `/bin/ls` *command*

The `ls` command

`ls` is a basic command of all Unix operating systems. The baseline behavior as described in detail in [POSIX-SU] and [XPG/4-CU] is supported by OSF together with some additional functionality. `ls` prints out file names and directory contents in various formats, *e.g.*:

4. [AES] is OSF's official API. It incorporates features coming either from BSD or System-V. Its definition is several years old, an updated version is in progress at OSF.
5. The word *unsupported* just means that OSF is not committed to the stability of these interfaces.

- Line-by-line, or with file names aligned in columns (option `-C`);
- The `-F` option appends a character at the end at a file name to give its type: '`/`' for directories, '`*`' for executables '`@`' for symbolic links, and '`=`' for sockets, as in (where `bin` is a symbolic link to `/usr/bin`):

```
% ls -F
bin@
```

- Just the file name, or both file name and additional information about the file (option `-l`) like the size, the number of links… For special files (*e.g.* in the `/dev` directory) `ls -l` prints out their *major* and *minor* numbers of the device in place of the size, for example:

```
% ls -l /dev/tty
lcrw-rw-rw- 1 root 19      2, 0 May  7 15:23 /dev/tty
```

- While listing directories with some options turned on (*e.g.* `-l`), the `ls` command prints out on the first line of output the actual size occupied by the directory elements, in 512 byte units, for example:

```
% ls -l
total 66
-rw-r--r-- 1 fabre wheel  32081 May  7 16:06 ls.c
-rw-r--r-- 1 fabre wheel    411 Feb 25 13:56 ls_msg.h
```

`ls` and portability

The purpose of this investigation is not to have a portable version of `/bin/ls`, but rather to detect and highlight the dependencies of OSF's version upon the various levels of API provided by OSF/1, in order to have a portable version of OSF's flavor of `/bin/ls`. Indeed, a side-effect of this investigation is that it seems impossible to write a version of `/bin/ls` providing all the functionality that one can expect from this command by strictly adhering to *de jure* APIs.

The `ls` command is built from one C source file `ls.c`. This file acquires object[6] definitions/declarations from various standard header include files. Some of these objects have got a precise (*de jure*) definition, either through OSF's specified interface [AES] or through other specified interfaces such as [XPG/4-IF] or [SVID-3]. Other objects have no *de-jure* specification, but are

6. By object we mean types, function names, constants…

present for historical reasons on almost all Unix flavors, and therefore are also present in OSF/1. The `ioctl()` function is such an historical facility.

We describe here the various issues raised while porting `ls`.

— *The* `errno` *variable*

`ls.c` includes `<sys/errno.h>` to provide the definition of the `errno` variable, and the `EACCES` constant. No standards body specifies this file. According to [XPG/4-IF], these entities are defined in the standard header file `<errno.h>`.

The `<sys/errno.h>` header was replaced by `<errno.h>`.

— *Size of the screen in columns*

To line-up names in columns, `ls` either uses the value present in the environment variable `COLUMNS`, or obtains the horizontal size, in characters, of the window it is running in. The latter is done by a call to `ioctl()` with the constants `STDOUT_FILENO` and `TIOCGWINSZ` and an empty `winsize` structure as parameters. After the call, the number of columns of the enclosing window is in the field `ws_col` of the `winsize` structure.

Here we come down to critical issues with regards to `ls` portability. Obtaining the size of a window seems to be highly OS-dependent: OSF/1 uses a structure named `winsize` and the command passed to `ioctl()` is `TIOCGWINSZ`[7], while SunOS defines a structure named `ttysize`, and the command name is `TIOGSIZE`.

Fortunately enough, these two interfaces have the same profile:

- For the structure: Four `unsigned short` handling the horizontal and vertical size of the screen, in numbers of characters and in numbers of pixels, in the same order;
- An integer constant to name the command.

7. This way of obtaining the size of a screen originally comes from BSD.

ANDF provides a way of unifying these two interfaces: As these two interfaces exactly map one onto each other, only one set of token names is required, say the couple `winsize/TIOCGWINSZ`. The ANDF target-independent headers could turn source code references to the `ttysize/TIOGSIZE` interface into ANDF references to tokens as defined by `winsize/TIOCGWINSZ`. On the target-platform, the token library could use either `ttysize/TIOGSIZE` or `winsize/TIOCGWINSZ` to implement the `winsize/TIOCGWINSZ` related tokens. Such a scenario provides the maximum flexibility:

- Source code could use both interfaces;
- Targets are only required to provide one of them.

For the time being, despite its broad availability on actual platforms, the `ioctl()` function is left unspecified by all standardization bodies, as it is too platform dependent.

This issue of obtaining the size of the window could be fixed in many ways:

1. Put the relevant code for each platform between `#ifdefs __MY_PLATFORM` and `#endif`.
   The generated ANDF will be portable only among platforms whose specific C code has been enclosed in `#ifdefs` at ANDF generation time. Furthermore, should the semantics of the `ioctl()`[8] function change in a future version of OSF/1, the ANDF code would no longer be valid, preventing upwards compatibility.
2. Defining a new C macro named `get_window_number_of_columns(int stream)` which will return the size of the window as an integer. The definition of this macro being in a new header file named, say `<osf-commands.h>`. The native version of this header would generate the proper macro expansion, and the ANDF headers could tokenise it. In this case, it is up to the token library to provide the definition of this token. That is, if a new platform is introduced, or if a given interface to obtain the window size changes, a corresponding token will have to be provided in the

8. `ioctl()` is an unsupported function provided by OSF/1.

token library for this platform. The user will then be able to use the *same* existing ANDF. When all OSF/1 commands have gone through this ANDF porting process, we will end up with the file `<osf-commands.h>` summarizing all *unspecified* interfaces used by OSF/1 commands, and therefore covering the actual API used by OSF/1 commands

3. Definition of a new set of ANDF-C header files extending the current set of APIs delivered by DRA, namely, [ANSI-C], GCC, [POSIX-IF], [SVID-3], [XPG/3-IF] and soon [XPG/4] and [AES][9]. This new environment, which we will call `osc12`, would describe unspecified facilities actually provided by OSF/1. Note that such a process will allow upwards compatibility as long as this API is actually supported by the target platform. When all OSF/1 commands have gone through this ANDF porting process, we will end-up with the environment named `osc12` summarizing all *unspecified* interfaces used by OSF/1 commands, and therefore having the actual API used by OSF/1 commands.

It is worth noting that (1) is a common approach but provides the least compatibility: A new platform might not be able to use existing ANDF as its specific code would not be included into existing ANDF code.

We have used the scheme described in (3) to ensure the maximum consistency with the existing environments:

- A new `osc12` ANDF environment has been created on top of DRA's XPG/4 environment. *i.e.*: At this starting point, `osc12` is strictly equivalent to DRA's XPG/4 environment.
- A new file `<sys/ioctl.h>` has been added to this environment. This file defines the very minimum required by `/bin/ls`:
  - A definition of the `ioctl(int,int,...)` function.
  - A definition of the `TIOCGWINSZ` constant.
  - A definition of `winsize` as a structure name.

9. This is AES revision A. As mentioned above, this version is several years old, and a new version is in progress at OSF.

- A definition of `ws_col` as an `unsigned short` field of the `winsize` structure.

— Internationalization support

`ls` uses the function `mbswidth()` (declared in `<mbstr.h>`) to obtain the width, in screen columns, of a file name expressed in *multi byte strings* (*mbs*), as opposed to *wide character strings* (*wcs*). Both mbs and wcs provide the same level of functionality. However, the mbs strings are partially[10] supported by *de-jure* standards whereas wcs are more completely supported. No standards body specifies `<mbstr.h>`; it is inherited from AIX[11].

The dates of last modifications of a file are converted into a character string according to the current national language support status. The conversion of a `tm` structure into an string of characters requires a temporary buffer. The size of this buffer is defined by an OSF/1 constant named `NLTBMAX` available from OSF/1's `<time.h>` header.

The call to `mbswidth()` was replaced by the sequence:

- Convert the mbs string into a wcs string;
- Call the XPG/4 function `wcswidth()` with the converted wcs string to get the actual screen width of the original mbs string.

The header `<mbstr.h>` was replaced in the list of include files, by the `<wchar.h>` `<limits.h>` and `<stdlib.h>` headers.

The constant `NLTBMAX` was added to the `osc12` environment as part of the `<time.h>` header.

10. Only *partially* because [ANSI-C] and [XPG/4-IF] provides almost only functions to convert a mbs string from and to a wcs string, and no functions to *e.g.* convert a mbs string to a long integer. See [ANSI-C] and [XPG/4-IF] for details.

11. AIX is IBM's flavor of Unix.

— File status

The information about a file is obtained by a call to the `stat()` function (defined in `<sys/stat.h>`) which fills up a `stat` structure for a given file name.

The basic type of files and their related information available through the `stat` structure is described in [XPG/4-IF]. OSF/1 supports others types of files imported from others members of the Unix family:

Symbolic links. They are defined in [SVID-3] and [AES]. Both these definitions provides a `S_IFLNK` constant to distinguish a link from other files, but strange enough, none of them provide a `S_ISLNK()` macro to question if a given file is a symbolic link, similar to those they provides for others types of files. OSF/1 does provides these two macros in its headers files. The `stat()` function returns the status of the object pointed to by the link, as opposed to the status of the link itself. To query the status of the link itself, the `lstat()` function is defined in [AES] and [SVID-3]. ls.c uses the BUFSIZ constant as the size of the temporary which holds the path to the pointed file. This is not correct: The maximum size of a symbolic link is `PATH_MAX`.

Sockets, originally from BSD. In order to distinguish them from others kinds of files, OSF/1 provides for this purpose a constant: `S_IFSOCK` and a macro `S_ISSOCK()`.

The constants `S_IFLNK`, `S_IFSOCK`, the macros `S_ISLNK()` `S_ISSOCK()` and the function `lstat()` have been added to `osc12`'s `<sys/stat.h>` headers.

Occurrences of `BUFSIZ` was replaced with either `PATH_MAX` or `PATH_MAX+1` when they meant the size of the largest symbolic link or the size of the temporary to holds the name of a link respectively.

— Size of blocks

File sizes are obtained by a call to the `stat()` function.

This structure is only partially defined by standards bodies, and the size in number of blocks is part of the information that is defined in some standards [SVID-3] and not in others [XPG/4-IF][12].

`ls.c` includes the header `<sys/param.h>` to obtain the constant `REPORT_SIZE`, which is used as disk block size to compute the number of file system blocks occupied by files in a directory. As XPG/4 requires this constant to be 512 bytes ([XPG/4-CU], page 446, 1st paragraph) we have hard-coded this definition into the source code. The original `ls.c` defined `REPORT_SIZE` as the same as `UBSIZE` which may not have be 512 for all implementations. This constant was taken from a BSD inherited file: `<sys/param.h>`.

The field `st_blocks` of the `stat` structure has to be multiplied by `S_BLKSIZE` to get the actual size of disk space in bytes allocated to this file. [SVID-3][13] defines this field as being: *The total number of physical blocks of size 512 bytes actually allocated on disk* (quoted from [SVID-3], page 6-135).

The definition of `REPORT_SIZE` to 512 bytes is now directly in the file `ls.c`, and the header `<sys/param.h>` is no longer included.

Reference to the constant `S_BLKSIZE` has been removed and replaced by the hard-coded constant 512.

— Major and minor modes of devices

With the `-l` option, `ls` lists major and minor numbers of devices, that is the number of the driver and the number of the device respectively. Such information currently has no *de jure* definition. OSF/1 provides this

12. This is a major drawback and means that the `ls` command could not be written in a fully portable fashion, *i.e.* it is not possible to write an [XPG/4-CU] compliant version of `ls` using only information provided by the [XPG/4-IF] API.

13. It is worth noting that there is a small discrepancy in [SVID-3] concerning this field: In the `stat` structure fields list (page 5-70 and page 6-133), a C comment in front of the `st_blocks` fields says: `/* Number of st_blksize blocks allocated for this object */`, while the description of the field (page 6-135) says: *The total number of physical blocks of size 512 bytes actually allocated on disk.* The latter is correct.

information in two different ways depending on whether the code is to have access to the kernel structure or not[14]:

C Macros if the compiled code is kernel code, `ls.c` obtains these macros from the `<sys/types.h>` include file, which in turn obtains them from the `<machine/types.h>` header.

C functions in other cases.

In order to have a public API (*i.e.* outside of the kernel), the function mechanism has been chosen:

The two function `major()` and `minor()` have been added to the `<sys/types.h>` `osc12` header.

— *Obsolete features of de-jure APIs*

Some features are declared obsolete in a given version of an API. This means that these features might be widthdrawn in futures releases. However, to ensure maximum upwards portability of source code, many vendors continue to support the obsolete features of an API as long as they do not conflict with the current one. *e.g.*:

[XPG/3-IF] says that constants `S_IEXEC`, `S_IWRITE` & `S_IEXEC`, used by `ls.c`, have been withdrawn and replaced by `S_IRUSR`, `S_IWUSR` & `S_IXUSR` respectively. However, OSF/1 still provides these definitions in its header files.

`ls.c` also uses a feature that has been declared obsolete by XPG/3 and withdrawn from XPG/4: The `S_IVTX` mode of a file. This was used to indicate that an executable had to be kept as long as possible in swap space in order to have a faster start-up during the next invocation.

The obsolete features have been added to the `osc12` environment. *i.e.* the constants `S_IEXEC`, `S_IWRITE`, `S_IEXEC` & `S_IFVTX`.

14. If the macro `_KERNEL` is set, then one gains access to the kernel definitions.

— Built-in constants

There is a built-in constant that is logically part of the API used by `ls` but which have not been placed in any API: The size, in numbers of bytes, of a user login- or group-name. `ls.c` had a plain constant (*i.e.* 16) assuming that user names will not be longer than 15 bytes[15]. The typical usage for Unix is 8 bytes.

This plain constant have been changed into a constant `LOG_GRP_MAX` set to 15 at the beginning of the file.

As OSF/1 does not even provide such a constant in its headers, it has not been moved to an OSF/1 header. It is worth noting that this will be a limitation of the ANDFized `ls`. *i.e.* it would install properly but would not run properly on a platform having login names up to, say, 24 characters.

However, this constant is a candidate to be moved into a standard header as soon as it is supplied by a future API.

— List of messages issued by `/bin/ls`

To run properly into an internationalized environment, `ls.c` does not provide its error messages directly in the source code but accesses them at run time in a locale-specific file. They are referenced by means of message numbers found in the `"ls_msg.h"` header.

The point is: are those constants defining messages part of `/bin/ls` API?

A scenario for distributing `/bin/ls` in a both computer-language- and natural-language-neutral format, would be

- The executable `/bin/ls` as an ANDF package, and
- One message file for each supported natural language. The choice of the right message file would be an installation parameter.

15. 16 includes the trailing null char.

In such a scenario, the coherency between message numbers and their actual string counterparts are part of `/bin/ls` coherency as a whole, and therefore are not related to any API.

— *Visibility of names*

As `ls` is made of only one C module `ls.c`, all global variables have been declared `static`. As the C producer does not keep the name of `static` variables, their actual names are not provided in the '`.j`' file.

5. Status

The current status is as follows:

- All changes to the source code mentioned in this paper have been incorporated into `ls.c`.
- A new ANDF API for OSF/1, named `osc12`, based on DRA's `xpg4` have been created according to this paper. This environment has successfully generated ANDF headers and a token library for a i386 platform.

The sizes of the various files compare as follows[16]:

File	ANDF	`cc`/i386
Object	`ls.j` 20014	`ls.o` 29015
Executable	30563	33682

The ANDF generated executable is 10% smaller than the `cc` generated one.

16. OSF/1 native `cc` is derived from `gcc`.

6. Future work

Future work includes:

- Building of an installer for a OSF/1 mips platform based on the Gandf experiment carried out by Richard L. Ford at OSF-RI in Cambridge (See [OSF-RF-II-93] for details). This installer will allow a second installation of `/bin/ls`.
- ANDFization of other OSF/1 Commands.

7. Bibliography

On operating systems

[ANSI-C] Programming languages - C, from ISO.

ISO/IEC 9899;

[POSIX-IF] Information technology - Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API) [C language], from ISO.

ISO/IEC 9945-1; IEEE Std 1003.1.

[POSIX-SU] Information technology - Portable Operating System Interface (POSIX) - Part 2: Shell and Utilities, from ISO.

ISO/IEC 9945-2; IEEE Std 1003.2-Draft.

[XPG/3-IF] CAE Specification - System Interface and Headers, Issue 3, from X/Open.

ISBN: 0-13-685843-0. Prentice-Hall.

[XPG/4-IF] CAE Specification - System Interface and Headers, Issue 4, from X/Open.

X/Open Document number: C202; ISBN: 1-872630-47-2.

[XPG/4-CU] CAE Specification - Commands and Utilities - Issue 4, from X/Open.

X/Open Document number: C203; ISBN: 1-872630-48-0.

[SVID-3] System V - Interface definition - Issue 3, from AT&T.

[AES] Application Environment Specification - Operating System - Programming Interfaces Volume - Revision A, from OSF.

ISBN: 0-13-043522-48-8; Prentice-Hall.

On the ANDF technology

[DRA-XII-92] The TDF specification - Issue 2.0 Revision 1 - December 1992.

The Defence Research Agency, St. Andrews Road, Malvern, Worcestershire, WR14 3PS, United Kingdom.

[OSF-SM-I-93] Porting with ANDF by Stavros Macrakis, from OSF.

6th article of OSF's ANDF Collected papers, Vol. 1, January 1993.

[OSF-RF-II-93] GANDF: status and design by Richard L. Ford, from OSF.

2nd article of OSF's ANDF Collected papers, Vol. 2, March 1993.