Porting OSF/1 AD[1]

Randall W. Dean

Research Institute
Open Software Foundation
One Cambridge Center
Cambridge, MA 02142

(617) 621-7310
rwd@osf.org

September 27, 1993

Abstract

OSF/1 AD is a portable multicomputer operating system. This paper describes a series of steps which will aid an engineer in their efforts in porting OSF/1 AD to a new hardware platform. This paper assumes that there is an existing uKernel port for the platform, existing OSF IK binaries for the platform and a familiarity with the OSF build environment. Given these requirements, this document will go through the steps of modifying the emulator, the server and the build/configuration files of OSF/1 AD for a new hardware platform.

1. Assumptions

- There is or will be a uKernel port to the machine.
- There are OSF IK binaries for the machine.
- Access to existing AD ports for reference.
- Working build environment and understanding of it.

2. General

The list of files needed to be created is easily generated by looking in an existing ports machine subdirectory in both the emulator and server. To determine what changes need to be made to nominally machine independent files, I did the following:

```
find . -type f -exec egrep 'mips|860|386' /dev/null {} \; | grep '#if'
```

The guidelines below contain most of the non-include files in the machine specific directories and the changes to machine independent files requiring the most work. This list of changes is not meant to be exhaustive. The proceeding find command will help locate other places where changes may be necessary for a given porting effort. Compile time errors should help to locate all the include files which need to be created or modified. Link time errors should locate what functions are missing and need to be supplied.

[1]This research was supported in part by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Material Command (AFMC).

3. Emulator Specific

The emulator was a totally new creation for Mach 3.0. Since most of the code in the emulator was written from scratch, a better effort was made to isolate machine dependencies in machine dependent files. The only changes that are required to machine independent files show up under the signal and build sections below.

3.1. Handling of system calls

The file **emulator/machine/emul_vector.s** contains the function *emul_common()* which is the entry point for all syscalls coming from the kernel emulation vector redirection mechanism. This routine in conjunction with *emul_syscall()* in **emulator/machine/emul_machdep.c** vector syscalls to the appropriate routine within the emulator. *emul_common()* must save registers, switch stack, and call *emul_syscall()*. The global *emul_exit()* is unused and just a continuation of *emul_common()* which switches stacks back, restores registers and returns.

While *emul_syscall()* is in a machine dependent file, most of its functionality is machine independent. *emul_syscall()* is called from *emul_common()* with a pointer to saved registers. The format of the saved registers is machine specific and defined in the *emul_regs* structs in **emul_machdep.c**. The only other machine dependencies are the register in which the syscall resides, the registers used for the syscall return value, and the mechanism for restarting syscalls. All the remaining code can be lifted directly from other ports.

3.2. Emulator Initialization

The only machine dependent code called at emulator initialization time is *emul_setup()* in **emulator/machine/emul_machdep.c**. This routine sets the task emulation vector. This is a machine dependent routine because some architectures have some special syscalls outside of the range used by all.

4. Server Specific

This is a nonexhaustive list of changes that need to be made to server machine independent files and routines that need to be written in machine dependent files. For the most part, items specified in this section do not need corresponding work done in the emulator.

4.1. Core Dumps

The routine *machine_core_info()* in **server/machine/bsd_machdep.c** is used by the machine independent core dump code to get machine specific information. Any existing implementation of this routine will server as an adequate template for creating a new version.

4.2. Ptrace Support

The ptrace support is split up in two places, **server/bsd/mach_process.c** and **server/machine/bsd_machdep.c**. The code necessary in **mach_process.c** to define what registers there are and what they are is straight forward. There are a number of examples present. The remaining changes in that file effect what interface is used tp the machine dependent ptrace routines. These are currently conditionalized per machine instead of OSF1_SERVER which they should likely be since all ported architectures use the same interface. Just add the new architecture along with the existing ones to use the *ptrace_{set,get}_u_word()* interfaces.

bsd_machdep.c must define the machine dependent routines called from **mach_process.c**. The existing versions serve as adequate documentation for generating a new version.

4.3. Network support routines

The routines *in_cksum()*, *ntohl()*, and *ntohs()* must be defined per architecture. These routines can simply be acquired from any existing IK port for the architecture.

4.4. Device configuration and Naming

The **server/machine/conf.c** contains the initialization of cdevsw and bdevsw. These tables map unix major/minor device numbers to the internal routines which support them and in the case of physical devices, the kernel name for these devices. The major/minor numbers will correspond to the IK numbers for those devices. The kernel name will usually correspond to the unix name that resides in **/dev**. In most cases the routines specified will be the generic machine independent routines which reside in **uxkern/*_io.c**. In the cases where the generic routines do not suffice for a device, the convention is to right the cover routine and place it at the end of **conf.c**. In almost all cases this cover routine is short and invokes one of the generic routines supplied. The existing ports have entries for a wide variety of devices which should server as adequate example for writing a new **conf.c**.

4.5. Exception to Signals

The routine *machine_exception()* in **server/machine/machine_exception.c** maps exception codes to unix signals. Look at existing versions to generate knew version.

4.6. Server on Server Support

The file **server/machine/second_syscalls.s** and **server/machine/second_SYS.h** combine to the server with unix system call access to an underlying unix. Since the unix server implements *read()*, *write()*, *open()*, and *close()* it cannot be linked with libc.a to get access to these services. The **second_*** files supply *second_read()/second_write()* etc. These routines can be generated by combining the existing version with the equivalent libc for the architecture being ported to.

4.7. Exec

To support exec requires changes in **server/sys/exec.h**, **server/bsd/kern_exec.c** and **server/bsd/ldr_exec.c**. These changes relate to the specific variation of the standard format or the use of different standard formats for that machine. The changes to **ldr_exec.c** and **kern_exec.c** are nearly identical. **server/machine/mach_o_hdr.c** needs to be written since the mach_o header format is not machine independent.

Two routines are needed to specify the arguments and to specify the start address at exec time. These are *set_arg_addr()* and *set_entry_address()* in **server/machine/bsd_machdep.c**. These can be copied almost directly from existing ports.

5. Joint Emulator and Server

The functionality of a number of system calls and sub-systems is spread between the emulator and the server. It is often necessary to have the same definitions of data structures and constants in both locations. This section will describe what files must be changed/written in tandem to supply the necessary functionality.

5.1. Signals

Writing the necessary code to support signals is easily the most difficult part of the porting effort. Duplicate code appears in both the emulator and server which if not identical will not work.

Some architectures have a different number of arguments to some signal syscalls. This effects the files **emulator/bsd_user_side.c** and **server/conf/syscalls.master**.

The sigcontext structure must be defined in **server/sys/signal.h**. Some architectures use sigreturn, sigcatch, or sigtramp pointers in the uarea. This is found in **server/sys/user.h**. If there is a change here, there might need to be a corresponding change to **server/sys/ushared.h**. This will also show itself in the necessary reading and writing of this field in **server/bsd/kern_sig.c**.

Signals are delivered to a user thread in one of two ways: initiated by the user, forced by the server. The user initiates signal taking by calling *take_signals()* in **emulator/machine/emul_machdep.c**. Signals are forced on the user by *sendsig()* in **server/machine/bsd_machdep.c**. Both methods of taking signals must result in the user stack looking identical and compatible with the signal code in libc. Returning from a signal always occurs through *e_sigreturn()* in **emulator/machine/emul_machdep.c**. This must unwind the state created by the two signal delivery mechanisms.

Most ports build and return from signals in quite different ways. This is mostly due to history and compatiblity with IK's on the same platform. Generating this code usually requires pouring over existing ports, libc, and an IK port signal code for the given platform.

5.2. Fork

Fork requires that three routines be written, one in the server and two in the emulator. Fork starts with *e_fork()* in **emulator/machine/emul_machdep.c**. This routine bundles up the registers which the child will inherit and invokes the *bsd_fork()* RPC. The routine *thread_dup()* in **server/machine/bsd_machdep.c** copies the registers of the parent to the child. It also passes the pid and return code back in registers in a machine dependent way. This leads to the routine *child_fork()* in **emulator/machine/emul_vector.s**. This routine is only executed by the child. It must restore registers like *emul_exit()* does before beginning the return to the user. The existing ports of these routines should make clear how this all works.

5.3. Bcopy

It is necessary to build version of *bcopy()* in both the emulator and server which have the ability to recover from an exception. The files **emulator/machine/user_copy.s** and **server/machine/user_bcopy.s** have these routines. The existing versions will clearly document what needs to be written for a new port.

6. Build Related

A number of files need to be written/modified for the build to proceed correctly.

The file **emulator/emulator_base.c** creates a program which outputs the address that the emulator is to be linked at. This requires coordination with **server/machine/vmparam.h**. This should be relatively straight forward.

The machine type needs to be #defined in **server/src/config/config.h**. A matching string constant will need to appear in **server/src/config/config.y**

The following files need to be created in **server/conf**:

- MASTER.machine
- MASTER.machine.local
- Makefile.machine
- files.machine

Table of Contents