Porting Erlang/OTP to Exotic Platforms

Tips, tricks, and having some fun along the way

Brett Cameron

March 2015

Abstract

Over the past 6-7 years the speaker has had cause (sometimes self-inflicted) to port Erlang/OTP to several exotic and some not-so-exotic operating system platforms, including OpenVMS and several UNIX variants such as Tru64, HP-UX (Itanium), and AIX. In this talk, the speaker will discuss the general process of building Erlang/OTP from source and some of the challenges that are commonly encountered when porting Erlang/OTP to new platforms. These challenges range from trivial matters such dealing with minor differences between C runtime library and header file implementations on different platforms through to dealing with somewhat more complicated and problematical areas such as threading, SMP support, and various processor, operating system, and file system peculiarities. Specific issues will be considered, and solutions will be presented. In addition to porting Erlang/OTP to the new platform, for anything but the most trivial of cases some work will generally also be required to port Erlang/OTP applications to the new platform. Many Erlang/OTP applications include C/C++ driver code that must also be ported, and porting this code can sometimes prove almost as problematical as porting Erlang/OTP itself. Additionally, Erlang/OTP application code may contain operating-specific functionality. Some of the challenges faced when porting large and complex Erlang/OTP applications such as Riak, CouchDB, and RabbitMQ will be considered. The speaker will also briefly discuss future plans for Erlang/OTP on the OpenVMS platform.

About me

Brett Cameron works as a senior software engineer at VMS Software Inc. (VSI, http://www.vmssoftware.com/index.html), helping to define and implement the company’s Open Source strategy for the OpenVMS operating system. Before joining VSI Brett worked as a senior software architect with HP’s Cloud and Enterprise Services groups. Brett lives in Christchurch, New Zealand and has worked in the software industry since 1992, and in that time he has gained experience in a wide range of technologies, many of which have long since been retired to the software scrapheap of dubious ideas. Over the past decade Brett has spent considerable time travelling the world helping organisations to modernize their legacy application environments and to better leverage Open Source technologies. In more recent times, his involvement with various Open Source projects and his work in the cloud computing space has caused him to develop a liking for functional programming languages, and Erlang in particular, which he has ported to several operating systems, including OpenVMS. Brett holds a doctorate in chemical physics from the University of Canterbury, and maintains close links with the University, delivering guest lectures and acting as an advisor to the Computer Science and Electronic and Computer Engineering departments on course structure and content. In his spare time Brett enjoys listening to music, playing the guitar, and drinking beer.

AGENDA

- Introduction
- Existing implementations and ports
- Building Erlang from source - a quick refresher
- Porting Erlang to a new platform
- Porting complex Erlang applications
- Some results
- Summary/conclusions
- Questions

VMS Software, Inc. (VSI)

- Formed (formerly announced) 31st of July 2014
  - http://www.vmssoftware.com
  - Headquartered in Bolton, MA
  - Founded by the investors and principals of Nemonix Engineering (http://nemonix.com/)
  - Still ramping up
    - Currently ~35 staff, looking to have ~70 towards end of 2015

- License agreement with HP to develop, sell and support the OpenVMS operating system and layered products

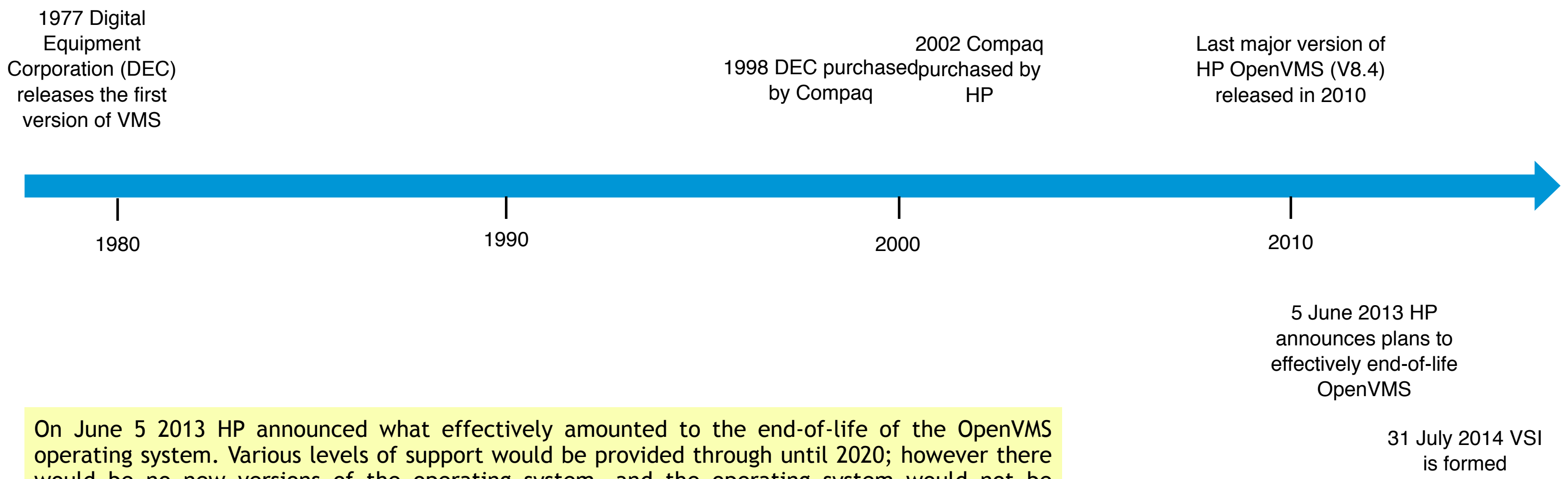
- Sole developer of future versions of OpenVMS
  - Big ticket item is porting the operating system to x86
  - Other key items on the roadmap include:
    - Making more Open Source technologies available (and supported)
    - Improved UNIX/Linux compatibility

© Gerrit Woertman

OpenVMS?

- A server operating system that runs on VAX, Alpha, and Intel Itanium processors (and soon x86)
- Originally developed by Digital Equipment Corporation (DEC)
- Proprietary
  - Not at all UNIX/Linux-like
- Unmatched security, stability, and legendary uptime performance (only 38 vulnerabilities in 37 years)
- Substantial and loyal installed base of over 300,000 systems supporting millions of users
- Used for numerous purposes across all sectors
  - Mail and other network services
  - Manufacturing
  - Transportation control and monitoring
  - Banking and financial services
  - Many large stock exchanges still run OpenVMS
  - Heath care
  - Telco
  - Government
  - Military
  - Chip manufacturers
  - ...

OpenVMS short history



1977 Digital Equipment Corporation (DEC) releases the first version of VMS

1998 DEC purchased by Compaq

2002 Compaq purchased by HP

Last major version of HP OpenVMS (V8.4) released in 2010

5 June 2013 HP announces plans to effectively end-of-life OpenVMS

31 July 2014 VSI is formed

On June 5 2013 HP announced what effectively amounted to the end-of-life of the OpenVMS operating system. Various levels of support would be provided through until 2020; however there would be no new versions of the operating system, and the operating system would not be supported on new hardware. This initial announcement was subsequently revised to extend some support dates; but the fact remained that the operating system was to all extents and purposes going end-of-life. Thanks to the incredible determination and hard work of several people, VSI was created to take over the ongoing development and support of OpenVMS and its layered products.

VSI plans to take OpenVMS into the future, porting the operating system to additional processor architectures, enhancing UNIX compatibility, and ensuring that Open Source technologies (such as Erlang) can be more readily ported to and fully utilized on the platform.

AGENDA

- Introduction
- **Existing implementations and ports**
- Building Erlang from source - a quick refresher
- Porting Erlang to a new platform
- Porting complex Erlang applications
- Some results
- Summary/conclusions
- Questions

Existing implementations and ports of Erlang

- From Erlang solutions (https://www.erlang-solutions.com/downloads/download-erlang-otp)...
  - Linux
    - Ubuntu, CentOS, Debian, Fedora, Raspbian
  - Mac OS X
  - Current versions of Windows (32-bit and 64-bit)

- Ericsson and Erlang Solutions have also built Erlang on a few other platforms
  - Solaris (including 64 bit)
  - FreeBSD
  - Tru64
  - Older versions of Windows
  - VxWorks and OSE real-time operating systems

- Builds for various other platforms have been reported…
  - QNX RTOS
  - IRIX
  - Assorted embedded devices running some variant of Linux
  - ...
  - Pretty much so long as you've got `gcc` and friends and enough compute resources, you should be able to build and run Erlang/OTP on anything that pretty much resembles Linux/UNIX

Some ports that I’ve been involved with

- AIX
  - This is going back a few years now
  - Ported for a RabbitMQ proof-of-concept for a (large) prospective customer
  - Used `gcc` et al as opposed to IBM AIX compiler suite
  - Sufficient to run RabbitMQ
  - Some rough edges
    - Terminal I/O
    - Branching in OTP based on return value of `os:type()`
    - No HiPE support

- HP-UX (Itanium)
  - Several HP teams were contemplating using RabbitMQ but wanted it on HP-UX
  - Fairly straightforward to get something working using `gcc` et al
  - Some issues encountered with threads, SMP, and in OTP
    - Will discuss some of these thing later

- Compaq Tru64
  - I was bored one evening
  - Similar results to HP-UX, although probably less issues overall

- HP OpenVMS (Alpha and Itanium)
  - Wanted an AMQP implementation for OpenVMS
    - RabbitMQ looked like the best option, but it requires Erlang
    - A friend/colleague dared me to try porting Erlang to OpenVMS
  - Numerous challenges
    - OpenVMS is not at all Linux/UNIX-like (although this can be fudged to some extent)
    - Probably on a par with porting Erlang to Windows
  - More on all of this later

- … and a couple of others

AGENDA

- Introduction
- Existing implementations and ports
- **Building Erlang from source - a quick refresher**
- Porting Erlang to a new platform
- Porting complex Erlang applications
- Some results
- Summary/conclusions
- Questions

Building Erlang from source

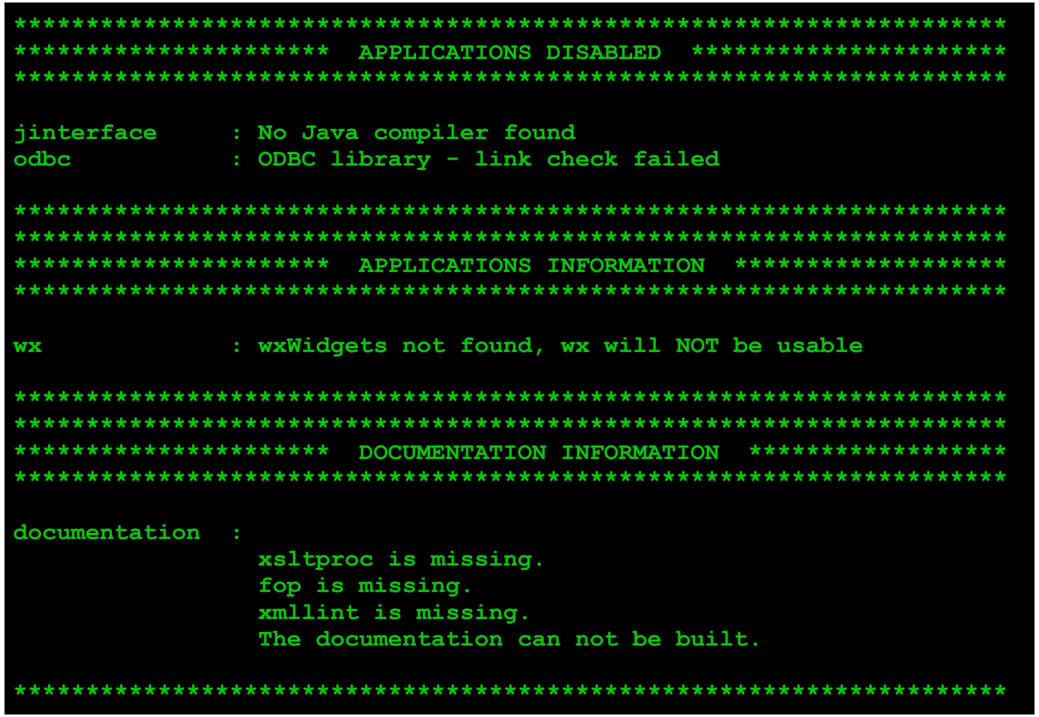
- Let's say we want to build Erlang/OTP from source on an Ubuntu Linux server...

```
$ sudo apt-get install build-essential
$ wget https://packages.erlang-solutions.com/erlang/esl-erlang-src/otp_src_17.1.tar.gz
$ gunzip otp_src_17.1.tar.gz
$ tar xvf otp_src_17.1.tar
$ cd otp_src_17.1
$ ./configure
$ make
$ sudo make install
```

Okay, well that was easy enough…

Let's look at the `./configure` step in a bit more detail…

Building Erlang from source



```
*********************************************************************
**********************  APPLICATIONS DISABLED  **********************
*********************************************************************

jinterface       : No Java compiler found
odbc             : ODBC library - link check failed

*********************************************************************
*********************************************************************
**********************  APPLICATIONS INFORMATION  *******************
*********************************************************************

wx               : wxWidgets not found, wx will NOT be usable

*********************************************************************
*********************************************************************
**********************  DOCUMENTATION INFORMATION  ******************
*********************************************************************

documentation    :
                   xsltproc is missing.
                   fop is missing.
                   xmllint is missing.
                   The documentation can not be built.

*********************************************************************
```

- When I ran `./configure` on my server, it gurgled away for a while checking various aspects of the environment and completed successfully with this output
- This is telling me which bits of Erlang/OTP will not be built because certain prerequisites are not installed
- If these components are required, it is just a matter of installing the relevant packages and re-running `./configure`

- But `./configure` also allows you to specify how you want Erlang to be built
  - Which components to include in the build and which to exclude
  - Enable or disable certain Erlang features
  - Installation directories
  - Compiler and linker flags
  - ...

Building Erlang from source - options

```
$ ./configure --help
```

- The above command lists the myriad of build options that are available
  - There are quite a few…

More on this topic later

The config.h file

- Running `./configure` generates a variety of bits and pieces required for the subsequent build (make)

- One of the more important files generated by `./configure` is `config.h`
  - This file is included by pretty much every C file
    - Deals with many of the platform-dependent bits and pieces
      - Header file differences
      - C RTL differences
      - …
    - Controls (to some extent) how things get built, what functionality is compiled in, ...

- Here's an example `config.h` (in this case generated on Ubuntu)

- When porting Erlang/OTP to a not particularly POSIX-like environment (such that you cannot do a `./configure`) it will be necessary to manually create an appropriate `config.h` or to modify one generated for another platform
  - Requires good knowledge of the C RTL
  - … more on this later

AGENDA

- Introduction
- Existing implementations and ports
- Building Erlang from source - a quick refresher
- **Porting Erlang to a new platform**
- Porting complex Erlang applications
- Some results
- Summary/conclusions
- Questions

The scale of the problem

- Appreciable (and generally quite complex) codebase
  - Some 500,000 lines of C code
  - Approximately 2,300,000 lines of Erlang code (including test suites) across ~4000 files
  - Sundry other files
    - Makefiles
    - Test scripts
    - …

Main components to be ported

- Executables
  - `beam`, `beam.smp`
  - `child_setup`
  - `epmd`
  - `erlc`
  - `erlexec` (run by the `erl` command)
  - `escript`
  - `heart`
  - `inet_gethost`
  - And a few other items (`dialyzer`, …)

- A few (~10) shared libraries
  - Number of shared libraries depends on the selected build options
    - Can also be somewhat platform-dependent
  - ODBC interface
  - WxWidgets
  - ASN1
  - Tracing
  - OpenSSL/crypto interface
  - …

Not surprisingly, `beam` is generally the most challenging component to port; however other bits and pieces are not without their challenges.

Good news and bad news

- Code and build environment are (mostly) POSIX-compliant
  - Fairly standard C code
  - Usual sort of build tools/environment
  - But what if you don't have a particularly POSIX-like environment?

- Also need to consider the OTP code
  - It's not just a matter of building the executables listed on the previous slide

So there's certainly some work to do, but it's possibly not as bad as might initially have been thought…

Typical issues and challenges

- Configuration options
  - Which options to use?
  - What if you can't do a `./configure; make`? *What the …?!*

- C code
  - Common problem areas include (but are not necessarily limited to)…
    - Threads
    - SMP
    - I/O and polling
    - Terminal I/O
    - Sockets
    - Anomalies ("bugs")

- OTP code
  - `os:type(), os:cmd()`

Let's look at a few examples…

Remember the configure script?

```
$ ./configure --help
```

- Some of the more important options to consider when porting Erlang to a new platform tend to be as follows (and probably a few others):

`--enable-threads, --disable-threads`	Enable/disable asynchronous thread support
`--enable-smp-support, --disable-smp-support`	*Enable/disable SMP support*
`--enable-kernel-poll, --disable-kernel-poll`	Enable/disable use of kernel poll
`--enable-sctp, --disable-sctp`	*Enable/disable inclusion of SCTP support*
`--enable-hipe, --disable-hipe`	Enable/disable inclusion of HiPE functionality
`--with-javac, --without-javac`	*With or without the Java interface*
`--with-termcap, --without-termcap`	With or without termcap libraries
`--with-ssl, --without-ssl`	*With or without SSL/TLS support*
`--without-wx`	Without WxWidgets
`--without-odbc`	*Without ODBC support (requires unixODBC, http://www.unixodbc.org/)*

- V r the initial build to
  p
  - ot and it is therefore
  - atures can be added in

Some examples…

AIX

```
$ ./configure --disable-smp-support --without-wx --without-javac --without-ssl --disable-megaco-reentrant-flex-scanner --disable-hipe
```

- Reasonably straightforward, assuming that GNU `make`, `gcc` and friends are available
  - Really do need GNU `make`
  - Most other `make` implementations (such as those that come out of the box with AIX and HP-UX) will not suffice
- Left out some of the tricky stuff for the initial build
- Using the AIX linker, so needed to tweak a few of the linker flags, particularly for building shared libraries
  - Use "`-Wl,-bexpall,-brtl`" instead of "`-Wl,-export-dynamic`"
  - Use "`DED_LDFLAGS = -G -bexapall -brtl`" instead of "`DED_LDFLAGS = -shared`"

OpenVMS

```
$ ./configure --enable-threads --disable-smp-support --disable-kernel-poll --disable-sctp --disable-hipe --without-termcap \
--without-javac --without-ssl
```

- Initial `./configure` was done on an Ubuntu server
  - Don't have (good) Bash shell or `autoconf` and friends on OpenVMS
  - More on this shortly…
- Don't bother with features that OpenVMS simply cannot handle (kernel poll, SCTP) and features that will be problematical (SMP, HiPE)
- Feeling brave by enabling asynchronous thread support
- SSL is easy to add in later
  - Subsequently also added in termcap support and currently working on SMP

For the record…

- There are something like 900+ unique macros in the C code used in `#ifdef`-type pre-processor statements to address platform differences and to control what gets compiled
- The values of many of the macros are determined by other high-level (configuration) macros
- But the fact remains that there are a lot of options and inter-dependencies that can cause fun and entertainment when porting the code to a new environment

SMP and CPU topology

- Erlang `beam.smp` determines CPU topology automatically at start-up
  - See `erts_cpu_info_update()` and other functions in `erl_misc_utils.c`
- In most cases (including OpenVMS) details regarding the number of cores configured and active can be determined via calls to `sysconf()` similar to the following:

```
#ifdef _SC_NPROCESSORS_CONF
    configured = (int) sysconf(_SC_NPROCESSORS_CONF);
    if (configured < 0)
        configured = 0;
#endif
#ifdef _SC_NPROCESSORS_ONLN
    online = (int) sysconf(_SC_NPROCESSORS_ONLN);
    if (online < 0)
        online = 0;
#endif
```

But on HP-UX things are a little different...

```
#if defined(hpux) || defined(__hpux) || defined(_hpux)
#  include <sys/pstat.h>
#elif...
.
.
.
#if defined(hpux) || defined(__hpux) || defined(_hpux)
    struct pst_dynamic psd;
#elif ...
.
.
.
#if defined(hpux) || defined(__hpux) || defined(_hpux)
    pstat_getdynamic(&psd, sizeof(psd), (size_t)1, 0);
    online = psd.psd_proc_cnt;
    configured = psd.psd_active_cores;
#elif ...
```

Threads

Firstly, some key points from Robert Virdings's excellent 2013 Erlang Factory talk "*Hitchhiker's Tour of the beam*":

- Schedulers
  - Semi-autonomous beam VM
  - One per VM thread
    - By default one VM thread per core
  - Contains its own run-queue
    - Run-queue contains things to be done
  - Run as separately as possible
    - Reduce nasties like locks/synchronisation

- Asynchronous thread pool
  - File I/O is done in the scheduler thread
    - It can take time
    - Blocks the scheduler while waiting
  - Using the asynchronous threads moves I/O operations out of the scheduler thread
    - Scheduler thread now no longer waits for file I/O
  - File I/O will automatically use them if created
  - Linked-in port drivers can use them if they exist
  - Inet driver never uses them

Threads

- For the most part no problems from a portability perspective (so long as you've got a decent POSIX threads library)
- Some typical issues include:
  - Missing functions
  - Thread stack size

http://www.reddit.com/r/aww/comments/2oagj8/multithreaded_programming_theory_and_practice/

Threads - missing functions

- The main one here tends to be `pthread_sigmask()`
- To quote the Linux Programmers Manual:

  "*The pthread_sigmask() function is just like sigprocmask(2), with the difference that its use in multithreaded programs is explicitly specified by POSIX.1-2001*"

- However on most platforms `sigprocmask()` is thread-safe and we can get away with the following sort of thing:

```
#ifdef __VMS
  return sigprocmask(how, set, oset);   /* Fingers and toes crossed */
#else
  return pthread_sigmask(how, set, oset);
#endif
```

Threads - per-thread stack size

Initially the SMP build of beam on OpenVMS would not even start, claiming it was out of stack space…

- Consider the following program that displays per-thread stack size:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
    pthread_attr_t attr;
    size_t stacksize;

    pthread_attr_init(&attr);
    pthread_attr_getstacksize(&attr, &stacksize);
    printf("%zd\n", stacksize);
    pthread_attr_destroy(&attr);
    return 0;
}
```

- Running this code on various platforms yields the following results (assuming default rlimit settings):
  - Ubuntu 14.04, 8388608 bytes
  - Cygwin (64-bit), 1036288 bytes
  - OpenVMS, 40776 bytes (!)
    - Whoa; that's possibly just a bit small... but as it turns out, not by that much

- Thread creation in `beam` is handled by the function `ethr_thr_create()` in `ethread.c`, which starts as follows:

```
int
ethr_thr_create(ethr_tid *tid, void * (*func)(void *), void *arg,
                ethr_thr_opts *opts)
{
    ethr_thr_wrap_data__ twd;
    pthread_attr_t attr;
    int res, dres;
    int use_stack_size = (opts && opts->suggested_stack_size >= 0
                          ? opts->suggested_stack_size
                          : -1 /* Use system default */);

#ifdef ETHR_MODIFIED_DEFAULT_STACK_SIZE
    if (use_stack_size < 0)
        use_stack_size = ETHR_MODIFIED_DEFAULT_STACK_SIZE;
#endif
.
.
.
```

- This essentially tells us that we can fix the thread stack size problem by defining the macro `ETHR_MODIFIED_DEFAULT_STACK_SIZE` in `config.h` to an appropriate value
- After a bit of trial and error it was found that 64K is enough on OpenVMS

Polling

- Descriptors of various types are polled using `select()` and `poll()`
  - Can optionally use kernel poll, if it's available

- Toggling between blocking and non-blocking
  - Can't use `fcntl()` on some platforms (OpenVMS, AIX, Windows) to toggle sockets or file descriptors blocking/non-blocking
  - Need to use `ioctl()` instead (or the Winsock equivalent thereof)
  - Erlang VM code knows about this
    - Need to ensure the `SET_BLOCKING()` and `SET_NONBLOCKING()` macros are correctly defined for the new platform in the `sys.h` header file

- On OpenVMS `select()` and `poll()` (currently) only work on sockets
  - But `beam` polls sockets, pipes, and potentially any other sort of descriptor
  - Need to implement our own versions of `select()` and `poll()` to handle this
    - Bit of a grubby hack
    - Performance implications
    - Generally not ideal

Use of os:type() and os:cmd() in OTP

- There are a significant number of `os:type()` and `os:cmd()` calls in the OTP library code
  - When porting Erlang/OTP to a new platform it is important to check these calls to determine whether any changes are required in order to accommodate the new platform and/or to avoid strange errors
  - Easy enough to find
  - Generally straightforward to fix for UNIX/Linux-like systems
  - May be somewhat more problematical to address for more exotic (non-Linux/UNIX) platforms

Let's look at a couple of examples...

Use of os:type() and os:cmd() in OTP

```
init([]) ->
    process_flag(trap_exit, true),
    process_flag(priority, low),

    OS = os:type(),
    PortMode = case OS of
                   {unix, darwin} -> false;
                   {unix, freebsd} -> false;
                   {unix, dragonfly} -> false;
                   % Linux supports this.
                   {unix, linux} -> true;
                   {unix, openbsd} -> true;
                   {unix, netbsd} -> true;
                   {unix, irix64} -> true;
                   {unix, irix} -> true;
                   {unix, sunos} -> true;
                   {unix, hpux} -> true;
                   {win32, _OSname} -> false;
                   _ ->
                       exit({unsupported_os, OS})
               end,
    Pid = if
              PortMode ->
                  spawn_link(fun() -> port_init() end);
              not PortMode ->
                  undefined
          end,
    ...
```

- See `lib/os_mon/src/memsup.erl`
- Initialization of system and process memory monitoring
  - Monitoring will fail to start on unsupported platforms
- Straightforward to add in support for HP-UX
  - A couple more similar changes are required elsewhere in this file

Use of os:type() and os:cmd() in OTP

```
%%--Check disk space----------------------------------------------------

check_disk_space({win32,_}, not_used, Threshold) ->
    Result = os_mon_sysinfo:get_disk_info(),
    check_disks_win32(Result, Threshold);
check_disk_space({unix, solaris}, Port, Threshold) ->
    Result = my_cmd("/usr/bin/df -lk", Port),
    check_disks_solaris(skip_to_eol(Result), Threshold);
check_disk_space({unix, irix}, Port, Threshold) ->
    Result = my_cmd("/usr/sbin/df -lk",Port),
    check_disks_irix(skip_to_eol(Result), Threshold);
check_disk_space({unix, linux}, Port, Threshold) ->
    Result = my_cmd("/bin/df -lk", Port),
    check_disks_solaris(skip_to_eol(Result), Threshold);
check_disk_space({unix, hpux}, Port, Threshold) ->
    Result = my_cmd("/usr/bin/df -l -k -P", Port),
    check_disks_solaris(skip_to_eol(Result), Threshold);
check_disk_space({unix, dragonfly}, Port, Threshold) ->
    Result = my_cmd("/bin/df -k -t ufs,hammer", Port),
    check_disks_solaris(skip_to_eol(Result), Threshold);
check_disk_space({unix, freebsd}, Port, Threshold) ->
    Result = my_cmd("/bin/df -k -l", Port),
    check_disks_solaris(skip_to_eol(Result), Threshold);
check_disk_space({unix, openbsd}, Port, Threshold) ->
    Result = my_cmd("/bin/df -k -l", Port),
    check_disks_solaris(skip_to_eol(Result), Threshold);
check_disk_space({unix, netbsd}, Port, Threshold) ->
    Result = my_cmd("/bin/df -k -t ffs", Port),
    check_disks_solaris(skip_to_eol(Result), Threshold);
check_disk_space({unix, sunos4}, Port, Threshold) ->
    Result = my_cmd("df", Port),
    check_disks_solaris(skip_to_eol(Result), Threshold);
check_disk_space({unix, darwin}, Port, Threshold) ->
    Result = my_cmd("/bin/df -i -k -t ufs,hfs", Port),
    check_disks_susv3(skip_to_eol(Result), Threshold).
```

- See `lib/os_mon/src/disksup.erl`
- The function `check_disk_space()` determines disk space by issuing a potentially platform-dependent command
- Straightforward to add in support for HP-UX
- Note that things are done quite differently for Windows
  - A port program (part of the Erlang/OTP distribution for Windows) is used
  - A similar solution might be required for other non-UNIX-like platforms
  - On OpenVMS spawning a sub-process to run a command is not very efficient and it might therefore be preferable to implement a port program that performs system calls to determine the required information

Operating system flavour

- `os:type()` returns the operating system family and flavour
  - For example `{unix,linux}`, or `{win32,nt}`
- The operating system flavour is determined by the function `os_flavor()` in `sys.c` via a call to `uname()`

```
void
os_flavor(char* namebuf,
          unsigned size)
{
    struct utsname uts;
    char* s;

    (void) uname(&uts);
    for (s = uts.sysname; *s; s++) {
        if (isupper((int) *s)) {
            *s = tolower((int) *s);
        }
    }
    strcpy(namebuf, uts.sysname);
}
```

- On HP-UX this returns `hp-ux`
  - But that little hyphen creates a bit of a problem
  - Remember all that code in the OTP libraries that checks the return value of `os:type()`
    - That code expects the flavour to be represented as an atom and atoms cannot contain hyphens
    - Enclosing in single quotes does not help in most places
  - The simplest solution is probably a bit of a grubby hack along the following lines…

```
void
os_flavor(char* namebuf,
          unsigned size)
{
    struct utsname uts;
    char* s;

#if defined(hpux) || defined(__hpux) || defined(_hpux)
    strcpy(namebuf, "hpux");
#else
    (void) uname(&uts);
    for (s = uts.sysname; *s; s++) {
        if (isupper((int) *s)) {
            *s = tolower((int) *s);
        }
    }
    strcpy(namebuf, uts.sysname);
#endif
}
```

Porting sometimes uncovers little anomalies…

What is wrong with the following piece of code (`erl_mmap.c`)?

```
#if HAVE_MMAP
#   define ERTS_MMAP_PROT                        (PROT_READ|PROT_WRITE)
#   if defined(MAP_ANONYMOUS)
#     define ERTS_MMAP_FLAGS                     (MAP_ANON|MAP_PRIVATE)
#     define ERTS_MMAP_FD                        (-1)
#   elif defined(MAP_ANON)
#     define ERTS_MMAP_FLAGS                     (MAP_ANON|MAP_PRIVATE)
#     define ERTS_MMAP_FD                        (-1)
#   else
#     define ERTS_MMAP_FLAGS                     (MAP_PRIVATE)
#     define ERTS_MMAP_FD                       mmap_state.mmap_fd
#   endif
#endif
```

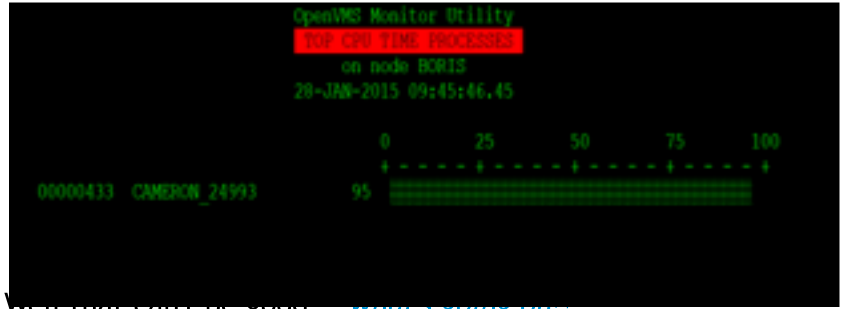
Well okay, not much really, but it probably should be as follows:

```
#if HAVE_MMAP
#   define ERTS_MMAP_PROT                        (PROT_READ|PROT_WRITE)
#   if defined(MAP_ANONYMOUS)
#     define ERTS_MMAP_FLAGS                     (MAP_ANONYMOUS|MAP_PRIVATE)
#     define ERTS_MMAP_FD                        (-1)
#   elif defined(MAP_ANON)
#     define ERTS_MMAP_FLAGS                     (MAP_ANON|MAP_PRIVATE)
#     define ERTS_MMAP_FD                        (-1)
#   else
#     define ERTS_MMAP_FLAGS                     (MAP_PRIVATE)
#     define ERTS_MMAP_FD                       mmap_state.mmap_fd
#   endif
#endif
```

- Most UNIX/Linux systems define both `MAP_ANON` and `MAP_ANONYMOUS`
  - However HP-UX does not
- Possibly a better solution would be to allow for either macro
- Various macros like `_POSIX_SOURCE`, `_BSD_SOURCE`, `_SVID_SOURCE`, and `_GNU_SOURCE` can also have a bearing on these sorts of things

Crazy (frustrating) stuff…

- Things appeared to be working pretty well on OpenVMS, but with the SMP build the Erlang VM seemed to be taking a disproportionate time to start...
- And even when not doing anything but sitting at the shell prompt strange things are observed....



```
                              OpenVMS Monitor Utility
                              TOP CPU TIME PROCESSES
                                  on node BORIS
                              28-JAN-2015 09:45:46.45

                                     0        25        50        75       100
                                     + - - - - + - - - - + - - - - + - - - - +
 00000433  CAMERON_24993        95   ####################################
```

- Well that can't be good... *What's going on..?*
- Profiling the code revealed that `beam` was spending pretty much all of its time in the function `erts_sys_main_thread()` in `sys.c`
  - This function doesn't really do too much
  - It basically just twiddles its thumbs in a while-loop waiting for a signal to arrive
  - Twiddling of thumbs is achieved via the call `select(0, NULL, NULL, NULL, NULL)`; a common way of implementing a thread-safe and interruptible sleep
  - Unfortunately this doesn't work on OpenVMS
    - The call returns immediately, causing the code to spin on the enclosing while-loop
    - Easily fixed by specifying an (effectively) infinite time value
    - But fun to track down…

One or two other portability issues…

- Use of `fork()`
  - Not available on some platforms
  - Only a small number of `fork()` calls in the Erlang code
  - Basically just `fork()`/`exec()` sequences that start other processes
  - Can typically substitute `vfork()`/`exec()`

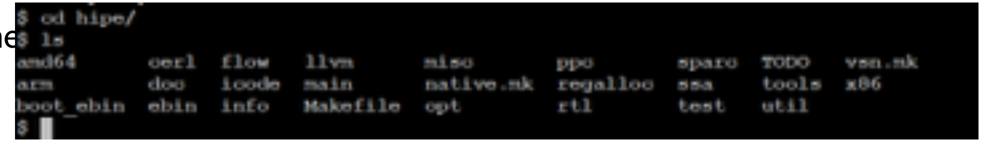
- File system differences
  - Not all file systems are created equal
  - Hierarchical or non-hierarchical
  - Different naming conventions
    - Different path separators
    - Mixed-case support (or the lack thereof)
    - Permitted characters
    - ...
  - Can be a major problem area when porting, particularly if no POSIX support is available

HiPE

- High Performance Erlang
- A "just-in-time" native code compiler for Erlang
  - Originally only SPARC and x86
  - A few other processor architectures are now supported
- Originally started around 1998 as a project at Uppsala University
- Fully integrated into Erlang/OTP since late 2001
- Ongoing research and development
  - Work with LLVM is particularly interesting (see http://www.erlang-factory.com/upload/presentations/519/erllvm.pdf)

- See http://www.it.uu.se/research/group/hipe/, http://www.slideshare.net/didip/high-performance-Erlang, and elsewhere for more information

- Bottom line:
  - If the platform you're porting Erlang/OTP to uses one of the currently supported processor types, HiPE might work
  - Otherwise, if you really want HiPE then you'll ne to learn some assembler



```
$ cd hipe/
$ ls
amd64      cerl  flow   llvm      misc       ppc       sparc  TODO   vsn.mk
arm        doc   icode  main      native.mk  regalloc  ssa    tools  x86
boot_ebin  ebin  info   Makefile  opt        rtl       test   util
$ 
```

Don’t panic!

What if you don’t have a particularly POSIX-like environment?

- No `make`, `gcc`, `m4`, `sed`, `autoconf`, binutils, ...
- My usual approach is to build Erlang on some other supported platform (such as Linux) and to capture a log of the output of the build process
  - Need to be sure to choose appropriate `./configure` options
- Carefully examine the `config.h` file created by the `./configure` step and modify it as appropriate for the target platform
- Convert the commands in the build log into an equivalent set of commands for the target platform
  - Keep the build procedure brain-dead simple (initially at least)
    - Easier to see what's going on
    - Can enhance it later
  - Systematically resolve compiler errors, compiler options, sort out include paths, linker options...
  - Incremental
  - Can be a bit of trial and error
- Identify and resolve platform differences (some of which are discussed on other slides)
  - C RTL
  - OTP libraries
- Cross fingers
- Test, rework, test, rework, …

AGENDA

- Introduction
- Existing implementations and ports
- Building Erlang from source - a quick refresher
- Porting Erlang to a new platform
- **Porting complex Erlang applications**
- Some results
- Summary/conclusions
- Questions

Typical problem areas

- Use of `os:type()`, `os:cmd()`
  - Use of operating system-specific commands
  - Subtle (and not so subtle) syntactical differences across UNIX/Linux implementations
  - Need to search code for these sorts of things and potentially modify code for the new platform

- Native Implemented Function (NIF's) and ports
  - Not all C/C++ code is created equal
  - Compiler differences
  - Building shared libraries (linker differences)
  - RTL and header file differnces
  - Other nasty stuff
    - Inline assembler
    - Other platform-specific code
    - ...

- Command procedures and environment variables

Many of the issues encountered when porting large and/or complex Erlang applications to new platforms are (perhaps not surprisingly) similar to those encountered when porting Erlang itself.

RabbitMQ

- http://www.rabbitmq.com
- 100% Erlang – no NIF's
- But…
  - Uses `os:cmd()` for various purposes
    - Determining how much memory RabbitMQ can use (if this is not explicitly set)
    - Determining and monitoring disk space
  - For example:

```
get_disk_free(Dir) ->
    get_disk_free(Dir, os:type()).

get_disk_free(Dir, {unix, Sun})
  when Sun =:= sunos; Sun =:= sunos4; Sun =:= solaris ->
    parse_free_unix(rabbit_misc:os_cmd("/usr/bin/df -k " ++ Dir));
%% Assumes queues stored under rabbitmq$root
get_disk_free(_Dir, {unix, openvms}) ->
    vms:freeblocks("rabbitmq$root") * 512;
get_disk_free(Dir, {unix, _}) ->
    parse_free_unix(rabbit_misc:os_cmd("/bin/df -kP " ++ Dir));
get_disk_free(Dir, {win32, _}) ->
    parse_free_win32(rabbit_misc:os_cmd("dir /-C /W \"" ++ Dir ++ "\"")).
```

Note that in this example, the function for OpenVMS uses a NIF to determine available disk space at the specified location. Implementing a NIF was just as easy as parsing the output of the OpenVMS equivalent of the `df` command, and somewhat more elegant!

- Even commands like `df` can have slightly different syntax across UNIX/Linux implementations
- And there is no `df` command at all on OpenVMS
  - … unless you're using GNV (GNU's Not VMS), but that's another story

RabbitMQ - determining total memory

```
.
.
.
get_total_memory({unix,openvms}) ->
    PageSize  = list_to_integer(os:cmd("write sys$output F$GETSYI(\"MEMSIZE\")") -- "\n"),
    PageCount = list_to_integer(os:cmd("write sys$output F$GETSYI(\"PAGE_SIZE\")") -- "\n"),
    PageCount * PageSize;
.
.
.
get_total_memory({win32,_OSname}) ->
    [Result|_] = os_mon_sysinfo:get_mem_info(),
    {ok, [_MemLoad, TotPhys, _AvailPhys, _TotPage, _AvailPage, _TotV, _AvailV],
     _RestStr} =
        io_lib:fread("~d~d~d~d~d~d~d", Result),
    TotPhys;

get_total_memory({unix, linux}) ->
    File = read_proc_file("/proc/meminfo"),
    Lines = string:tokens(File, "\n"),
    Dict = dict:from_list(lists:map(fun parse_line_linux/1, Lines)),
    dict:fetch('MemTotal', Dict);

get_total_memory({unix, sunos}) ->
    File = cmd("/usr/sbin/prtconf"),
    Lines = string:tokens(File, "\n"),
    Dict = dict:from_list(lists:map(fun parse_line_sunos/1, Lines)),
    dict:fetch('Memory size', Dict);

get_total_memory({unix, aix}) ->
    File = cmd("/usr/bin/vmstat -v"),
    Lines = string:tokens(File, "\n"),
    Dict = dict:from_list(lists:map(fun parse_line_aix/1, Lines)),
    dict:fetch('memory pages', Dict) * 4096;

get_total_memory(_OsType) ->
    unknown.
```

- In this case we're parsing the output of the OpenVMS lexical function `f$getsyi()` to determine available memory
- The return value of each call to `f$getsyi()` is a single integer value, so not much parsing is required
- But spawning sub-processes to do this sort of thing is not really very efficient…

Maybe Erlang/OTP should enhance built-in support for some of these more common operating system interactions…

Riak

- Not 100% Erlang
- SpiderMonkey JavaScript engine
  - https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey
  - Used to be quite straightforward to port (going back a few years)
  - Recently updated and enhanced...
    - Now a bit more challenging to port ☹
- LevelDB and Bitcask backends
  - Bitcask backend is fairly vanilla C code and straightforward to port
  - LevelDB code is particularly evil... or possibly I'm just allergic to C++
    - Heavily optimized (understandably)
      - Test suite able to totally swamp an old HP rx2660 (CPU and I/O) running OpenVMS
    - Atomic operations
      - Code for this stuff tends to be quite platform-specific
      - Inline assembly code
      - Platform-specific builtins
    - Some minor C RTL differences to contend with when porting to OpenVMS
    - Fun and games with C++ and shared libraries on HP-UX ia64
    - LevelDB backend requires SMP beam
      - … not quite sure how performant Riak would be with non-SMP beam!

Use of os:cmd() in Riak

- Minimal but significant
- For example, Riak uses the following code to check whether a particular process exists:

```
os_pid_exists(Pid) ->
    %% Use kill -0 trick to determine if a process exists. This _should_ be
    %% portable across all unix variants we are interested in.
    [] == os:cmd(io_lib:format("kill -0 ~s", [Pid])).
```

  This might be portable across all (most) Linux/UNIX variants, but it certainly will not work on non-Linux/UNIX platforms like Windows or OpenVMS

- There are also one or two "`ls -l`" commands, and "`rm -rf`" commands used in various pieces of test code

AGENDA

- Introduction
- Existing implementations and ports
- Building Erlang from source - a quick refresher
- Porting Erlang to a new platform
- Porting complex Erlang applications
- Some results
- Summary/conclusions
- Questions

Some results

- Main focus in recent times has been around porting Erlang to OpenVMS

- Generally positive results

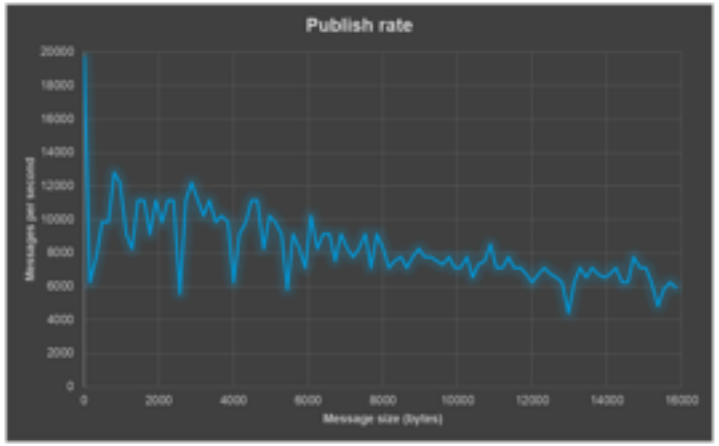
- Working ports of several releases of Erlang/OTP
  - A few issues to resolve, but generally the OpenVMS ports are stable and work well
  - SMP and full 64-bit support are probably the two biggest areas requiring refinement

- Working (or mostly working) ports of several sophisticated Erlang applications
  - RabbitMQ
  - Yaws (works very nicely)
  - CouchDB
  - Riak

- Performance is "satisfactory"
  - Largely limited by performance of Erlang/OTP TCP/IP driver code on OpenVMS
  - Kludged versions of `poll()` and `select()` are a major limiting factor
  - Needs work

Some results – RabbitMQ performance

- Can achieve message rates (publish and consume) of up to ~19000 messages per second on an old dual 1.6GHz CPU rx4640 server using small messages
- Rates drop of rapidly with increasing message size
- This sort of result is actually not bad for OpenVMS
- But is still well below what can be achieved on a comparable Linux environment
- Mentioned `poll()` and `select()` issues previously
- OpenVMS TCP/IP driver needs work generally (we're working on it)
- No problems with clustering, Mnesia, …



AGENDA

- Introduction
- Existing implementations and ports
- Building Erlang from source - a quick refresher
- Porting Erlang to a new platform
- Porting complex Erlang applications
- Some results
- **Summary/conclusions**
- Questions

Summary

- The `beam` VM and other runtime components (`epmd`, `inet_gethost`, ...) are written in standard C
- Porting to any UNIX/Linux platform should usually be fairly straightforward, particularly if `gcc` and related GNU tools are available
  - Generally minor (but potentially subtle) changes are required
    - C RTL and header file differences
  - Important to check `os:type()` and `os:cmd()` calls in Erlang library code

- Porting to non-UNIX/Linux platforms is more problematic, but may not be as difficult as might be initially thought
  - Most C compilers and C RTL's will be up to the task
  - File system differences will often be the biggest challenge
  - Other types of I/O may also present some problems
  - Overcoming a lack of GNU tools may require some improvisation
  - Take an incremental approach
    - Get something simple working then add more features

- Porting HiPE to a new processor type is non-trivial
  - Thankfully HiPE is not essential (although it may be nice to have)

- Many of the same considerations apply to porting Erlang/OTP itself and complex Erlang applications

Summary – future plans

- Want Erlang/OTP to be part of our Open Source strategy for the OpenVMS operating system

- Have a working OpenVMS port
  - A bit rough around the edges and there are a few loose ends to tidy up, but have proved viability

- Having Erlang on OpenVMS makes sense
  - Similar design goals with respect to fault tolerance, availability, ...
  - Of relevance to existing OpenVMS users
  - Can help to attract new users
  - Might even look at adding support for OpenVMS ICC (Intra Cluster Communication) and a few other things

vms Software

AGENDA

- Introduction
- Existing implementations and ports
- Building Erlang from source - a quick refresher
- Porting Erlang to a new platform
- Porting complex Erlang applications
- Some results
- Summary/conclusions
- Questions

Thank you