January 1996

OS-9 International

Vol. 5, Issue 1

EFFO S 9

European Forum For OS-9

8606 Greifensee, Switzerland
Fax +41 1 940 38 90
email os9int@effo.ch

sFr. 10.00
ISSN: 1019-6714

startup

Question: It is there but it is not there, what's that?

Another question: What computer hardware is so much delayed that even software for it was available earlier?

Answer to both questions: Motorola's new flag ship MC68060.

In fact, production of the MC68060 is so much delayed that the corresponding OS-9 version (3.0.2) was ready for shipment at a time when even not a single processor chip was available to the normal mortal. Today, only a few hand-selected samples are making their way to CPU board manufacturers.

One of the EFFO activists is in the rare and fortunate position to own an MC68060-based VMEbus development system. This system was used for testing purposes and also triggered the upgrade of processor-dependent OS-9 International software (refer, for example, to the article on the *cpucache* program in this issue). The honour goes to Microware that OS-9 V3.0.2 for the MC68060 was so stable and performed so well that the responsible person at Microware could decide – although not planned initially – to release it officially.

On average, the MC68060 is two to three times faster than an equally clocked MC68040. The term "equally clocked", of course, means that an MC68060 with an internal clock frequency of 50 MHz is compared to an MC68040 with an external clock frequency of 25 MHz. Motorola has adopted Intel's way to specify a chip's maximal clock frequency, although the average Motorola customer may be more difficult to fool with such PR tricks than the average PC customer.

This increase in performance is achieved by a smaller number of cycles required to execute many of the integer instructions and, mostly, by more efficient and additional on-chip caches. The fact that a large part of the 060's performance is based on very sophisticated cache techniques, makes it relatively difficult to predict the performance increase that can be obtained by migrating from the MC68040 to the MC68060. The often-cited 100 MIPS can only be achieved, if a code segment is running in a closed loop that entirely fits into instruction cache. If it does not (e.g. if the code is much longer than 8 kByte instruction cache), the MC68060 benchmarks about 22 MIPS, which is exactly the same value as obtained on an MC68040.

Anyway, the average increase of about a factor of two to three is very impressive, and, since the MC68060 is pin-compatible to the MC68040 (except that a DC/DC converter is required to deliver 3.3 V power supply), existing 040-boards may be upgraded easily. Many people would, therefore, be more than happy, if the MC68060 became finally available in production lots. Unfortunately, it is not only the MC68060 that is nearly impossible to obtain: recently, someone wanted to buy a single MC68040 – he was told that delivery time is now 30 weeks!

Good old pianola maker – do we really merit to be treated so badly?

Carsten Emde

Peter Dibble at CERN

Martin Merkel

Peter Dibble, computer scientist at Microware, visited CERN on October 9, 1995. The present article summarises the main topics of his talk.

Reorganisation of Microware

In 1995, Microware restructured the general activities of the company. Additional to the known tools and system software sectors, there are now departments working in the field of new media and consumer electronics. The latter mainly covers wireless communication like cellular telephones. This reorganisation was triggered by the needs of markets like smart consumer electronics that Microware is targeting now in addition to their traditional embedded controls market. All of this is also important with respect to OS-9, since there will be some cross-over to the systems software sector, particularly from the new media department. Peter mentioned the Serial Protocol File Manager (SPF) that supports various protocols such as ATM and that is a layered file manager similar to Unix streams.

Peter also mentioned the Multimedia Application User Interface (MAUI) that has been designed primarily for consumer electronics – a field that traditionally relies on small executables and fast response. MAUI, however, is not intended to be used in the context of a typical windowing environment. Other file managers developed for multimedia applications like the MPEG File Manager (MPFM) are less likely to be of general use for the established OS-9 market.

Current Research

The newly targeted consumer electronics industry could not decide in favour of OS-9, unless Microware was able to safely exclude also rarely occurring errors. Consumer electronics products are sold in high quantities so that even rarely and intermittently occurring errors may become noticeable. This type of error is often based on timing problems or race conditions (e.g. signals or interrupts re-enabled at the wrong time). Therefore, during the last 12 months Peter's research activities focused on the development of appropriate test procedures.

Recent History

Tool kits

In September 1995, Microware has supplemented the already known Ultra C tool kit by the new Ultra C++ tool kit. Both packages contain Microware's ANSI C compiler, a source level debugger, an enhanced shell, a printer spooler and the manual set. The Ultra C++ tool kit additionally includes the recently released Ultra C++ compiler that has been Plumb Hall validated and that supports templates and exceptions. Furthermore, the source level debugger *srcdbg* has been updated to allow for source level debugging of C++ code; unfortunately, it does not yet evaluate C++ expressions. Ultra C++ is shipped with the Rouge Wave tools.h++, the complex and IOStream packages. The Ultra C compiler itself has been enhanced with an MC68060 target option and specific compiler optimisations for this new processor type. New is also an *fpsp* library, which executes approximately twice as fast as the *fpsp* exception handler module. Due to many new library routines, the Ultra C library reference manual has grown significantly.

FasTrak for Windows

Microware's cross-development tool FasTrak is now available for both UNIX and Microsoft Windows platforms. The Windows versions will be synchronised with the UNIX releases. The next FasTrak for Windows release will run additionally on Windows '95 and Windows NT platforms. However, FasTrak for Windows is somewhat limited in its functionality by the operating system.

OS-9 3.0.1

The 3.0.1 maintenance release is finally available for all currently supported platforms except the Motorola MVME162FX systems, which will be supported from 3.0.2 onwards. The 3.0.1 release fixed some important bugs in the kernel e.g. with the *sleep* call [1] and system state time-slicing.

Power PC Support

Microware is now in a second beta test period with OS-9(000) for the Power PC. After the first beta test last winter, they shipped a pre-release this summer, which has been used by customers but was not regarded as absolutely stable. The following table summarises the main Power PC variants and Microware's support plans:

- 601: supported (but will be dropped as they have no hardware to support)
- 602: no platform on hand
- 603: supported
- 604: not supported yet, but planned for near future
- 403: supported
- 801: planned for near future

Near Future

UCC++

The Ultra C++ compiler is currently implemented as a two-part front end. This approach will be dropped with the next Ultra C++ release in favour of a single-layer front end. Furthermore, it is planned to improve and enhance debugging functions, to implement the standard template library (STL) that was proposed to the ANSI committees by Hewlett-Packard and, finally, to fix reported bugs of the first release.

Posix.1

Microware will integrate the Posix 1003.1 library developed by RTSoft in Russia into the Ultra C compiler. The exact procedure of this implementation has still to be decided by the Marketing Department. With the exception of fork this library implements the full Posix functionality as function calls layered on top of the existing Ultra C libraries. This integration is planned to be completed by end of 1995. Testing this library extension is not straightforward, because the Posix validation kit relies on the implementation of the *fork* system call. An implementation of the Posix real-time extensions might follow later.

Modular ROMs

The Boot ROMs have undergone a major restructuring, basically to provide support for FasTrak. In addition, there will be some sort of network port available in the low-level (boot) software. Also the "modular ROM" structure is supposed to be as flexible and dynamically configurable as the operating system itself. The modular ROMs will contain:

- New boot software
- FasTrak debugging server
- Low-level I/O for debugging

The low-level I/O support can be added to the running system after boot time.

FasTrak

The next FasTrak release will allow both C and C++ source level debugging of user and system state code. This will include file managers, drivers and even interrupt service routines. Also support for emulators other than HP products is planned. This is going back to a request from customers that required support for less expensive emulators. For now, only emulators for 68k family processors are supported.

Full Power PC Support

Currently, Microware is shipping embedded software for OS-9/Power PC. Full support (disk-based and extended versions) including RBF and ISP is in beta test now. The final release will follow very soon.

Improved SCSI Performance

SCSI II uncovered some timing problems in the Microware SCSI drivers. They have been solved, allowing SCSI devices to be run close to the rated transfer speed of the drive.

ISP

The next ISP release(s) will become more robust and faster. On the other hand, they are not yet up to the latest Berkeley releases. They will, however, support Simple Network Management Protocol (SNMP), and Domain Name System (DNS). In contrast to an earlier announcement, the so-called remote utilities will not be included in this release.

Motif

Microware will bring their X11 and OSF/Motif implementations up to revisions R6 and 1.2.4.

Far Future

Disks Greater than 4 GByte

Microware is working on support for disks greater than 4 GByte, which involves both work on RBF and on the OS-9 utility programs. The problems are that C language I/O assumes 32 bit signed offsets and the OS-9 I/O system is limited to a file size of 32 bit unsigned (4 GByte).

Asynchronous I/O

Peter started to look into the possibility of implementing asynchronous I/O, since this is needed to support threads. Microware will try to integrate this feature into *IOMan*, and it will probably be implemented using a simple I/O language. Threads will also require non-blocking versions of blocking system calls like for example *wait*.

PCF

Peter will add support for Windows '95 long file names.

Very Far Future

More Processors

Within the next two years, Microware will provide support for processors other than 68k, 80x86, Pentium and Power PC. However, Peter felt unable to make any clear announcements what processors these will be.

FasTrak

Microware's development environment will be enhanced with a long list of new features such as language-sensitive editors, source browsers and other development tools.

C++

Microware plans to add various OS-9 specific and also general real-time and embedded objects. They will ship standard libraries beyond STL, ANSI C++, IOStream and tools.h++.

Power PC Performance Measurements

The final part of Peter's talk covered performance measurements he made using the OS-9 port to the Power PC 603. These results have been presented in detail at the Open Bus Systems conference in Zurich, October 11 to 13, 1995. His findings can be summarised as follows:

- Typically, a 66-MHz PPC 603 performs two to five times faster than a 25-MHz MC68040.
- The worst-case performance of the PPC 603 is about one sixth of the MC68040.
- Taken in context, the interrupt response time of the PPC 603 is surprisingly good.
- More research is still required.

Reference

[1] Forster B (1995) *OS-9 Version 3.0 - What Is New ?* OS-9 International 3(1): 9-12.

Martin Merkel is OS-9 co-ordinator at the European Laboratory for Particle Physics, CERN. He can be reached at the Computing & Networks Division, Microprocessor Support. His email address is either <Martin.Merkel@CERN.CH> or <os9.support@cern.ch>.

Peter Dibble can be reached by email at <dibble@microware.com>.

A Post-Mortem Debugger for OS-9

Carsten Emde

Introduction

One of Murphy's laws on software development states that a program behaves normally whenever running under the control of a debugger and that it starts to bomb and to do other nasty things as soon as it is shipped to a customer. In order to obtain debugging information, even if the program is not under the control of a debugger and even if it has not been forked from a user's shell, a tool would be required that always provides relevant data about the state of a crashed program. A so-called "post-mortem debugger" can be used as a tool for the above scenario. The following article describes its implementation under OS-9.

History

The first version of a post-mortem debugger for OS-9 that was integrated into the *cstart.r* module was realised by Peter Sager at Advanced Systems Software but it was never released as a generally available product. Based on this idea, a post-mortem debugger was written that does not rely on the *cio* trap handler library, only uses low-level I/O and takes into consideration the needs of CC V3.2, GNU C V2.5.8 and UCC V1.2. In addition, the module was written in such a way that it runs independently from specific compiler options such as enabling trap handlers or ANSI mode and that also floating-point exceptions are enabled. The output format, however, was kept as close as possible to Peter Sager's original version.

Installation

Prior to installing the post-mortem debugger, it is recommended to backup the existing *cstart.r* file to *cstart_orig.r*. The post-mortem debugger can then be installed by entering

```
copy -rf /d0/1_96/PMD/cstart_pmd.r -w=/dd/LIB
```

It is enabled, i.e. any subsequently linked program is equipped with the post-mortem debugger, if the command

```
copy -rf /dd/LIB/cstart_pmd.r /dd/LIB/cstart.r
```

is entered, and it is disabled, if the command

```
copy -rf /dd/LIB/cstart_orig.r /dd/LIB/cstart.r
```

is entered. If a shell is used that supports aliases, the following two alias definitions may be helpful:

```
alias pmdon copy -rf /dd/LIB/cstart_pmd.r /dd/LIB/cstart.r

alias pmdoff copy -rf /dd/LIB/cstart_orig.r /dd/LIB/cstart.r
```

From OS-9 V3.0 onwards, the above aliases must read

```
alias pmdon copy -rf /dd/MWOS/OS9/68000/LIB/cstart_pmd.r /dd/MWOS/OS9/68000/LIB/
cstart.r

alias pmdoff copy -rf /dd/MWOS/OS9/68000/LIB/cstart_orig.r /dd/MWOS/OS9/68000/
LIB/cstart.r
```

Technical Description

The *cstart* modules that contain the post-mortem debugger installs exception handlers for the following conditions:

Exception	OS-9 error	Description
2	102	Bus error
3	103	Address error
4	104	Illegal instruction
5	105	Integer division by zero
6	106	CHK, CHK2 instruction
7	107	Unimplemented trap
8	108	Privilege violation
10	110	Unimplemented A-line op code
11	111	Unimplemented F-line op code

In addition, exception handlers for the following floating point exceptions are installed:

Exception	OS-9 error	Description
48	148	Branch or set on unordered condition
49	149	Inexact result
50	150	Divide by zero
51	151	Underflow
52	152	Operand error
53	153	Overflow
54	154	Signalling not a number

By default, the floating point control register *fpcr* has the exception enable byte (bit 8 to 15) cleared so that floating point exceptions are not taken. The *fpcr* bits must be set according to the following table, in order to enable the post-mortem debugger's floating point exception handling:

Bit	Description
8	Inexact decimal input
9	Inexact operation
10	Divide by zero
11	Underflow
12	Overflow
13	Operand error
14	Signalling not a number
15	Branch/Set on unordered condition

In case one of the named exceptions is taken, the name of the executed function, offset of the offensive instruction, calling sequence of the function, register dump and specific data such as the address that caused a bus error are written to the error path. This path is best redirected to a terminal or to a disk file. This ensures that it can be retrieved even by non-experienced people (e.g. at the factory) and made available to the responsible programmer.

Example

Considering the short example program

```
#define CRASHADDR 0xcecececece
#include <stdio.h>

main()
{
  printf("Calling crashfunc()...\n");
  crashfunc();
}

crashfunc()
{
  unsigned char *crashaddr = (unsigned char *) CRASHADDR;
  unsigned char result;

  result = *crashaddr;
}
```

that is compiled and linked with the command

```
cc crash.c -g
```

so that the program binary *crash* and the symbol table file *crash.stb* are produced. Without the post-mortem debugger, the program would simply abort with exit code #102, and the shell would decode this information as bus error, if enabled to do so. If, however, the post-mortem debugger is used, the program provides a lot of information as shown in the following example output:

```
Calling crashfunc()...

*** ACCESS VIOLATION (BUS ERROR) ***

    violation address: $cecececе

PROCESS REGISTER STACK IMAGE:
 PC = $01973044    SR =     $0018    USP= $019666ea    SSP= $01965456
 D0 = $019666a6    D1 = $0196567c    D2 = $00000001    D3 = $00000003
 D4 = $00000000    D5 = $000004aa    D6 = $00001bc0    D7 = $00000000
 A0 = $cecececе    A1 = $019727be    A2 = $01966b0a    A3 = $01966b02
 A4 = $01966afe    A5 = $019666f8    A6 = $0196d000    A7 = $019666ea

SUBROUTINE TRACEBACK:
 trapped in       crashfunc              at offset $00000020
 called by        main                   at offset $00000020
 called by        _cstart                at offset $00000102

Error #000:102  (E$BusErr) A bus trap error occurred.
```

The following program forces exception processing of the FP-divide-by-zero vector:

```
main()
{
  enable_fpexcpt();
  fbyzero();
}

fbyzero()
{
  double a = 1.0, b = 0.0;

  a = a / b;
}

#asm
EXCPT_MASK equ $ff00
enable_fpexcpt:
  fmove.l fpcr,d0 ; get floating point control register
  or.l #EXCPT_MASK,d0
  fmove.l d0,fpcr ; enable all fp exceptions
  rts
#endasm
```

If linked to the post-mortem debugger, the program produces the output

```
*** FP DIVIDE BY ZERO ***
```

followed by process register image and subroutine traceback as in the above example. The shell may add the OS-9 error message, if enabled to do so:

```
Error #000:150  (E$FPDivZer) floating point coprocessor divide by zero
```

Limitations

For the time being, there is no built-in interface from the post-mortem debugger to the source level debugger *srcdbg*. It is, nevertheless, possible to directly display the source line in error by using the *dil* command (disassemble and list) within *srcdbg*:

```
SrcDbg: fo crash
Forked: "crash"
Reading symbol file "crash.dbg".
crash.c
Reading symbol file "crash.stb".
File: "crash.c"
Context: $0\crash\main
   4: main()
      ^
Context: $0\crash\main
SrcDbg: dil crashfunc
  12: crashfunc()
      ^
crashfunc+0x8       >203CFFFFFFBA     move.l #-70,d0
crashfunc+0xe       >6100F4B4         bsr.w _stkchec
crashfunc+0x12      >5D8F             subq.l #6,a7
  14:   unsigned char *crashaddr = (unsigned char *) CRASHADDR;
                                    ^
crashfunc+0x14      >2F7CCECECECE0002 move.l #-825307442,2(a7)
  17:   result = *crashaddr;
        ^
crashfunc+0x1c      >206F0002         movea.l 2(a7),a0
crashfunc+0x20      >1F500001         move.b (a0),1(a7)
```

All modules required to use the post-mortem debugger have been added to the OS-9 International code disk that is available as EFFO PD disk #121 or in electronic form from <os9int@eltec.de>.

Carsten Emde can be reached by email at <carsten@effo.ch>.

µEMACS for OS-9

Hubert Nehring

Introduction

The *µEMACS* editor is available for almost every operating system and hardware platform. Even OS-9 comes with it. However, the officially released OS-9 *µEMACS* is a very out-of-date version called *umacs*. Nevertheless, almost every OS-9 user is familiar with the basic concepts and usage of this full-screen editor. This article deals with a new port of version 3.12 of *µEMACS* to OS-9, which is now available as EFFO disk PD #126. Throughout this article, the term *µEMACS* refers to this specific port and not to the *µEMACS* in general. The following article contains details and features of the new implementation and gives some ideas and hints on customising *µEMACS* to special needs.

Details of this Port

The aim of this port was to allow *µEMACS* to take advantage of as many OS-9 specific features as possible by its internal structure. Furthermore, some other changes and additions have been made to give *µEMACS* a somewhat smarter appearance and to let it handle some of the OS-9 specifics.

In detail, the following improvements have been made:

- Keyboard input and clock updates in the mode line are signal-driven unlike the polling scheme used in the original source.
- Some effort was made to accelerate file name completion and the *show-files* command.
- *VIEW* mode is automatically invoked when loading a file without write permission.
- The output from sub processes is read into a µ*EMACS* buffer via named pipes instead of using temporary files on disk.
- The *pipe-command* command (bound to ^*X@* by default) is now affected by a numeric argument. If it is executed with a numeric argument of '1' the standard error path of the executed shell command is also directed into the MicroEMACS buffer 'command'. If executed with a numeric argument of '2' the output and error output of the executed shell command are

inserted into the current buffer at point. This may be useful for reading error messages during compilation.

- The *µEMACS* environment variable *$hardtab* is initialised with the appropriate value from the terminal's device descriptor option field *PD_Tabs*.
- The new environment variable *MEPATH* is introduced to hold a list of directories searched for macro files.
- If no path is given macro files suffixed by *.cmd* are searched for in the following directories:
  - user's *HOME* directory
  - current directory
  - directories pointed to by the *MEPATH* environment variable
  - */dd/LIB/ME*

  In case the search was not successful, the suffix *.cmd* is appended and the macro file is searched again. The trailing *.cmd* may, therefore, be omitted in macro file names.
- There are two start-up files: a system-wide start-up file called *emacsrc* that is searched for in every directory pointed to by the *MEPATH* environment variable or in the */dd/LIB/ME* directory. A second start-up file named *.emacsrc* may be located in the user's *HOME* directory. This procedure facilitates separating system-wide and user-specific customisation.
- The *-?* command line option was added in order to make *µEMACS* compatible to the OS-9 program calling convention. Unknown options are not ignored but cause *µEMACS* to abort with error code 1.

Mouse support and file locking are not yet implemented.

Using *µEMACS*

µEMACS is started by typing *me*, which may be followed by file names to be loaded and command line options. Like all other OS-9 utilities, these options are listed by typing *me -?*:

```
Syntax: me [<opts>] {<file names> [<opts>]}
Function: MicroEMACS editor
Options:
  @<file>      execute <file> instead of the 'emacsrc' and '.emacsrc' files
  -c           the following files can be changed (not VIEW mode)
  -e           execute 'error.cmd' instead of 'emacsrc' and '.emacsrc'
  -g<num>      position the cursor at line <num> of the first file
  -i<var> <value>
```

```
            initialise an EMACS variable <var> with <value>
-k<key>     the following files are loaded using CRYPT mode with <key>
            as the encryption key
-r          set MicroEMACS into 'restricted mode'
-s<string>  search for <string> in the first file
-v          the following files are invoked in VIEW mode
```

After starting *µEMACS*, a function key window is displayed for convenience. It may be switched off and on by pressing the *F5* key. The *F6* key invokes the help system.

µEMACS uses the *termcap* file to adapt itself to the terminal and keyboard properties. It recognises function keys and special keys like the *home* or *page down* key.

If the terminal does not provide function keys, the character sequence *^C 1 ... ^C 0* and *^C ! ... ^C)* can be used instead of *FN1 ... FN10* and *S-FN1 ... S-FN10*, respectively. This is convenient for US keyboard layout only. This setting is defined in the standard start-up file *emacsrc* and may be changed there.

Customisation

The most important feature of *µEMACS* is its versatility. Its behaviour is controlled by various so-called environment variables. These environment variables are entirely different from the shell's environment variables, they are not related to each other. Furthermore, *µEMACS* provides a macro language for easy creation of new commands serving special needs. Customising *µEMACS* can be as simple as setting some environment variables and changing key bindings or may be as complex as writing large systems of macro procedures.

Changing Variables and Key Bindings

The simplest way to customise *µEMACS*' behaviour is to change the settings of its environment variables and key bindings. This is most often done in the start-up files *emacsrc* or *.emacsrc*, respectively.

For example, *µEMACS* features the safe saving method, which means that *µEMACS* does not save a buffer's content immediately, but stores it in a temporary file first, then deletes the original file and renames the temporary file to the original file name. If you do not like this saving method, it may be switched off by adding the following line to the start-up file:

```
set $ssave FALSE
```

Other environment variables which may be considered regarding customisation are for example *$timeflag*, *$posflag*, *$disphigh*, and *$hscroll* controlling the appearance of time and cursor position in the mode line, the way characters with codes >127 are displayed and how horizontal

scrolling is done, respectively. The help system contains a complete list of all available environment variables and their current values. In addition, such a list is generated on-line by executing the *M-x describe-variables* command.

µEMACS functions and macros may be bound to or unbound from key sequences or function keys. The command *describe-bindings* creates a list of all current key bindings. These bindings are globally active, i. e. they are valid for all buffers. If, for example, the *set-mark* command shall be bound to the *Shift-F2* (or *F12*) function key, the following line must be added to the start-up file:

```
bind-to-key set-mark S-FN2
```

Unbinding this command from a key is easily done by

```
unbind-key S-FN2
```

If you wish *M-?* to invoke the new help system instead of loading the old-style and out-of-date *emacs.hlp* file, add the following line to the start-up file:

```
macro-to-key get-help M-?
```

The macro *get-help* is defined in the standard start-up file *emacsrc*.

Macros

µEMACS comes with a built-in macro language providing named procedures, control structures, user variables and functions. To keep things simple for the internal interpreter it uses a somewhat odd-looking syntax, especially the prefix notation looks very unfamiliar at first glance. Nevertheless, it provides a powerful and versatile way to extend *µEMACS*, which is certainly worth getting familiar with. Although this article cannot cover all the details of macro programming, it is intended to give some hints and stimulating examples.

Generally, it is a good idea to have a look at the provided macro files and the standard start-up file *emacsrc* to get some ideas about the underlying programming principles. In addition, the help system provides some documentation on functions, directives, etc.

An interesting section for customising is the macro found in *emacsrc*, which sets the default mode depending on the filename.

```
store-procedure set-default-mode
    !if &gre &len $cfname 1          ; if the file name length > 1
        set %rctmp &rig $cfname 2    ; get the two rightmost characters
        !if &or &seq %rctmp ".c" &seq %rctmp ".h"
                                     ; are they equal to ".c" or ".h" ?
            add-mode "cmode"         ; yes -> set CMODE
        !endif
    !endif
    !if &gre &len $cfname 3
```

```
        set %rctmp &rig $cfname 4
        !if &or &seq %rctmp ".cpp" &seq %rctmp ".hpp"
            add-mode "cmode"
        !endif
    !endif
    !if &gre &len $cfname 3
        set %rctmp &rig $cfname 4
        !if &or &seq %rctmp ".mss" &seq %rctmp ".txt"
            add-mode "wrap"
        !endif
    !endif
!endm

set $readhook set-default-mode
```

By setting *$readhook* to *set-default-mode*, the above macro is executed during the *find-file* command. For this purpose, the above listing must be included in the standard start-up file *emacsrc*. Hence, *μEMACS* is forced into *CMODE* when reading a file which name ends in *.c*, *.h*, *.cpp*, or *.hpp*. *WRAP* mode is set on reading files ending in *.mss* or *.txt*. This behaviour can easily be extended. The auto-indentation feature of *CMODE* may be convenient for assembler source files. So, if you wish assembler source files to be edited in *CMODE* automatically, the second *!if* statement in the *set-default-mode* procedure may be changed to

```
!if &or &or &seq %rctmp ".c" &seq %rctmp ".h" &seq %rctmp ".a"
```

Change the last *!if* statement, if you want *WRAP* mode to be applied also on files ending in *.doc*:

```
!if &or &or &seq %rctmp ".mss" &seq %rctmp ".txt" &seq %rctmp ".doc"
```

The above script lines may serve as examples how to include other rules to set modes such as *MAGIC* or *OVER* depending on the name of the file being loaded.

Now consider a more complex problem: One of the most frequent experiences in programming are error messages from the compiler. If these messages are redirected or loaded into a buffer, *μEMACS* should be able to analyse them and react in an appropriate way, i. e. load the source file and go to the line containing the error. The following macro is intended as a first step of solving this problem. It can surely be extended to accelerate the whole turnaround cycle of program development in a convenient way.

For error messages created by the *GNU C*-compiler, the following macro exercises this task, if the error messages are loaded into a buffer named *errormsg*. This can be done by either redirecting the standard error path by a command line similar to *make >>errormsg* and subsequently loading the file *errormsg* or by using the pipe-command ^X@ with a numeric argument of 1 or 2 and appropriately renaming the buffer. To enable regular expression search pattern be sure that *MAGIC* mode is set for this buffer.

```
; gotoerror.cmd
;
; Macro to analyse error messages created by GNU C-compiler
```

```
; - the appropriate file is loaded into a MicroEMACS buffer
; - the cursor is set to the line the error is in

store-procedure goto-next-error
    select-buffer "errormsg"        ; make buffer 'errormsg' current
                                    ; search for error message
    !force search-forward "^\([^: ]+\):\([0-9]+\):"
    !if &seq $status TRUE
        delete-other-windows        ; if found arrange windows
        split-current-window
        5 resize-window
        next-window
        !force find-file &group 1   ; load appropriate source file
        !if &seq $status TRUE
           goto-line &group 2       ; and goto line with error
        !endif
    !else
        write-message "No error message"
    !endif
!endm
```

Let us see how it works: First the buffer *errormsg* is made the current buffer. *GCC*'s error messages have the general format starting at the very left margin:

```
<filename>:<line number>:<explanation>
```

In consequence, such an error message line can be searched for using the pattern *^\([^:]+\):\([0-9]+\):*. This pattern translates to a search for the beginning of a line (^) where three conditions are met:

- one character, which is not a blank and not a colon ([^:]+)
- a colon followed by at least one digit ([0-9]+)
- another colon (:)

By grouping the search pattern of the filename and the line number by a pair of parentheses (\(and \)), it is possible to access them later as *&group 1* and *&group 2*, respectively. The *!force* directive forces *µEMACS* not to abort the macro on failure of the search command, which is the default behaviour when a command does not succeed.

If this search is successful, i. e. *!if &seq $status TRUE*, windows are arranged in a nice way, the appropriate file referenced by *&group 1* is loaded into the other window and the cursor is positioned on the line indicated in the error message and referenced by *&group 2*.

In order to enable *µEMACS* to execute the macro listed above, it must be part of a file located at a place where *µEMACS* expects to find macro files (see above). Assuming the file name is *gotoerror.cmd* it can be loaded into *µEMACS* either manually by the command *M-x execute-file gotoerror.cmd* or automatically during start-up by adding *execute-file "gotoerror.cmd"* to a start-up file. To invoke the macro use *M-x execute-procedure goto-next-error*. For easy access, the macro may be bound to a function key or any convenient key sequence as shown previously.

Summary

The version of *μEMACS* described herein is now available. It has been specially adapted in order to fit into the OS-9 environment as best as possible. *μEMACS* is a general purpose tool for creating or changing text files and is even capable of serving as an environment for program development as briefly shown. The author is willing to keep this EFFO PD up-to-date.

Acknowledgement

I would like to thank Martin T. Whitaker for stimulating and helpful discussions.

Hubert Nehring can be reached via email at <nehring@uni-duesseldorf.de>

Pascal-to-C Converter

Carsten Emde

Introduction

The Pascal language has a particular importance for the OS-9 operating system. First, Omegasoft Pascal helped many people to upgrade from Flex to OS-9, since this Pascal implementation allowed to produce code that could easily be shared between these two very different operating systems. Second, on the original MC6809-based OS-9, the Omegasoft Pascal compiler was probably more frequently used than the C compiler, because both the compiler and the final code had better performance. Even in the early days of OS-9/68000, Omegasoft Pascal played an important role, because it allowed to re-use 6809-programs without requiring major modifications.

It is only today that the Pascal language is less important for the OS-9 world, mainly because there were too many different dialects and ANSI C has achieved a much better level of portability. On the other hand, the two languages Pascal and C probably have much more in common than they have differences, and it may not take more than a week or two for a Pascal programmer to learn writing C programs. This change is greatly facilitated, if he or she can translate the existing Pascal programs to C and continue maintaining the sources on the C side. The Pascal-to-C converter *p2c* serves for this purpose.

History

The Pascal-to-C converter has been written and made available under the GNU general license by Dave Gillespie. He also wrote a complete users' manual that was used as a basis for this article. The current version of the Pascal-to-C converter is 1.20.1.

Implementation under OS-9

In principle, the *p2c* sources just have been re-compiled under OS-9. The only modification made to the sources relates to language extensions Omegasoft Pascal has made for constants. In addition to 8-bit and 16-bit decimal and hexadecimal (preceded by '$') constants, Omegasoft Pascal allows to define binary and 32-bit constants. Binary representation is assumed when a

constant begins with the '%' symbol (same as in OS-9 assembly language) such as `%01101001` and a 32-bit constant is assumed, if it ends with an upper-case 'L' such as `$1234AB57L`. All size specifiers (leading '#' symbol, trailing 'L') may be combined with any of the representation specifiers (leading '%' or '$' symbol).

Using the *p2c* Converter

The *p2c* converter is invoked from the command line using the syntax

```
p2c [ options ] [ file [ module ] ]
```

The input consists of a set of source files; output is a set of *.c* and *.h* files that comprise an equivalent program in any of several dialects of C. Output code may be kept machine- and dialect-independent, or it may be targeted to a specific machine and compiler. Most reasonable Pascal programs are converted into fully functional C, which will compile and run with no further modifications, although *p2c* sometimes chooses to generate readable code at the expense of absolute generality. *P2c* endeavours to insert notes and warning messages into the output code to point out areas which may require human intervention. Output code is arranged to be readable and efficient, and to make use of C idioms wherever possible. The main goal of the translation is to produce C files, which are pleasant and "natural" enough to be acceptable as the new source files for a program. In a pinch, *p2c* will also serve as an ad-hoc Pascal compiler.

Code generated by *p2c* normally does not assume that characters are signed or unsigned. Also, it assumes that *int* is the same as either *short* or *long* but does not depend on which. However, if *int* is not the same as *long*, it is best to use a modern C compiler, which supports prototypes. Generated code does not require an ANSI compatible compiler (unless ANSI-style code is requested), but it does use various ANSI-standard library routines.

All generated code includes the file *<p2c/p2c.h>*, which in turn includes *<stdio.h>* and various other common resources. Also, many translated programs will need to be linked with the runtime library, typically `-l=libp2c.l`.

Given a file name, *p2c* reads from the specified file and outputs to a file with a *.c* suffix added or substituted. For example,

```
p2c myfile.pas
```

reads from *myfile.pas* to produce the file *myfile.c*. The input file may contain a Pascal main program or a single Pascal module (or "unit" in Turbo and UCSD Pascal nomenclature), or it may just contain a number of procedures and declarations. *P2c* is designed to work for correct input programs. That is, it will accept partial programs but may occasionally produce bus errors if the input refers to undefined symbols.

If the input is a module, the translator will also produce a file *module.h* containing a translation of the module's interface section. The implementation section may be omitted in which case only the *.h* file will be of interest. If the program or module has include files, these may cause additional *.c* files to be generated depending on the value of the *ExpandIncludes* option (see below).

If no file name is given, *p2c* reads Pascal from the standard input and writes the resulting C source code to standard output (though a *.h* file may still be produced). If file name and module name are given, the file may include several modules (or units). The specified module is translated; any others are skipped. The output files will be named *module.c* and *module.h*. *P2c* never translates more than one module per run.

Before starting, *p2c* reads the file */dd/lib/p2c/p2crc* for a number of configuration parameters. (The actual path used on a particular system may vary. The *-i* option is a handy way to examine this file.) If the *P2CRC* environment variable is set, it designates the name of a file to read instead of the system file; this file can start with *Include %H/p2crc* to include the system file. Next, *p2c* attempts to read the file *p2crc* in the current directory for further configuration. If this file does not exist, *p2c* looks for *.p2crc* instead.

Options

-o cfile Use *cfile* in place of *file.c* or *module.c* as the primary output file. A single dash ('-o -') says to write the C code to standard output.

-h hfile Use *hfile* in place of *module.h* as the output file for interface text. This only has effect, if the input is an HP Pascal module or a Turbo Pascal unit.

-s sfile Read interface text from *sfile* before beginning the translation. This file typically contains one or more modules, often with interface sections omitted for speed, which the program or module being translated will use. (Typically the *ImportFrom* and *ImportDir* parameters in *p2crc* are set up to allow *p2c* to locate interface text without needing any -s options.)

-pn Display progress of translation in the form of a line number/file name display. This is refreshed every n lines, 25 by default.

-c rcfile Read local configuration commands from *rcfile* instead of *p2crc* or *.p2crc*. A dash ('-c -') in place of *rcfile* causes no local configuration file to be used.

-v ("vanilla") Do not read from the system configuration file /dd/lib/*p2c/p2crc*. Since some of the parameters in this file are required, your local configuration file must include those parameters instead. This also suppresses the file named by the *P2CRC* environment variable.

-H homedir Use *homedir* instead of /dd/lib/*p2c* as the *p2c* home directory. The system *p2crc* file will be searched for in this directory.

-I pattern Add *pattern* to the *ImportDir* search list of places to find modules which are imported. The pattern should include a *%s* to represent the module name, and should evaluate to a potential file name for that module's source code.

-i This special option, which must be the only argument on the command line if used, simply copies the system configuration file */dd/lib/p2c/p2crc* to standard output in its entirety. It may be used with *-H*, but *-i* is most useful precisely when the location of the home directory is not known.

-q Quiet mode. Suppresses output of status messages during translation.

-En Abort translation after n errors. If n is omitted, it defaults to zero, which means that an unlimited number of errors are allowed. Use *-E1* to make *p2c* halt after the first error.

-e Echo the Pascal source into the output file, surrounded by *#ifdefs*. This is the same as the *CopySource* parameter in the *p2crc* file.

-a Produce ANSI C. This is a convenient override for the *AnsiC* parameter in the *p2crc* file.

-L language Select input language name, such as VAX or TURBO. This is a convenient override for the *Language* parameter.

-V Verbose mode. This causes *p2c* to generate an additional *.log* file with further details of the translation, such as a list of warnings and notes including those which are suppressed in the regular output.

-M0 Disable memory conservation. This prevents *p2c* from freeing various data structures after translating each function, in case this new conservation feature causes unforeseen problems.

-R Regression testing mode Formats notes and warning messages in a way that makes it easier to run *diff* on the output of *p2c*.

P2c also understands a few debugging options, which may occasionally be useful when tracking down translation problems. The *-dn* option sets the debug level to n, a small integer, which is normally zero. Debugging output is written into the regular output file along with the C code; the larger n is chosen, the more "wallpaper" is produced. Also, *-t* prints debugging information at every Pascal token, *-Bn* enables line breaker debugging, and *-Cn* enables comment placement debugging.

Choice of Source Language

The Language configuration parameter or *-L* command-line option tells *p2c*, which Pascal dialect to expect in the input file. Any language features, which do not overlap between dialects, are supported all of the time. The Language parameter is consulted when a syntax or usage is detected that has different meanings in two different dialects, and also to determine default values for various other translation parameters as described below.

The following language words are supported by *p2c*. Names are case-insensitive.

Pascal Dialects

HP This is the default language. Most features of HP Standard Pascal, the Pascal Workstation version, are supported. Some features of MODCAL, HP's extended Pascal, are also supported. This is a superset of ISO standard Pascal, including conformant arrays and procedural parameters.

HP-UX Almost identical to the "HP" dialect.

Turbo Turbo Pascal 5.0 for the IBM PC. Few conflicts with HP Pascal, so the *Language* parameter is not often needed for Turbo. Most important is that the Turbo and HP dialects use 16 and 32 bit integers, respectively.

UCSD UCSD Pascal. Similar to Turbo in many aspects.

MPW Macintosh Programmer's Workshop Pascal 2.0. Should also work well for Lightspeed Pascal. Object Pascal features are not supported, nor is the fact that *char* variables are sometimes stored in 16 bits.

VAX VAX/VMS Pascal version 3.5. Most but not all language features supported. This has not yet been tested on large programs.

Oregon Oregon Software Pascal/2. All features implemented.

Berk Berkeley Pascal with Sun extensions.

Modula-2

Based on Wirth [1]. Proper setting of the *Language* parameter is not optional. Translation will be incomplete in most cases, but should be good enough to work with. Structure of local submodules is essentially ignored; like-named identifiers may be confused. Type *WORD* is translated as an integer, but type *ADDRESS* is translated as *char ** or *void **; this may cause inconsistencies in the output code.

Modula-2 modules have two parts in separate files. Suppose these are called *foo.def* (definition part) and *foo.mod* (implementation part) for module *foo*. Then a pattern like *%s.def* must be included in the *ImportDir* list, and *LibraryFile* must be changed to refer to *system.m2* instead of *system.imp*. To translate the definition part, give the command

```
p2c foo.def
```

to translate the definition part into files *foo.h* and *foo.c*; the latter will usually be empty. The command

```
p2c -s foo.def foo.mod
```

will translate the implementation part into file *foo.c*.

Even if all language features are supported for a dialect, some predefined functions may be omitted. In these cases, the function call will be translated literally into C with a warning. Some hand modification may be required.

Configuration Parameters

P2c is highly configurable. The defaults are suitable for most applications, but customising these parameters will help to get the best possible translation. Since the output of *p2c* is intended to be used as human maintainable source code, there are many parameters for describ-

ing the coding style and preferred conventions. Others give hints about the program that help *p2c* to generate more correct, efficient, or readable code.

The *p2crc* files contain a list of parameters, one per line. The system configuration file, which may be viewed using the *-i* option to *p2c*, serves as an example of the proper format. The available configuration parameters allow to define *p2c*'s behaviour in the following categories: general, input language, target language, target machine, braces and placement of statements, indentation, line breaking, comments and blank lines, special comments, stylistic options, coding options, naming conventions, target library and checking.

General

The keywords *Debug* and *TokenTrace* enable debugging at a defined level and additional debugging output, respectively. The keyword *Include* allows for the definition of additional Pascal header files to be included.

Input Language

The most important keyword of this category is *Language* that has already been explained above. Other keywords more specifically define the interpretation of the input language, e.g. whether Pascal comments may be nested, what size is used in C floating-point variables and how non-alpha characters in Pascal identifiers are handled.

Target Language

The *AnsiC* keyword defines the C dialect, i.e. whether K&R, ANSI, or ANSI with GNU C extension is used. Furthermore, structure assignment, casting and the type of *anypointers* can be declared.

Target Machine

The following target machines can be defined to tailor the output program to a particular architecture: HPUX-300, SUN-68K, BSD-VAX, BSD, SYSV. In addition, the bit sizes of the various C types, the representation of bit fields and the handling of signed and unsigned numbers can be set individually.

Braces and Placement of Statements, Indentation, Line Breaks, Comments and Blank Lines

Keywords of these four categories allow to format the C program listing; a wide variety of format features including a sophisticated line breaker exist so that virtually every personal style can be realised.

Stylistic Options

These options define the style of the C program such as redundant parentheses or the usage of spaces in functions. In addition, the syntax for true/false comparisons (*if (x)* vs. *if (x != 0)*) the return syntax (*return(x);* vs. *return x;*) and the pointer syntax (e.g. **cp* vs. *cp[0]*) etc. can be selected.

Coding Options

This category contains more than 70 keywords; they control, for example, whether global variables and functions are declared as static or not, whether strength reduction takes place (e.g. *x & 15* instead of *x % 16* or *x >> 2* instead of *x / 4*) and whether character constant comparisons are replaced by functions (e.g. *isspace(c)* instead of *c==' '*).

Naming Conventions

A variety of keywords define names for output files such as *CodeFileName*, *ModuleFileName* and *HeaderFileName* for C language, module and header file, respectively. Furthermore, prefix and suffix definitions can be made for hidden and other variables that are produced during the translation procedure. Last not least, the *p2c* translator can build internal lists of synonymous and reserved words so that name clashes with C library functions can be avoided.

Target Library

More than 100 keywords of this category define all issues that are related to using libraries in the C environment. It is, for example, possible to force quotes or brackets in the *#include* statement, to define values for TRUE and FALSE and to specify numbers for typical I/O errors as returned by the operating system.

Checking

Finally, special checks can be enabled that help to increase the quality of the produced code, even if that particular feature is not part of the original Pascal source. If the keyword *MallocCheck*, for example, is set to "1", *p2c* will check, if *malloc* returns NULL. Other checks relate to errors that may occur during file I/O.

Reference

[1] Wirth N (1982) *Programming in Modula-2*; edition 2. Springer, Berlin, Heidelberg, New York.

The Pascal-to-C converter is available as EFFO PD disk #120; printed manuals can be ordered on a per-page basis (see EFFO order form).

Carsten Emde can be reached by email at <carsten@effo.ch>.

Upgrading *cpucache* for MC68060

Carsten Emde

Summary

In a previous article [1], a program called *cpucache* was presented that allows to display and manipulate the status of MC68030 and MC68040 on-chip caches. In the meantime, the MC68060 processor is – finally – born. One of the important differences between the MC68040 and the MC68060 is the cache management; therefore, the *cpucache* program required major upgrade changes in order to be used on MC68060-based systems. The new version has been added to the OS-9 International code disk that is available as EFFO PD disk #121.

The Cache Control Register

In contrast to the MC68040 processor that only used bit 15 and bit 31 of the 32-bit cache control register *CACR*, the MC68060 processor makes rather exhaustive use of it and added a variety of new bit definitions to the *CACR*. The following bits are now defined

Bit	**Processors**	**Function** (enabled, if set)
13	68060	Half cache operation mode (instruction cache)
14	68060	No allocate mode (instruction cache)
15	68040/60	Instruction cache
21	68060	Clear all user entries in the branch cache
22	68060	Clear all entries in the branch cache
23	68060	Branch prediction cache
27	68060	Half cache operation mode (data cache)
28	68060	Cache remains valid after *cpush* instruction
29	68060	Four-entry FIFO store buffer
30	68060	No allocate mode (data cache)
31	68040/60	Data cache

Evaluation of bit 23 and 29 ("branch prediction cache" and "four-entry FIFO store buffer") has been added to the *cpucache* program. The most important settings of the MC68060's cache control register can now be inspected and modified. The *-b* option that was used to enable or disable both data and instruction caches of the MC68040, now restores all cache settings to the default values. The meaning of the *-d* and the *-i* option remained unchanged, i.e. they only affect data and instruction cache, respectively.

The Transparent Translation Registers

The definitions of the transparent translation registers *ITT0*, *DTT0*, *ITT1* and *DTT1* of the MC68060 are identical to the MC68040 processor. The *-t* option that allows to decode their settings, therefore, behaves the same way as when running on an MC68040-based system [1].

The Processor Configuration Register

The newly added *-p* option enables display of the chip's revision number, the FPU switch and the setting of the superscalar dispatch bit. This information is available in the so-called processor configuration register (abbreviated *PCR*, not to be confused with the program counter, that should now always be abbreviated as *PC*). The following definitions apply to the *PCR* register that may only be accessed in supervisor mode using the *movec* instruction:

Bit	**Description**
0	Enable superscalar dispatch, if set
1	Disable FPU (simulate MC68EC/LC060), if set
7	Enable debug features, if set
8-15	8-bit device revision number
16-31	16-bit processor type identification number 0x0430 for MC68060, 0x0431 for MC68EC/LC060

Other Changes

OS-9 V3.0 no longer supports the *CP2_DDIO* bit in the compatibility word of the *init* module that allowed to disable data cacheing during I/O operations. In consequence, this bit is not evaluated, if the *cpucache* program is running on an OS-9 V3.0 system. In addition to the initial program version, this release of the *cpucache* program also allows to inspect and to modify the setting of the MC68020's *CACR*.

Example Program Output on an MC68060 System

```
<thle27/root/sh>/dd:cpucache -tp
This 68060 (OS-9 V3.0.2) system cannot disable data cache during I/O.

Cache control register:
   State       Value     Data cache  Instr. cache  Branch cache  Store buffer
   No I/O   0xA0808000    enabled       enabled       enabled       enabled
      I/O   0xA0808000    enabled       enabled       enabled       enabled
   Global   0xA0808000    enabled       enabled       enabled       enabled
   Depth                     0             0

Transparent translation registers:
  Register  Base   Mask  Enable Super U1 U0       Cache mode       Read/Write
    DTT0    0x00   0x01    1    super  0  0  cache/write-through  read/write
    ITT0    0x00   0x01    1    super  0  0  cache/write-through  read/write
    DTT1    0x00   0xFF    1    super  0  0  no cache/serialized  read/write
    ITT1    0x00   0xFF    1    super  0  0  cache/write-through  read/write

Processor configuration register:
   Revision number:      1
   Floating-point unit:  enabled
   Superscalar dispatch: enabled
```

The above setting of all data transparent translation registers to write-through cache mode instead of the better performing copy-back cache mode is, unfortunately, necessary for the time being, since the current releases of OS-9 network software would not run otherwise.

Conclusion

The *cpucache* program is now available for the MC68060 processor. Every serious MC68060 user should have this program handy, especially if the processor does not behave as expected.

References

[1] Emde C (1994) *On-chip caches on Motorola Processors.* OS-9 International 3(3): 23-40.

[2] MC68060 MC68LC060 MC68EC060 Microprocessors User's Manual, Motorola Inc.

Carsten Emde can be reached by email at <carsten@effo.ch>.

Letter to the Editor

Big Hard Disks Under OS-9

OS-9 International 1/95, p. 21

Based on the recommendation given in this article, I created the required descriptors to access two partitions of a big SCSI hard disk. The procedure to create such descriptors is all but simple, and it would be good to have an appropriate tool for this purpose. In short, the first partition must be formatted without the *CNTL_AUTOSIZE* bit set, and the descriptor must contain appropriate information about the number of cylinders, tracks etc. to create a partition of the desired capacity. Next, the first three byte of this hard disk partition (total sectors on disk) must be written to the logical sector offset of the second descriptor (offset 0x69, 0x6a and 0x6b) in addition to, again, appropriate values for the disk geometry. Thereafter, the second partition can be formatted. This procedure only works, if the port nibble (least significant four bits at offset 0x33) of the second descriptor contains another value than the first descriptor (e.g. bit 3 set). BTW: Microware's original drivers only mask the port nibble so that no more than 16 logical devices can be created on the 8 SCSI addresses.

So far, so good. Having created the two descriptors and formatted the disk, the two partitions could be accessed sequentially without any problem. If, however, an attempt was made to simultaneously access the two partitions, e.g.

```
dcheck /h0 >/r0/dcheck0 & dcheck /h1 >/r0/dcheck1 &
```

error 175 (hardware damage) occurred. This problem was only solved by the hardware manufacturer who sent a fixed version of the low-level SCSI driver. According to the hardware manufacturer, a globally defined time-out constant was not large enough. This problem has been forwarded to Microware, since the inappropriate value is part of the OS-9 PortPak. The fixed driver version also solved the above mentioned limitation: it masks not only the port nibble but the entire byte so that up to 256 different logical devices can now be created.

Hans Tietz, <100115.66@compuserve.com>

OS-9 Conference Announcement

Reto Peter

Introduction

The European Forum For OS-9 (EFFO) is organising a conference dedicated to the OS-9 operating system. This conference has already been announced in the last issue of OS-9 International and is targeted for system software developers, industrial and research programmers, system integrators, support engineers and everybody interested in the OS-9 real-time operating system.

Conference Program

The conference is devoted to two main topics that are:

- Self-hosted *vs.* cross software development for OS-9
- Network connectivity of OS-9

These topics will be covered by invited speakers who are known for their expertise in the particular field. The presentations will mostly be given in German.

Call for Papers

If you or your company recently developed a solution for a problem that might be of common interest and you are willing to share your experience, please submit an application to contribute to the free paper session. Such an application should consist of a concise title and a description of the talk (approximately 100 to 200 words). At the most, 16 presentations of 30 minutes each (20 minutes talk, 10 minutes discussion) are scheduled.

Deadline for submitting a proposal for a presentation is **Thursday, February 29, 1996**. Please use the attached form and submit it as described below.

Organisation

The conference takes place

from Friday, September 20 until Sunday, September 22, 1996

at the **Hotel Seeblick** in Emmetten in Switzerland.

The conference registration fee is SFr. 390.– including hotel accommodation (single bed room) and full board. EFFO members receive a rebate of 10%.

If you plan to participate, we kindly ask you to complete the attached form and send it by mail or fax to **EFFO, CH-8606 Greifensee, Switzerland, +41 1 940 38 90**. Additional forms can be obtained from EFFO. You can also register via email at the address <conference@effo.ch>. Please register until **Thursday, February 29, 1996**. A confirmation and the conference documents will be sent to you in May 1996.

Reto Peter can be reached at <reto@effo.ch>.

Reto Peter

Monthly EFFO Meetings and AGM 1996

The monthly EFFO meeting takes place each first Friday of a month. The next meetings will take place in either one of two locations. One is the already known Restaurant "Zunfthaus am Neumarkt" , the other is the Restaurant "Palmhof", both situated in the city of Zurich. On March 1 and April 12 we meet at the Zunfthaus, on May 3 and June 7 at the Restaurant Palmhof. The address of the latter is Universitätsstr. 23. As usual, the meetings start at 8 PM, but most participants meet at 7 PM in the Restaurant to have supper together. Everybody interested in OS-9 is kindly invited to join the meeting. The meeting place for the following meetings from July onwards will be published in the next issue of OS-9 International.

EFFO's AGM 1996 will take place in the Restaurant "Herberge" in Teufenthal, Saturday, March 9, 1996. It starts at 14:30. EFFO members will get a written invitation soon.

Imprint

OS-9 International

Published by	European Forum For OS-9 (EFFO)
President	Werner Stehling
Vice President	Reto Peter
Director of Finance	Stephan Paschedag
Editor-in-Chief	Carsten Emde
Design	Marc Balmer, Werner Stehling (layout)

Address
European Forum For OS-9
P.O. Box
8606 Greifensee
Switzerland

FAX +41 1 940 38 90
email os9int@effo.ch

Printed directly from disk by Fotoplast, Zurich, Switzerland
ISSN: 1019-6714

Subscriptions

OS-9 International is the official organ of the European Forum For OS-9 (EFFO). The subscription is included with the annual EFFO membership fee. In addition, it is available by separate subscription for non-EFFO members, single issues are also available. All following prices are given in Swiss Francs, shipping included:

	Switzerland	*Europe*	*Overseas*
One year (3 issues)	25.00	30.00	35.00
Single issue	10.00	12.00	14.00

To subscribe to OS-9 International or to order a single issue send a letter, postcard, fax or email to EFFO.

Advertisements

OS-9 International is not only an ideal platform for discussing OS-9 related topics, it is also the ideal place to advertise. OS-9 International reaches end-users, system-software developers and, nevertheless, decision-makers.

Please contact EFFO for detailed information on how to place an ad in OS-9 International.