September 1995

OS-9 International

Vol. 4, Issue 3

EFFO OS9

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

Comparable to other operating systems, there are several user groups in the world supporting OS-9 and providing help in form of software collections, periodicals and support via e-mail. However, in contrast to other operating systems, OS-9 user groups usually do not organise meetings or even professional conferences, although such meetings are important and indispensable social events to establish an operating system's users community and to provide the required exchange of know-how and opinions. Examples for successful professional conferences organised by user groups are the increasing number of meetings on the various aspects of the Unix operating system.

Impressed by the success and the importance of such conferences, the European Forum For OS-9 (EFFO) has decided to organise a conference on important topics of OS-9. This meeting will take place in September 1996 – details are given in an article of this issue.

During the last two months, EFFO activists spent a lot of time in organising the conference. In particular, they spoke to a number of industrial and academic OS-9 programmers and system integrators and asked them whether they would be willing to give a lecture at such a conference. In addition, they asked for their opinion, whether such a conference would be successful or not. To their greatest surprise, many invited speakers were enthusiastic about the idea of an OS-9 conference – all of them immediately accepted the invitation to give a lecture, and all of them wholeheartedly encouraged EFFO to organise the meeting. Finally, EFFO activists spoke to a number of potential participants: again, a wide majority expressed their interest in the conference.

Nevertheless, EFFO admits that this conference is an experiment. Will enough people attend the conference? Are the topics well selected? It was, therefore, decided to keep the organisational work to a minimum in making German the conference language. Those among you who are now disappointed may be glad to hear that another conference will be organised in 1998 with English as conference language should the 1996 conference be successful.

EFFO and OS-9 International are looking forward to meeting you at the first EFFO OS-9 conference!

Carsten Emde

Reto Peter

Werner Stehling

Shell Utilities

Carsten Emde

Introduction

The availability of a shell similar to *sh*, *csh* and *bash*, of the GNU *C/C++* compilers and of the GNU *make* has made porting UNIX software to OS-9 much easier than ever before. Many of the EFFO PD programs have been generated using these tools. Unfortunately, some UNIX shell scripts do not only rely on the existence of a compatible shell program, but these scripts often require utility programs such as the expression evaluator *expr*, the file scanner *grep* and the stream editor *sed*. One of these programs *grep* is even part of the OS-9 distribution, but this version does not quite offer all the functionality a normal UNIX shell script would expect from the *grep* program.

Distribution Policy

Initially, it was planned to add the shell utilities to the already existing shell disk (*sh* for OS-9, PD disk #107). The binary programs of the *sh* and the *sh* utilities, however, have different sources: the *sh* for OS-9 is public domain software that may be copied without restriction – the only rule is that it may not be sold and not be part of a commercial product without having obtained the written permission from the author. The shell utilities are GNU software so that GNU's special *copyleft* conditions must be observed. In consequence, this software may only be distributed to others, if all required source material is made available, at least upon request. Because of the different natures of the copying conditions, it was decided to combine the shell utilities on a different PD disk that is now released separately as PD #105. It contains the following shell utilities

Program	**Function**
env	Run a program in a modified environment, display environment
expr	Evaluate expressions
logname	Print user's login name
printf	Format and print data (same as C language function)
su	Change user ID (switch user) or become super-user
whoami	Print name of the current shell's user

and the two programs

Program	**Function**
grep	Print lines matching a selection pattern
sed	Stream editor

In addition, the distribution disk contains the file *copying* with further details about the GNU distribution policy. The current article gives an overview and a general description about the usage of the shell utility programs.

env – Run a Program in a Modified Environment

```
Syntax: env [-] [-i] [-u name] [--ignore-environment] [--unset=name] [--help] [--version]
            [name=value] [command [args...]]
Authors: Richard Mlynarik and David MacKenzie
Version: 1.10 as part of GNU shell utilities
```

The shell utility *env* runs a command with a temporarily modified environment. The modifications are specified by command line arguments of the form

```
variable=value
```

or by command line options. The new value of an argument may be an empty string

```
variable=
```

which is different from unsetting a variable using the *-u* or the *--unset* option, since the Bourne shell may differentiate between empty and unset variables using the *-u* option or the *set -u* command. Considering the shell script *testenv* containing

```
set -u
if test $var
then
  echo 'Variable $var exists and is NOT empty'
else
  echo 'Variable $var exists and is empty'
fi
```

each of the shell command lines

```
$ env -u var sh testenv
$ env var= sh testenv
$ env var=123 sh testenv
```

return a different result

```
Unset variable: var
Variable $var exists and is empty
Variable $var exists and is NOT empty
```

respectively.

The first remaining argument specifies the program to invoke; it is searched for according to the specification of the *PATH* environment variable. Any following arguments are passed as arguments to that program. If no command name is specified following the environment specifications, the resulting environment is printed. This is like specifying the command *printenv*.

If the '-', *-i* or *--ignore-environment* command line option is specified, the program starts with an empty environment, any setting from the inherited environment is ignored as shown in the following two examples:

```
$ env printenv | grep PATH
PATH=.:/dd/CMDS
$ env - printenv | grep PATH
$
```

expr – Evaluate Expressions

```
Syntax: expr <expression> [<expression>]
Author: Mike Parker
Version: 1.10 as part of the GNU shell utilities
```

The shell utility *expr* evaluates the expressions passed as arguments, prints the result to standard output path and returns the logical result as exit status. A value of 0 (TRUE) is returned, if the result is neither NULL nor 0, a value of 1 (FALSE) is returned, if the result is NULL or 0, and a value of 2 is returned in case of an invalid expression. Operands may either be numbers or strings; the latter do not need to be quoted to be recognized by *expr*, but it may be necessary to quote certain strings in order to protect them from being parsed and modified by the shell.

For example, the script

```
value=1
expr $value + 1
```

prints *2* and returns *0*. The result can, of course, directly be assigned to the same or another variable. The construct

```
value=`expr $value + 1`
```

is equivalent to the C expression

```
value++;
```

Assuming the variables

```
value='$value'
value1='$value'
```

the script

```
expr $value = '$value1'
```

prints *0* and returns *1*, while the script

```
expr $value = $value1
```

prints *1* and returns *0*.

The shell utility *expr* evaluates the following syntax elements

arg1 | arg2	*arg1* if it is neither null nor 0, otherwise *arg2*
arg1 & arg2	*arg1* if neither argument is null or 0, otherwise 0
arg1 < arg2	*arg1* is less than *arg2*
arg1 <= arg2	*arg1* is less than or equal to *arg2*
arg1 = arg2	*arg1* is equal to *arg2*
arg1 != arg2	*arg1* is unequal to *arg2*
arg1 >= arg2	*arg1* is greater than or equal to *arg2*
arg1 > arg2	*arg1* is greater than *arg2*
arg1 + arg2	arithmetic sum of *arg1* and *arg2*
arg1 - arg2	arithmetic difference of *arg1* and *arg2*
arg1 * arg2	arithmetic product of *arg1* and *arg2*
arg1 / arg2	arithmetic quotient of *arg1* divided by *arg2*
arg1 % arg2	arithmetic remainder of *arg1* divided by *arg2*
string : regexp	anchored pattern match of the regular expression *regexp* in *string*
match string regexp	same as *string : regexp*
substr string pos length	sub string of *string*, *pos* counted from 1
index string chars	index in *string* where any *chars* is found, or 0
length string	length of *string*
(expression)	value of *expression*

Comparisons are arithmetic if both arguments are numbers, else lexicographic.

grep, ggrep – Print Lines Matching a Pattern

```
Syntax: grep [-[[AB] ]<num>] [-[CEFGVchilnqsvwx]] [-[ef]] <expr> [<files...>]
Authors: Mike Haertel, Arthur David Olson, Richard Stallman, Karl Berry,
         Henry Spencer, Scott Anderson, David MacKenzie, James Woods.
Version: 2.0
```

The utility *grep* searches the named input files or standard input if no files are named, or the file name '-' is given for lines containing a match to the given pattern. By default, *grep* prints the matching lines.

For users wishing to have the original Microware *grep* program simultaneously available with GNU *grep*, the binary program *ggrep* is also part of the distribution disk. This program is entirely identical to GNU *grep* with the only exceptions of different file and module names. There are three major variants of *grep*, controlled by the following options.

-G Interpret pattern as a basic regular expression (see below). This is the default.
-E Interpret pattern as an extended regular expression (see below).
-F Interpret pattern as a list of fixed strings, separated by new lines, any of which is to be matched.

All variants of *grep* understand the following options:

-num Matches will be printed with *num* lines of leading and trailing context. However, *grep* will never print any given line more than once.
-A num Print num lines of trailing context after matching lines.
-B num Print num lines of leading context before matching lines.
-C Equivalent to -2.
-V Print the version number of *grep* to standard error path. This version number should be included in all bug reports (see below).
-b Print the byte offset within the input file before each line of output.
-c Suppress normal output; instead print a count of matching lines for each input file. With the *-v* option (see below), count non-matching lines.
-e pattern Use *pattern* as the pattern; useful to protect patterns beginning with '-'.
-f file Obtain the pattern from file.
-h Suppress the prefixing of filenames on output when multiple files are searched.
-i Ignore case distinctions in both the pattern and the input files.
-L Suppress normal output; instead print the name of each input file from which no output would normally have been printed.
-l Suppress normal output; instead print the name of each input file from which output would normally have been printed.

-n Prefix each line of output with the line number within its input file.
-q Quiet; suppress normal output.
-s Suppress error messages about non-existent or unreadable files.
-v Invert the sense of matching, to select non-matching lines.
-w Select only those lines containing matches that form whole words. The test is that the matching sub string must either be at the beginning of the line, or preceded by a non-word constituent character. Similarly, it must be either at the end of the line or followed by a non-word constituent character. Word-constituent characters are letters, digits, and the underscore.
-x Select only those matches that exactly match the whole line.

Regular Expressions

A regular expression is a pattern that describes a set of strings. Regular expressions are constructed analogously to arithmetic expressions by using various operators to combine smaller expressions.

The utility *grep* understands two different versions of regular expression syntax: basic and extended. In GNU *grep*, there is no difference in available functionality using either syntax. In other implementations, basic regular expressions are less powerful. The following description applies to extended regular expressions; differences for basic regular expressions are summarised afterwards.

The fundamental building blocks are the regular expressions that match a single character. Most characters, including all letters and digits, are regular expressions that match themselves. Any metacharacter with special meaning may be quoted by preceding it with a backslash.

A list of characters enclosed by '[' and ']' matches any single character in that list; if the first character of the list is the caret '^', then it matches any character not in the list. For example, the regular expression *[0123456789]* matches any single digit. A range of ASCII characters may be specified by giving the first and last characters, separated by a hyphen. Finally, certain named classes of characters are predefined. Their names are self explanatory, and they are *[:alnum:]*, *[:alpha:]*, *[:cntrl:]*, *[:digit:]*, *[:graph:]*, *[:lower:]*, *[:print:]*, *[:punct:]*, *[:space:]*, *[:upper:]* and *[:xdigit:]*. For example, *[[:alnum:]]* means *[0-9A-Za-z]*, except that the latter form is dependent upon the ASCII character encoding, whereas the former is portable. Note that the brackets in these class names are part of the symbolic names, and must be included in addition to the brackets delimiting the bracket list. Most metacharacters lose their special meaning inside lists. To include a literal ']' place it first in the list. Similarly, to include a literal '^' place it anywhere but first. Finally, to include a literal '-' place it last.

The period '.' matches any single character. The symbol *w* is a synonym for *[[:alnum:]]* and *W* is a synonym for *[^[:alnum]]*.

The caret '^' and the dollar sign '$' are metacharacters that match the empty string at the beginning and the end of a line, respectively. The symbols \< and \> match the empty string at the beginning and the end of a word, respectively. The symbol *b* matches the empty string at the edge of a word, and *B* matches the empty string provided it is not at the edge of a word.

A regular expression matching a single character may be followed by one of several repetition operators:

?	The preceding item is optional and matched at most once.
*	The preceding item will be matched zero or more times.
+	The preceding item will be matched one or more times.
{n}	The preceding item is matched exactly n times.
{n,}	The preceding item is matched n or more times.
{,m}	The preceding item is optional and is matched at most m times.
{n,m}	The preceding item is matched at least n times, but not more than m times.

Two regular expressions may be concatenated; the resulting regular expression matches any string formed by concatenating two sub strings that respectively match the concatenated sub expressions.

Two regular expressions may be joined by the infix operator '|'; the resulting regular expression matches any string matching either sub expression.

Repetition takes precedence over concatenation, which in turn takes precedence over alternation. A whole sub expression may be enclosed in parentheses to override these precedence rules.

The back reference *n*, where *n* is a single digit, matches the sub string previously matched by the *n-th* parenthesised sub expression of the regular expression.

In basic regular expressions the metacharacters '?', '+', '{', '|', '(', and ')' lose their special meaning; instead use the backslashed versions \?, \+, \{, \|, \(and \).

Diagnostics

Normally, exit status is *0* if matches were found, and *1* if no matches were found. The *-v* option inverts the sense of the exit status. Exit status is *2* if there were syntax errors in the pattern, inaccessible input files, or other system errors.

Bug Reports

Email bug reports to *bug-gnu-utils@prep.ai.mit.edu.* Be sure to include the word *grep* somewhere in the *Subject:* field and the version number (*-V* option).

logname – Print User's Login Name

```
Syntax: logname
Author: unknown
Version: 1.10 as part of the GNU shell utilities
```

This version of logname prints the calling user's name, as found in the file */dd/sys/utmp*, and exits with a status of *0*. If there is no */dd/sys/utmp* entry for the calling process, *logname* prints an error message and exits with a status of *1*. The *logon* surrogate for Microware's *login* may be used to appropriately maintain the *utmp* data base; *logon* was released as part of the TOP software package.
Two command line options are available

--help Print a usage message on standard output and exit successfully and
--version Print version information on standard output, then exit successfully.

printf – Format and Print Data Format

```
Syntax: printf <argument> [<argument>]
Author: David MacKenzie
Version: 1.10 as part of the GNU shell utilities
```

The shell utility *printf* prints its arguments interpreting '%' directives and '\' escapes in the same way as the C *printf* function. The argument is evaluated as many times as necessary to convert all of the given arguments. This *printf* version interprets *\0ooo* as an octal number (*ooo* is 1 to 3 digits) specifying a character to print, and *\xhhh* as a hexadecimal number (*hhh* is 1 to 3 digits) specifying a character to print. It has an additional escape, \c, which causes to produce no further output, and an additional directive, *%b*, which prints its argument string with '\' escapes interpreted the way they are in the string.
When *printf* is invoked with exactly one argument, the following two options are recognized:

--help Print a usage message on standard output and exit successfully.
--version Print version information on standard output then exit successfully.

For example, the shell command line

```
printf %6.9g\\n `expr 1234567 \* 100`
```

returns *123456700* while

```
printf %6.9g\\n `expr 1234567 \* 1000`
```

returns *1.234567e+09*.

sed – Non-Interactive Stream Editor

```
Syntax: sed [-nV] [--quiet] [--silent] [--version] [-e command]
            [-f  script]  [--expression-command]  [--file-script] [file...]
Author: unknown
Version: 2.05
```

Edit commands are entered either from a script file or directly as command line arguments. Several commands can be specified using repeated *-e* options or by separating them with a semicolon from each other. If only a single command is needed, the *-e* option can be omitted. It often is required to include the edit commands in quotes in order to protect them from being parsed by the shell. A script file may contain as many commands as required, either one command per line or, again, separated by semicolons.

An edit command consists of two parts: the address denominator and the function. The address denominator defines, which lines of a text file are to be treated, the function defines the change that is going to be made at the specified line or lines. The address denominator may be omitted in which case the function will be applied to all lines of the text file.

The edit action is performed a single line at a time. The line to be edited is copied into the working memory, and every edit command is applied to that line in the order they are specified in the command line or in the script file. Thereafter, the result is printed to standard output path, if not suppressed explicitly using the *-n* command line option.

In addition to the working memory, there are other intermediate memory buffers available that can be used from the edit functions. In principle, the working memory may also contain several text lines at once.

The address denominator may specify the line to be treated using its absolute line number but it is also possible to use regular expressions (see above). All functions except 'a', 'i', 'q' and '=' accept an address range that can be specified using start and end address separated by a comma. The dollar symbol can be used to specify the last line. If the end address is given in form of a regular expression, the first line matching that expression is taken as end line.

Regular expressions must be included in slashes. The *sed* stream editor uses the same regular expression syntax as *grep* (see above). In addition, the pattern may contain the expression \n that matches the end of a line.

Considering an example input file named *text* containing

```
I am line 1 (AAA)
I am line 2 (BBB)
I am line 3 (CCC)
I am line 4 (DDD)
I am line 5 (EEE)
```

and the function 'd' that deletes all lines in the current working memory, the command

```
$ sed 2d text
```

would delete line number *2* and produce the output

```
I am line 1 (AAA)
I am line 3 (CCC)
I am line 4 (DDD)
I am line 5 (EEE)
```

and the command

```
$ sed 2,4d text
```

would delete lines *2* to *4* and produce the output

```
I am line 1 (AAA)
I am line 5 (EEE)
```

The same result as in the last example could have been obtained by specifying the regular expression *[B-D]* to select only lines that contain one of the characters 'B', 'C' or 'D' to be deleted:

```
$ sed /[B-D]/d text
```

The following functions are implemented:

a
text The string *text* is written to standard output path before reading the next input line.

b label Jumps to the next line labelled *:label* in the script (not in the text) and continues editing with the next edit command after *:label*.

c
text All lines in the current working memory are deleted, and the string *text* is written to standard output path. If an address range is given, the string *text* will be printed once only when the end of the range is reached.

d All lines in the current working memory are deleted, and the next input line is read. Any edit commands that follow this function are skipped, even if they are located in the specified range.

D The first line in the current working memory is deleted, and the next input line is read. Any edit commands that follow this function are skipped, even if they are located in the specified range.

g The entire working memory is replaced by the content of the buffer; the content of the working memory is lost.

G The content of the buffer is appended to the current working memory.

h The content of the current working memory is written to the buffer; the content of the buffer is lost.

H The content of the working memory is appended to the buffer.

i
text The string *text* is immediately inserted to standard output path.

l The content of the working memory is printed to standard output path, non-printable characters are represented as octal numbers.

n The content of the working memory is printed unchanged to standard output path, and the next input line is transferred to the working memory.

N The next input line (including its end-of-line delimiter) is appended to the current content of the working memory. The line number of the current range is incremented.

p The current content of the working memory is printed to standard output path.

P The first line of the current working memory is printed to standard output path.

q The stream editor *sed* exits; no other commands are executed and no other input lines are read.

r file The content of the file *file* is read into the working memory. The current content of the working memory is lost. If the file *file* does not exist, the read command is silently ignored.

s/expression/replacement/[mode] The first occurrence of the text matching the expression *expression* is replaced by *replacement*. Instead of the slashes '/' any other character can be used. The following characters can be used for *mode*:

- **n** A number between 1 and 512 specifies to only replace the n-th occurrence of the specified expression.
- **g** (global) All matching text strings are replaced and not only the first one;
- **p** Whenever a replacement is done, the current content of the working memory is printed to standard output path.
- **w file** Whenever a replacement is done, the current content of the working memory is written to the file *file*.

t label The editor jumps to the label *label*. If *label* is not specified, the editor jumps to the end of the script file.

w file The current content of the working memory is written to the file *file*. If *file* specifies an invalid file name, the write command is silently ignored.

x The contents of the buffer and of the working memory are exchanged.

y/string1/string2/ Every occurrence of a character in *string1* is replaced by the respective character in *string2*. Both *string1* and *string2* must have the same length.

! function The function *function* is executed for all lines that do NOT match the specified range.

: label A label for the functions 'b' and 't' is defined.

= Prints the current input line number to standard output path.

Options

-n Only those lines are printed to standard output path that are explicitly specified using the 'p' function.

-V The program's version and a short help text are displayed.
-e string The string *string* is used as edit command, may occur repeatedly.
-f file The edit command from file *file* are executed.

The stream editor *sed* represents a very powerful tool that can solve many, even complex editing tasks [2]. The detailed description would exceed the scope of this article.

su – Change User ID or Become Super-User

```
Syntax: su [ - ] [ username [ args ] ]
Author: David MacKenzie
Version: 1.10 as part of the GNU shell utilities
```

The shell utility *su* allows to become another user during a login session. If invoked without a *username*, *su* defaults to becoming the super user. The optional argument '-' may be used to provide an environment similar to that the user would expect, if he or she had logged in directly. Additional arguments may be provided following the *username*, in which case they are supplied to the user's login shell. In particular, an argument of *-c* will cause the next argument to be treated as a command by most command interpreters. The command will be executed under the shell specified by *$SHELL*, or if undefined, by the one specified in the password file */dd/SYS/password*. The user will be prompted for a password, if appropriate. Invalid passwords will produce an error message. All attempts, both valid and invalid, are logged to detect abuses of the system. The current environment is passed to the new shell. The value of *$PATH* is reset to */dd/cmds:.*, if not explicitly specified in the user's *.login* or *.profile* file.

The following command line options are available:

-l, --login	Make the shell a login shell
-c cmd, --command=cmd	Pass the *cmd* to the shell specifying the shell's *-c* option
-f, --fast	Pass *-f* to the shell (for *csh* or *tcsh*)
-m, --preserve-environment	Do not reset environment variables
-p	same as *-m*
-s shell, --shell=shell	Run *shell* if */dd/sys/shells* allows it. This file may contain a list of generally permitted shell programs. If this file does not exist, this security check is bypassed.
--help	Display help text and exit
--version	Output version information and exit

A mere '-' implies *-l*. If *username* is not given as command line argument, the system super user 0.0 is assumed.

whoami – Print User Name

```
Syntax: whoami
Author: Richard Mlynarik
Version: 1.10 as part of the GNU shell utilities
```

The shell utility *whoami* prints the user name associated with the current effective user ID. Like many other tools, it has the two options

--help Print a usage message on standard output and exit successfully and
--version Print version information on standard output then exit successfully.

Port to OS-9

All sources could be compiled under OS-9 without requiring any modification to the code; *gcc* V2.5.8, *gmake* V3.7.1, *sh* V1.8 edition 52 and the current version of the *OS9Lib* were used in all cases. The sources to the shell utilities and to *sed* and *grep* are available on the common GNU file servers, as part of Linux distributions and from EFFO upon request.

References

[1] The programs' manual pages.

[2] Dougherty D (1991) sed & awk, edition 1, O'Reilly & Associates, Inc., Sebastopol, CA, 397.

Carsten Emde can be reached by e-mail at <carsten@effo.ch>.

Re-Entrancy Problems

Stephan Paschedag

Introduction

One major problem of multitasking systems is to control the sharing of limited resources like CPU, memory and I/O devices. In addition, OS-9 and other multitasking systems not only allow the sharing of hardware resources, but also the sharing of program code. As this is one of OS-9's strengths, it often is the preferred operating system for small computers. On the other hand, several restrictions must be observed when writing sharable code. Program code is called re-entrant, if it can be shared without undesired side effects. The aim of this article is to give the necessary background information about re-entrancy and to name some problems often being underestimated or not commonly known.

Tasks and Threads

An example of shared code with no obvious problems is, for instance, an editor that is executed simultaneously by two users. Although the program code resides only once in memory, the data areas of the two users are completely separated. One data area together with the editor's program code represent the simplest form of a task. Besides having distinct data areas, the editor must meet additional requirements to be re-entrant. The most important ones are

- self modifying code is strictly forbidden and
- access to shared data must be protected

which are normally fulfilled by the available OS-9 editors.

Typical examples of shared data under OS-9 are data modules and system globals. Global variables are often shared data. In addition, other hardware related components must also be regarded as shared data, for instance

- sequential I/O devices such as terminals or tape streamers and
- hardware registers such as timers.

OS-9 offers several mechanisms like semaphores, events and signals to protect access to shared data. Re-entrancy at task level, therefore, can easily be achieved by using these system services.

In contrast to the above example, some operating systems allow the execution of several programs inside a single task. These programs are called threads of the task. The global data of the task are shared by all its threads. If, in addition, threads execute the same program code, it is impossible to tell from within the task, which thread is currently running.

In principle, the same re-entrancy problems as discussed above for tasks can also appear at thread level. These problems are not only harder to detect but also much more difficult to solve, because the threads operate on the same common data set. Especially, use of library functions can be problematic, because it is usually not known whether they are re-entrant or not.

Threads under OS-9

OS-9 supports threads in a very limited form. In user state, intercept handlers are the only way to implement threads. The kernel checks at the beginning of each time slice whether a signal is pending for that particular process. If a signal is pending, the intercept handler is called instead of continuing with normal program execution. In the broadest sense, the main program and the intercept handler can be regarded as threads of the task.

Although intercept handlers often play an important role in OS-9 programs, many operations are dangerous to employ. Unfortunately, there is only very little documentation covering this aspect. The following chapters of this article illustrate these limitations and give hints on how to overcome them.

Global Variables

Global variables are the most common source of problems, because they are the only means of communication between the intercept handler and the normal program, further called main thread.

Protecting Global Variables

The following typical example of critical code

```
void foo()
{
    int path = open("file", S_IREAD|S_IWRITE);

    if (path == -1) {
        if (errno == EOS_PNNF) {    /* file does not exist */
            path = creat("file", S_IREAD|S_IWRITE);
        }
        else {    /* file exists, but failed to open */
            exit(errno);
        }
    }
}
```

which does not look suspicious at first sight, may lead to serious problems.

What happens, if the intercept handler modifies the global variable *errno*? If this happens between the *open* call and the test of *errno*, the function will not work properly. It is not possible to detect, whether the file does not exist, or whether the *open* failed for some other reason. Not even protecting the critical area with *sigmask* does help, because it is not known, whether *open* calls the *sleep* function, which would clear the signal mask. Most I/O functions are likely to call *sleep* while waiting for an input or output operation to complete. The obvious solution to this problem is to copy *errno* into a local variable at the beginning of the intercept handler and to restore it at the end:

```
void sighand(int sig)
{
    int old_errno = errno;
    ...
    write(1,buf,bufsize);    /* write may alter errno ! */
    ...
    errno = old_errno;
}
```

Intercept handlers, therefore, must not alter global variables directly or indirectly.

Protecting Operations on Global Variables

The following example illustrates another problem. Both the main thread and the intercept handler access a global doubly-linked list. Insert and delete operations are not protected against a task switch that activates the intercept handler. Accessing the linked list inside the intercept handler can be dangerous, because it is not sure that the main thread has already finished the insert or delete operation and, therefore, the link pointers may only partially be updated.

```
struct list {
    struct list *next;
    struct list *prev;
    Data *element;
};
```

```
/* Insert new element before pos */
void Insert(struct list *pos, Data *data)
{
    struct list *tmp;
    tmp = malloc(sizeof(struct list));
    tmp->element = data;
    tmp->next = pos;
    tmp->prev = pos->prev;
    pos->prev->next = tmp;
    pos->prev = tmp;
}
```

OS-9 offers a mechanism to protect such operations. The *sigmask* function allows the programmer to control whether the kernel is allowed to call the intercept handler or not. The argument passed to the *sigmask(int x)* function increments (x=1), decrements (x=-1) or clears (x=0) a mask in the process descriptor. The kernel will only call the intercept handler, if the mask is 0; otherwise, the signal is queued until the mask becomes 0. The mask cannot be decremented below 0. Calling the *sleep* function implicitly clears the mask.

The insert function can now be protected using the described *sigmask* mechanism:

```
void Insert(struct list *pos, Data *data)
{
    struct list *tmp;
    tmp = malloc(sizeof(struct list));
    tmp->element = data;
    sigmask(1);
    tmp->next = pos;
    tmp->prev = pos->prev;
    pos->prev->next = tmp;
    pos->prev = tmp;
    sigmask(-1);
}
```

The intercept handler needs not to protect such operations, because the kernel implicitly protects the entire code in the intercept handler by incrementing the signal mask before entering the handler and decrementing it at the end.

Library Functions Using Global Variables

Many functions in Microware's standard C libraries use global variables. Unfortunately, this is not documented. In general, it can be said that all library functions accessing global variables are potentially not re-entrant. The documentation only points out that buffered I/O functions may not be used within intercept handlers. For example, unpredictable results, like unordered or duplicated output, are the consequence, if *printf* is called interchangeably from both the main thread and the intercept handler. This is due to the fact that the global data structure *FILE stdout* is used to maintain the global output buffer. Even protecting *printf* with *sigmask* does not solve the problem, because the low-level I/O routines may call *sleep*. A partial solution

is to open a second file inside the intercept handler to the same device. This eliminates the potential duplicate output, but, however, the output is still unsynchronised.

The additional file variable must be local to the intercept handler, because the intercept handler itself must be re-entrant.

```
void sighand(int sig)
{
    FILE *fp = fdopen(_fileno(stdout),"w");     /* duplicate FILE stdout */
    ...
    fprintf(fp,"%d\n",sig);
    ...
    fclose(fp);
}
```

It is, therefore, highly recommended to avoid any I/O function inside intercept handlers at all.

Library functions have to be verified not to share global data structures with functions called from the main thread, before they can be used inside intercept handlers. This verification is not an easy task. The library modules have to be searched for global data using the *rdump* or *libgen* utilities. It is important to note that static variables are also global variables, but they are not marked as such. They are, therefore, not directly visible in the module dump. Static variables can only be detected using the debugger or by comparing the total size of global data with the sum of all sizes of the global non-static variables in the module.

If the library module uses global data, it is difficult to determine whether its functions are safe or not. To answer this question the code has to be reviewed with an assembly-level debugger.

Memory Allocation and Deallocation

Using memory allocation and deallocation functions like *malloc* and *free* inside intercept handlers is very dangerous. This is due to the fact that, in order to improve performance, *malloc* not always uses the system call *F$SRqMem*, which is quite time consuming and which cannot allocate arbitrary sized blocks of memory. Therefore, *malloc* requests large blocks of memory from the operating system and allocates smaller units for the application program. A global doubly linked list is used for this mechanism with the problems described above. If the combination of *malloc* and *free* is used in different threads of a task, the program will sooner or later terminate with a bus error. The fault address is usually $8765ABCD which is Micorware's magic number to mark memory blocks allocated by *malloc*.

For the time being, the system memory request functions must be used instead, if an application really needs to allocate memory in an intercept handler. It is planned that one of the next *gcc* releases for OS-9 will include memory request functions that are safe to be used in threads.

Compiling For Re-Entrancy

Not only library functions can cause problems with global variables but also the user's application itself due to optimising strategies of modern compilers. For example the following code

```
int flag;

void sighand(int sig)
{
    if (sig == MYSIG) {
        flag = TRUE;          /* set flag if MYSIG has been received */
    }
}

int main(int abc, char *argv[])
{
    intercept(sighand);
    ...
    flag = FALSE;
    while (flag == FALSE) {   /* poll the global flag */
        sleep(0);             /* wait for next signal */
    }
    ...
}
```

seems to be correct but it contains a potential problem. Highly developed optimisers of modern compilers assume that variables are not modified without their knowledge. To speed up program execution the compiler is allowed to generate code that temporarily keeps variables in processor registers. In the above example, *flag* can be kept in a register which results in an infinite loop because *sighand* will modify *flag* in memory and leaves the temporary copy unchanged. Fortunately, the ANSI-C standard offers a method to prevent creation of temporary copies. Such variables have to be declared as volatile:

```
volatile int flag;
```

Consequently, all variables modified in intercept handlers should be declared as volatile.

Trap Handlers

Calling trap handler functions inside the intercept routine can result in very confusing effects. Hours can be spent trying to debug such a program, without any results. The method of implementing trap handlers, as shown in the example code distributed with Professional OS-9, is not re-entrant. Microware's implementation of trap handlers has one major disadvantage: a return address is stored in a global variable. Calling a trap handler function goes through the following stages:

```
int a,b,c; /* global */

void test()
{
    foo(a,b,c);
}
```

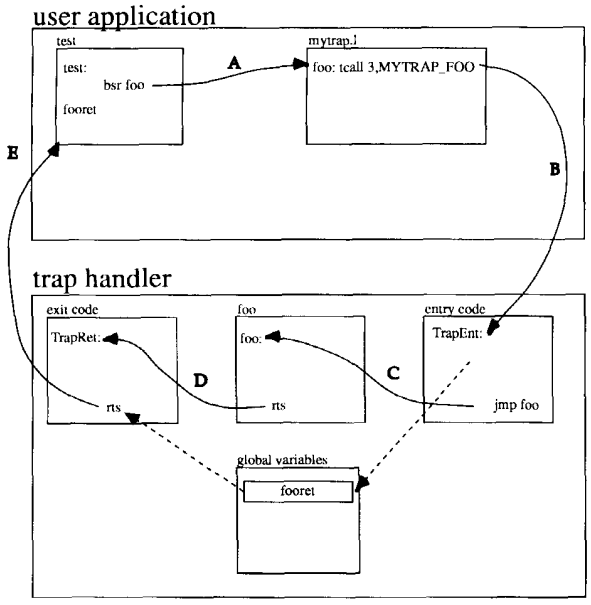
The above source code with a call to the trap handler function *foo* is compiled into:

```
test:   move.l  c(a6),-(a7)
        move.l  b(a6),d1
        move.l  a(a6),d0
        bsr     foo
fooret  rts
        ...

foo:
        tcall   3,MYTRAP_FOO  call trap handler #3 with function code MYTRAP_FOO
```

The entry code of the trap handler needs to set-up the call to *foo*'s implementation inside the trap handler:

```
TrapEnt:
        get function code (MYTRAP_FOO)
        calc pointer to foo's implementation in trap handler
        save address of fooret (return address) in global variable
        replace return address by TrapRet
        clean stack frame
        jump to foo's implementation in trap handler
TrapRet:
        copy errno and other global variables to caller's variables
        return to address saved in global variable
```



There is a good reason for saving the return address E in a global variable. Usually, trap handler functions are written in C like library functions. This implies that the argument passing mechanism is the same in both cases. The stack frame for a call of a library function A must be the same as for the corresponding call inside a trap handler C. The problem arises, because some processing before, *TrapEnt*, and after the actual function, *TrapRet*, in the trap handler is required. This leaves two return addresses to be used by D and E on the stack, which conflicts with the argument passing requirements.

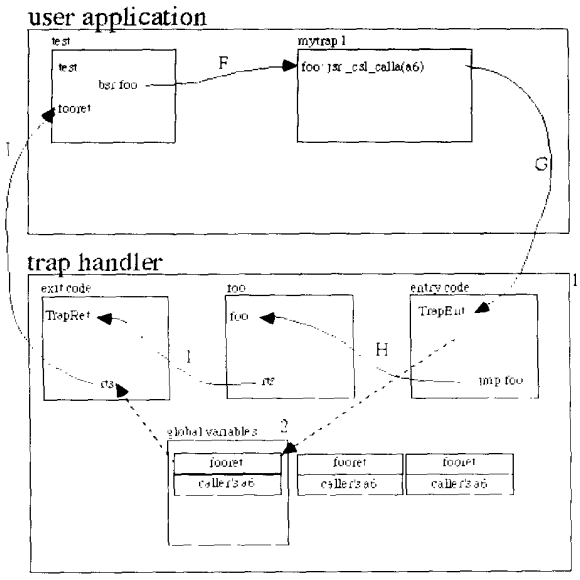
If such a trap handler function is called inside an intercept routine, while the main thread is already running in the same trap

handler, the global variable is overwritten. This results in both routines returning to the same address instead of two individual ones.

It is not easy to find a solution to this problem. One possibility is to have two global variables, one for the return address to the main thread and one to the intercept handler. The trap handler's entry code would then look like:

```
TrapEnt:
        get function code (MYTRAP_FOO)
        calc pointer to foo's implementation in trap handler
        if test-and-set return-var #1 not yet used
            save address of fooret in return-var #1
            replace return address by TrapRet1
        else
            save address of fooret in return-var #2
            replace return address by TrapRet2
        endif
        clean stack frame
        jump to foo's implementation in trap handler
TrapRet1:
        copy errno and other global variables to caller
        return to address saved in return-var #1 and clear it
TrapRet2:
        copy errno and other global variables to caller
        return to address saved in return-var #2 and clear it
```

The above implementation only works, if the intercept handler is not interruptable, that means it does neither call *sigmask(0|-1)* nor *sleep*.



Microware uses a similar approach for its *csl* trap handler distributed with Ultra C V1.2. Instead of calling trap handler functions conventionally, i.e. using the *tcall* macro, they bypass the trap instruction and call the entry code in the trap handler directly G. In order to be able to call the entry code the trap handler's position in memory 1 and its static storage pointer 2 must be known. The trap handler's initialisation routine, which is called with *F$STrap*, returns the pointer to the entry code *TrapEnt* in the trap handler and its static storage pointer. This information is saved in global variables *_csl_calla* and *_csl_a6*, respectively, that are declared in the *cstart.r* library. This is also the place where the trap handler is initialised from. The call to the trap handler now looks as follows:

```
foo:
        jsr     _csl_calla(a6) call the entry code
        dc.w    MYTRAP_FOO
```

The entry code does the following:

```
TrapEnt:
        get function code (MYTRAP_FOO)
        calc pointer to foo's implementation in trap handler
        if already in trap handler
            allocate a new buffer for caller data
        endif
        save caller information in buffer
        set-up trap handler static storage pointer (_csl_a6)
        clean stack frame
        jump to foo's implementation in trap handler
```

This method has two advantages, it is re-entrant and it is much faster than the standard calling method. The way Microware has implemented the *csl* trap handler works only, if the trap handler knows where the main thread has stored the trap handler's static storage pointer *_csl_a6*. The linker always assigns the same offsets to the two variables, because they are declared in *cstart.r*. This is not the case for user supplied trap handlers, where the call to the trap handler needs to pass the static storage pointer to the entry code, for example on the stack:

```
foo:
        move.l  _csl_a6(a6),-a7
        jsr     _csl_calla(a6)
        dc.w    MYTRAP_FOO
```

The entry code must be modified to handle the additional stack parameter.

If user supplied trap handlers are to be used inside intercept handlers, it is necessary to modify them in the way described above, which is much more efficient than protecting all trap-function calls inside the main thread with *sigmask*.

Conclusions

Writing intercept handlers can be quite a tricky task and, therefore, needs a lot of care. As Microware states, intercept handlers should be kept as short and simple as possible. Implementing applications that do some more processing in the intercept handler require sound knowledge of the operating system, the compiler and its libraries. In all cases, the consequences of each statement of the intercept handler must be known. In addition, careful analysis is required, which parts of the main thread have to and can be protected against side effects introduced by the intercept handler. Special attention must be paid to global and static variables. In general, the following recommendations can be given:

What is Safe

- not to use the same global variables in different threads
- to make the intercept handler as short and simple as possible
- just to set a volatile flag and to do the processing in the main thread
- to use the *csl* trap handler distributed with Ultra C V1.2
- to use library functions that do not access global or static variables and that only take constants as arguments (no pointers)

What is Not Safe

- to access or to modify shared global variables unless the access is protected
- to use one of the *malloc* and *free* family functions
- to call an inappropriate trap handler (*cio*, *csl* before Ultra C V1.2)
- to use functions that access shared global variables
- to use functions that access static variables

What to Avoid

- functions of which the source code is not available to the programmer, must be assumed to be unsafe
- functions that might call *sleep*, for example I/O functions

Stephan Paschedag can be reached at <stp@effo.ch>.

First Aid

Werner Stehling

Introduction

The boot procedure of an OS-9 development system equipped with a single hard disk entirely relies on the existence of a valid OS-9 boot file on this hard disk. This boot file normally is called *OS9Boot* and must contain all modules that are required to successfully complete the boot procedure. If this file is damaged for some reason, the system no longer boots. Often the same hard disk is also used for storing the most recent versions of the boot modules so that backup media such as DAT etc. can only be accessed, if the system is running. Careful programmers, therefore, have an additional hard disk connected to the system. Real programmers, however, have not and need to go the painful way to restore their system using the originally distributed floppy disks. This article describes the generation of an OS-9 emergency disk that helps to quickly resuscitate the system, if not equipped with a second hard disk that holds the required files.

The Painful Way

The painful way of repairing the system disk is to boot from the original distribution floppy disks, to load required drivers and descriptors, to reformat the hard disk, if necessary, and to install the latest backup. Although this method is straight-forward, two important points may have been overlooked:

- a sufficiently recent backup is not available, or
- a current backup is, indeed, available but driver and descriptor to access the backup medium are part of the backup and not available elsewhere (the can-opener is in the can ...).

To protect against the second scenario, it is necessary to store the required drivers and descriptors on a separate floppy disk, if they are not part of the original OS-9 distribution disks.

Sometimes it may not be necessary to reformat the hard disk and to restore its contents completely, because only the logical directory structure is corrupt and can be repaired. Most programmers, therefore, have additional utilities for disk repair that, again, may not be part of the OS-9 distribution and must be found in the pile of floppy disks. At least, a lot of disk-jockeying is required.

Finally, the psychological situation at the very moment when the hard disk crashes must be considered, which usually leads to an enormous stress condition. It is, therefore, highly desirable that the recovery procedure can be performed in the normal and well-known environment. This includes the use of the favourite shell, editor and terminal configuration. Recovery is greatly facilitated, if all tools and programs are located on a single floppy disk.

The Concept of an Emergency Boot Disk

The standard OS-9 floppy disk is 3.5", double-sided, double-density and formatted in universal format (sector and track offset = 1). It has a net capacity of about 630 kByte, which is not very much and usually too small to hold all desired files. Therefore, the idea was born to write all files that are not needed for booting to a compressed archive. This archive is expanded during the start-up procedure and its content is written to ram disk so that the floppy drive is no longer occupied.

For example, the proposed emergency floppy disk could have the following directory structure:

```
                        Directory of /d0 11:07:06
BOOTOBJS        CMDS            OS9Boot         disk.tar.gz     startup
tar.gz
                    Directory of /d0/BOOTOBJS 11:07:10
r0.3M           ram
                      Directory of /d0/CMDS 11:07:11
cio             gzip            iniz            load            shell
```

The utility programs are compacted in the file *disk.tar.gz*. The decision to use the combination of *tar* and *gzip* instead of *lha* is based on the observation that *gzip* produces smaller archives by about 25%. Additional space is saved by storing the *tar* program in compressed form, too. This is not possible with *lha*, since it contains both archive program and compressor in one binary file.

The emergency floppy disk is expanded to the ram disk at boot time and generates the following directory structure:

```
                       Directory of /r0 11:11:23
.login          .profile        CMDS            SYS             TMP
                      Directory of /r0/CMDS 11:11:23
BOOTOBJS        attr            cfp             cio             cmp
code            copy            dcheck          deiniz          del
deldir          diff            dir             dsave           dump
echo            format          free            grep            gzip
ident           iniz            less            lha             link
list            load            load_dat        login           math
mdir            me              mfree           mkdir           moded
move            os9gen          patch           pd              printenv
rename          save            sh              shell           sleep
space           sysdbg          tape            tar             tmode
top             tr              tsmon           undel           unlink
xmode
```

```
                    Directory of /r0/CMDS/BOOTOBJS 11:11:24
bootlist.h0.vccs                d0              dd.d0           dd.h0.vccs
dd.r0.3M        h0_fmt.vccs     h3.vccs         init            kernel
mt0_6_sony      nil             null            pipe            pipeman
r0.3M           ram             rbf             rbsccs          rbvccs
rtc62421        sbf             sbsony          sc68562         scf
scsi40          ssm040          syscache040.harry               sysgo
term            tk68230         w0
                            Directory of /r0/SYS 11:11:24
emacs.rc        errmsg          moded.fields    password        termcap
                            Directory of /r0/TMP 11:11:24
```

Comparing the floppy disk capacity of about 630 kByte with the total size of the uncompressed emergency floppy disk

```
space -kd=/d0 /r0
Space requirement of "/r0":

                                  Contents  Contents    KBytes    KBytes   # of
Directory/file name                (files)   (total)      used    (min.)  Files
-------------------               --------  --------   -------   -------  ------
/r0/CMDS/BOOTOBJS                    76971     76971     88.75     88.00      31
/r0/CMDS                            854700    931671    946.25    945.50      55
/r0/SYS                             101662    101662    102.50    101.75       5
/r0/TMP                                  0         0      1.25      0.50       0
/r0                                   1726   1035059   1053.00   1050.75       2

Number of files                    : 93
Number of subdirectories           : 4
Total file contents                : 1010.80 KBytes (1035059 Bytes, 100%)
Space allocated now on /r0         : 1053.00 KBytes (4212 Sects (256), 104%)
Minimum required on /d0            : 1050.75 KBytes (4203 Sects (256), 104%)
```

of about 1050 kByte shows that the floppy disk can hold 65% more data by using the proposed archive technique. Of course, sufficient memory for installing a ram disk of 1.2 MByte must be available.

Using the Emergency Boot Disk

The Boot Procedure

The OS-9 cold start procedure consists of several steps, irrespective of the current boot medium hard disk or floppy disk. First, a short routine that is located in the boot EPROM looks at disk sector 0 and reads the long word at offset $15–$18 that contains the first sector number of the system boot file *OS9Boot* [1]. From OS-9 V2.4 onwards, the long word may contain the address of its file descriptor block. The file *OS9Boot* holds all necessary information and modules to start the OS-9 system. For the time being, *OS9Boot* cannot be a compressed file; thus, it must be as small as possible. A typically configured system contains the file *bootlist* or similar in the */dd/CMDS/BOOTOBJS* directory, where the required modules are listed.

In contrast to a standard hard disk system that normally uses the same default device during an entire OS-9 session, the emergency floppy disk is the default device at the beginning of the boot procedure, but the RAM disk becomes the default device later on. This change cannot be performed easily, since OS-9 defines the default device in the configuration module *init* that is inspected during boot procedure and also at any subsequent call to the *login* program. The following procedure was chosen, since it does not involve changes to system modules such as *sysgo*. It is based on the fact that a module already resident in memory can be replaced by another one with the same name, if the revision number of the new module is higher than that of the existing one.

In detail, the device descriptor */dd* of the *OS9Boot* file is identical to the floppy disk descriptor */d0*. This device is initially used as current data directory and the *CMDS* directory on this device is used as current execution directory. At the end of the boot procedure, another */dd* descriptor is loaded that is identical to the RAM disk descriptor */r0*. This ensures that files such as *password* and *motd* are searched in the *SYS* directory of the RAM disk so that the floppy disk is no longer needed.

The following list exemplifies the contents of the *init* module; the standard utility *moded* can be used for inspection:

```
moded: l
reserved                        :          0
number of irq polling entries   :         32
device table size               :         32
initial process table size      :         64
initial path table size         :         64
startup parameter string        :
first executable module         :      sysgo
default directory name          :        /dd ← this is the above mentioned default device
console terminal name           :      /term
customization module list       : OS9P2 syscache ssm
clock module name               :    tk68230
ticks per time slice            :          2
reserved                        :      $0000
site code                       :          0
installation name               : Force SYS68K/CPU-40
cpu type                        :      68040
operating system level          : $01020403
os-9 revision name              : OS-9/68K V2.4
initial system priority         :        128
minimum priority                :          0
maximum age                     :          0
module directory size (unused)  : $00000080
initial event table size        :          0
compatability flag #1           :        $00
compatibility flag #2           :        $0c
irq stack size (longwords)      :        256
coldstart "chd" retry count     :          0
```

The init module also specifies the name of the program to be forked first, usually *sysgo*. The purpose of the *sysgo* module, as proposed by Microware, is to start a shell with the standard

input path redirected to the *startup* file and then to enter an endless loop, from where repeated shell sessions for the system console are started.

The names *shell* and *startup* are hard-coded in the *sysgo* module. Although they could be replaced, it was decided to keep the original Microware *shell* on the boot floppy, because it requires less space than any other shell.

The *startup* File

The *startup* file of the emergency floppy disk holds the instructions to install a ram disk, to decompress the archive and to switch to the RAM disk as default device.

```
* OS-9 2.4 EMERGENCY BOOT DISK
* ============================

* install a 3-MByte RAM disk
load cio
chd /dd/bootobjs
load -d r0.3m ram
iniz /r0

* expand the archive to RAM disk
chd /dd
gzip -cdfv tar.gz >/r0/tar
gzip -cdfv disk.tar.gz >/r0/disk.tar
chd /r0
load -d tar
tar xvb 90 disk.tar

* change directories and get the uncompressed files from floppy
chx /r0/cmds
chd /dd/cmds
copy -b90 -w=/r0/cmds *
chd ../bootobjs
copy -b90 -w=/r0/cmds/bootobjs *

* re-map the /dd device (mandatory to find SYS/password)
chd /r0/cmds/bootobjs
load -d d0 w0 dd.r0.3m
iniz /d0 /w0 /dd

* get the hard disk stuff
load -d rbvccs h0_fmt.vccs
iniz /h0

* CAUTION: /h0 is format-enabled !!!

* clean up
chd /r0
attr tar -epe
move tar -w=cmds
del disk.tar
mkdir TMP
```

```
load math

* now it's user's turn
* type 'load_dat' to install the DAT driver

tsmon -pd /term
```

The module revision number of *dd.r0*, as already mentioned above, must be higher than that of *dd.d0*, in order to be accepted as new */dd* device. The revision level *M$Revs* is stored at byte offset $15 in the module header. It can be changed in the only correct way by editing the descriptor source accordingly, or – in the usual quick and dirty way – by patching the descriptor with a utility such as *patch*, *beav* or *debug*.

Generating the Emergency Boot Disk

A template directory called *EMERGENCY* can be found on the OS-9 International PD disk. Part of this directory is a shell script called generate that can be used to automatically create the emergency disk. Shell variables hold the list of utilities that should be included in the emergency boot disk. It is not necessary to indicate full path descriptions individually; the shell function *copyfiles* automatically copies the first source file that is found while scanning the path variable *SRC_PATH*. The following listing shows the *sh* shell function *copyfiles* that takes care of the various file copying tasks. The entire script is available on the OS-9 International code disk PD #121.

```
copyfiles() {       ### copy options in $1, list of files in $2 ###
  for i in `echo $2 | tr '\t' '\n' | tr ' ' '\n'`
  do
    p=""
    for d in $SRC_PATH      ### scan all source path directories ###
    do
      if test -r $d/$i      ### file present ? ###
      then
        p=$d/$i
        break
      fi
    done
    if [ $p != "" ]
    then
      echo "copying $1 $p to $CWD/$i"
      copy $1 $p
    else
      echo "<<<<<< not found >>>>>>  $i  <<<<<<"
    fi
  done
}
```

All files designated to be included in the archive are collected on RAM disk in the same directory structure as reconstituted by the emergency boot procedure on RAM disk.

Conclusion

Many people believe that the oddest thing in a programmer's life is a hard disk crash. The truth, however, is that a hard disk crash is only a disaster, if adequate measures are not taken in advance. Two precautions may help to cope with a hard disk crash in a more relaxed way:

- backup as often as possible and
- have an emergency boot disk at hand.

Reference

[1] Microware (1991) OS-9 Technical Manual, edition Rev. J, OS-9 2.4, Microware Systems Corp., Des Moines, Ia.

Werner Stehling can be reached at <stehling@effo.ch>.

OS-9 Conference Announcement

Reto Peter

Introduction

The European Forum For OS-9 (EFFO) was founded with one main goal in mind: to support the operating system OS-9 and its users. Therefore, EFFO has decided to organise a conference that is dedicated to the OS-9 operating system. EFFO invites system software developers, industrial and research programmers, system integrators, support engineers and everybody interested in the OS-9 real-time operating system to attend the conference.

Conference Program

Several EFFO meetings have been devoted to determine current areas of interest in the OS-9 community. In addition, contributions to the *comp.os.os9* and other usenet fora have been monitored. As a result, two main topics have been found to be mentioned most frequently and to be discussed most controversially:

- 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.

In addition, EFFO invites interested participants to submit proposals for presenting a free paper. The content of such a presentation can be the description of a successfully launched OS-9 project, of an unusual but efficient OS-9 strategy or of any other OS-9 solution of general interest. Although it is preferred that these presentations centre on the above-mentioned two main topics, other topics may also be covered, if sufficiently important. A programme committee will decide upon acceptance of a particular submission.

There will be no official conference proceedings but it is intended that OS-9 International will publish an article about the conference. It is, however, the hope that many speakers can be convinced to publish their presentation in OS-9 International.

Conference Language

This first EFFO-organised conference is also intended to be a test whether there is sufficient demand for such an event. Therefore, it was decided to invite only speakers from Switzerland or close neighbourhood. All talks scheduled up to now will be given in German, but submitted contributions may be presented in English, too.

If the conference turns out to be a success and not a total financial disaster for EFFO, it is planned to organise subsequent meetings with English as conference language.

Preliminary List of Invited Talks

The following list has been translated to English, but, as already mentioned above, the talks will be given in German language.

Self-hosted *vs.* cross software development for OS-9	**Speaker**	**Company**
General aspects of program development for OS-9	Carsten Emde	Computer Experts
Self-hosted development under OS-9	Martin Raabe	Eltec
Principles of designing an architecture-independent compiler	Stephan Paschedag	SBG
Resource editors and connectivity of the graphical user interface	Wolfgang Ocker	reccoware
Cross development using FasTrak	Martin Merkel	CERN
Development of PLC-applications	Hans Wiedemann	PEP
Cross development under Unix/Linux	(not yet confirmed)	
Cross development under Macintosh OS	Lukas Zeller	ZEP
Network connectivity of OS-9	**Speaker**	**Company**
Technical aspects of network transport layers	(not yet confirmed)	
OS-9 and the various field busses	Herbert Henze	Schlafhorst
OS/2 and OS-9	Pius Meier	IBM
Macintosh and OS-9	Beat Forster	Spectralab
Unix and OS-9	Helmut Kohl	Eltec
DOS/Windows and OS-9	Lothar Albrecht	Dr. R. Keil
VMS and OS-9	(not yet confirmed)	
OS-9 and OS-9	Werner Stehling	ETH Zurich

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.

Date and Location

The conference takes place

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

at the **Hotel Seeblick** in Emmetten in Switzerland. "High above Lake Lucerne, higher than even the picturesque village of Emmetten, on a natural terrace favoured by the sun, lies the Seeblick Conference and Vacation Centre. Seeblick is surrounded by spectacular mountains and peaceful lakes." Saturday afternoon is reserved to verify the above quotation from the Seeblick's guest information.

Registration

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>.

Letters to the Editor

Debugger Insights

OS-9 International 2/95, p. 35

There is a comment to be made to Carsten Emde's article "Debugger Insights": OS-9 allows to define so-called special memory areas for non-volatile SRAM. OS-9 modules will be searched here after startup and made visible, if header parity and module CRC are correct. Special memory areas can be defined either as part of the Boot ROM's *MemList* or using the coloured memory list of the *init* module. If your OS-9 system contains modules in non-volatile RAM to be debugged, hard breakpoints can be used in fact. But if the system crashes during this phase, modules with hard breakpoints will not be found after restart, because the hard breakpoints have invalidated the modules' CRC.

Beat Forster, Spectralab Kilchberg, <beat@effo.ch>

This article states that the source level debugger *srcdbg* always uses hard breakpoints. This is not entirely correct, since hard breakpoints are only the default setting, but can be disabled. The option command 'o' can be specified with the argument *rom* to toggle between hard and soft breakpoints. The same feature is available in the option menue of FasTrak's source level debugger that is derived from *srcdbg*.

Anonymous

_getsys();

Reto Peter

OS-9 International Code Disk

The OS-9 International code disk contains example programs, scripts and related documents to articles that appeared in OS-9 International. The code disk is available

- as EFFO PD disk #121,
- via email server: mail to *os9int@eltec.de* with *help* as subject,
- via the mailbox of the Computer Verein Frankfurt (CoVe): +49 69 38 19 78,
- via chestnut ftp server (submitted, but the server is currently not online)

Monthly EFFO Meetings

The monthly EFFO meeting takes place each first Friday of a month in the Restaurant "Zunfthaus am Neumarkt" in Zurich. Its exact address is Zunfthaus am Neumarkt, Neumarkt 57, CH-8001 Zurich, phone +41 1 252 79 39. It can easily be reached from the main railway station using tram 3 or bus 31 (stop "Neumarkt").

As usual, the meeting starts 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.

EFFO's Annual General Assembly 1996

Announcement: EFFO's AGM 1996 will take place in the Restaurant "Herberge" in Teufenthal, Saturday, March 9, 1996. EFFO members will be invited in writing beginning of next year.

Imprint

Published by OS-9 International
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.