June 1994

OS-9 International

Issue 1/94

Published by:
Marc Balmer
Hagentalerstrasse 12
CH – 4055 Basel

Phone +41 61 43 55 01
Fax +41 61 43 55 02
E-Mail os9int@msys.ch
ISSN 1019-6714

Editors:
Marc Balmer
Carsten Emde

startup

Finally, the first end-users have received their copies of OS-9 version 3.0. The following two new features may be representative for the 3.0 upgrade changes:

First, task-switching is no longer disabled in system mode. In the past, it could happen that a high-priority process did not become active as quickly as expected because a low-priority process was in system mode where it could not be suspended by the task switcher. In 3.0, this no longer happens provided that the setting of the respective compatibility bit is appropriate to allow kernel preemption.

Another less spectacular upgrade change relates to the dsave utility. The "tmode -w=1 pause" string in the pre-3.0 version of dsave-generated backup procedures has not only served as an example for ignoring basic strategies of software engineering, but it also stimulated the invention of strange work-arounds. The '-t' option of the 3.0 dsave makes such work-arounds obsolete and shows that Microware continues to be interested in improving OS-9 even if only somewhat hidden details are concerned and even if some people may feel that it comes very late...

What is still lacking? The pre-2.4 file system used a file descriptor block that contained an 48-entry long segment list but only 47 of them were used. The 48th entry was always set to 0 — reserved for future expansion, e.g. to point to the location of the segment list's continuation. That's what we thought. Who at Microware can imagine our disappointment when we realized that from version 2.4 onwards this 48th entry was simply used to store an additional entry? Why was this free entry not used to let RBF dynamically expand its segment list so that error #217 (segment list full) would disappear from OS-9's error list? The fact that a 512-byte sector medium allows for a maximum of 99 segments does not help very much: Murphy teached us that, if there is a limitation anywhere in a system, this limitation will not only certainly be reached but also at a time and in a place where it leads to maximum trouble. Unfortunately, tools that refresh a scattered disk are not included in Microware's set of standard tools nor do they reasonably work if purchased from third parties. Microware's comment: Good old RBF will certainly never be expanded, but a new file manager with a more effective file structure can be expected in the future.

So, let's thank the people at Microware for the changes made in the 3.0 version, but let's also continue to keep them informed about what still needs to be done. One of the aims of OS-9 International is to provide a medium for such information and, again, we invite our readers to use OS-9 International for this purpose.

Carsten Emde

Keep Your Clock Up To Date, Part 2

Marc Balmer, Gerd Oxé

Exact time and date information

Every OS-9 computer is equipped with at least a clock chip or a general purpose device to provide a timer function. While these timers are certainly precise enough for measuring relatively short periods of time, they are not well suitable for keeping the correct time and date over a longer period, since almost every chip drifts by a few seconds per day. Even so called real-time clocks only approximately meet the real time.

Time signal stations can be a solution to this problem as they emit very precise time information using specifically modulated radio frequencies. Time signal stations are available in many countries; unfortunately, almost every transmitter is using its own time format, see [1].

The last issue of OS-9 International presented a receiver for the German DCF-77 time signal station [2]; the current article covers the software aspects.

Receiving the time signal

The DCF-77 time signal station uses a simple technique to transmit time information: the complete time information is packed into 59 bits (see diagram in the last issue of OS-9 International), and one bit is transmitted every second by reducing the transmitter's amplitude by about 75%. A logical 0 is encoded by an amplitude reduction that lasts 100 milliseconds; a reduction of 200 milliseconds denotes a logical 1. Using this scheme, the complete time information is transmitted during one minute. To allow for exact synchronization, the transmitter's amplitude is not reduced during the last second of every minute.

Connecting the receiver to the system

Various techniques are applicable to connect a DCF-77 receiver to a computer system: The most obvious way is to convert the two states of the amplitude to

a square wave signal, i.e. to a stream of zeroes and ones. In this case, the length of the reduction is determined to define the logical bit state.

A more sophisticated way that is used by most commercially available receivers and by the receiver presented in the first part of this article is to provide a data line with the decoded time information and a separate clock line. These lines can be used to load a shift register or to enter the data on an input port bit using the clock line as an interrupt generating strobe. A device driver is required to read the data line and to combine the bits to the complete time information. In this case, the time of the amplitude reduction is measured by the receiver.

The DCF-77 receiver presented in the first part of this article, however, not only provides the data and clock line but also the undecoded square wave signal on the clock line as an additional information. This allows for a rather unconventional, but elegant, method to both connect the receiver to the computer hardware and decode the time information. In essence, this method is based on interpreting the undecoded square wave signal as if it were an RS-232 bit stream signal so that a standard serial interface and the standard read command can be used.

Using the serial interface

Before the bit stream signal can be used as an RS-232 input signal, the required communication parameters must be determined and the controller be programmed accordingly: The longest duration of the low state of the square wave signal (200 ms) is taken as ten consecutive serial bits (one start bit, eight data bits, one parity bit) each of 20 ms, so that a total of 50 bits would be transfered per second. Consequently, if the serial interface is set to 50 Baud, 8 data bits and even-parity, a 20-ms section of the square wave signal represents one bit in the controller's input byte; the first 20-ms section, however, is not considered since it is interpreted as start bit. Figure 1 shows this relation between the DCF-77 timing and the 50-baud serial protocol. A logical 0 encoded from the time signal station as a low-level pulse of 100 ms duration, therefore, causes the serial controller's input byte to contain binary 11110000 (hexadecimal F0). A logical 1 encoded as a 200-ms low level pulse simply causes an input byte of 0.

The only hardware requirement to connect the square wave signal from the DCF-77 receiver to a serial RS-232 port of an OS-9 system is a TTL-to-V.24 level shifter. This shifter is realised using Maxim's MAX232 chip. Only the receive data (RxD) and ground (GND) pins are used, all other pins remain unused. Figure 2 shows the basic circuitry.

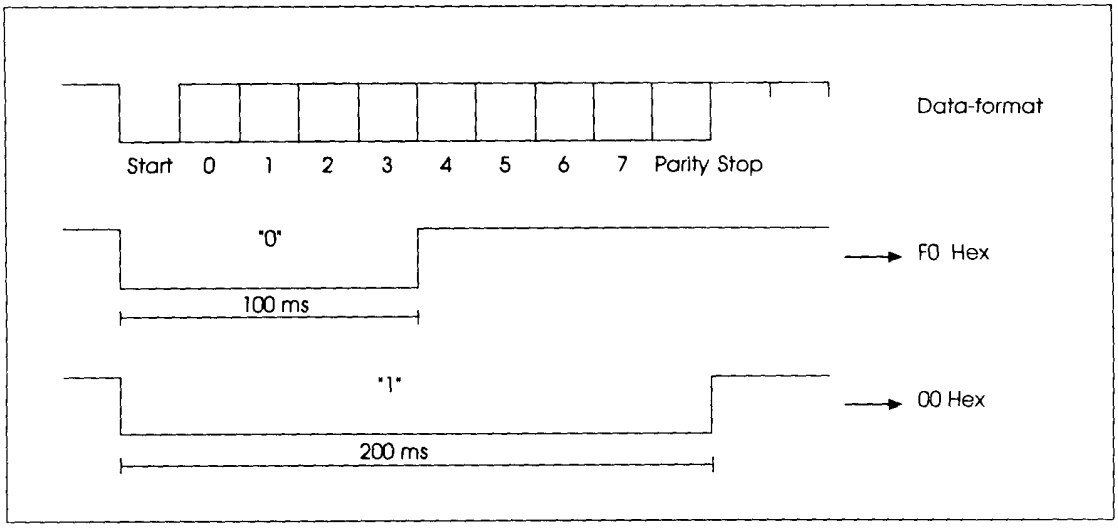

Figure 1: The DCF-77 time information in relation to a serial protocol.

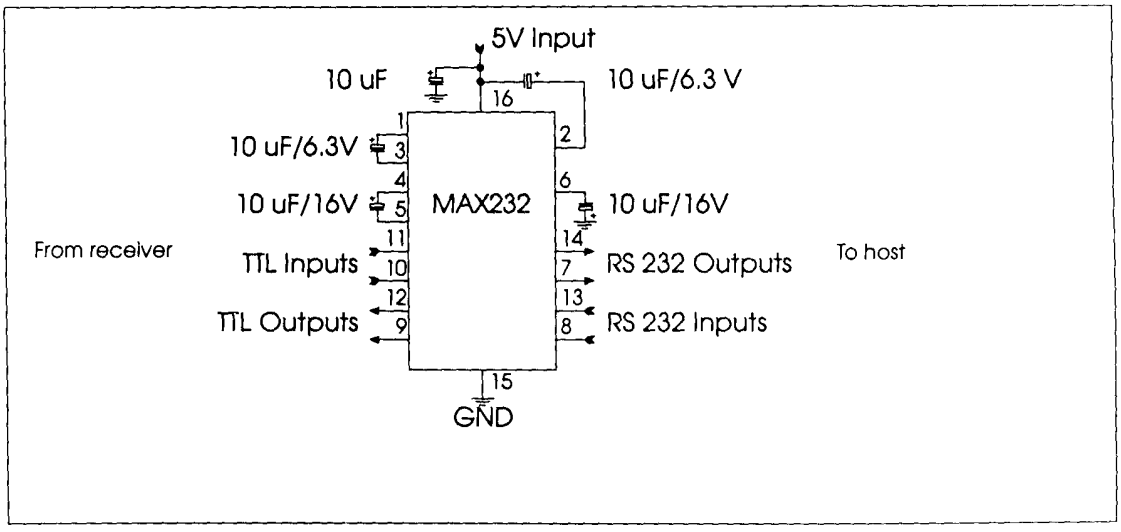

Figure 2: Circuitry required to connect the DCF-77 receiver to the RS-232 interface.

Decoding the time information

The very low level functions do nothing else than collecting the time bits and storing them in a structure. As the low-level interface may need to synchronize with the DCF-77 transmitter, it cannot be expected to return very quickly. Under worst-case condition, i.e. if the function is entered just after the first bit has been transmitted, it may take up to two minutes until the function has completely collected the time information.

Collecting the bits

There is, however, still a problem when using the serial interface to decode the time information. This problem is due to the evaluation of the parity bit. Both input byte values, hexadecimal F0 and 0, would need the parity bit to have the even (high level) state. This is alright in the first case; but in the second case (200 ms low level), the input signal has still low level when the parity bit is expected so that a parity error is generated. Fortunately, standard OS-9 SCF drivers cope with this situation since they return the E$Read error (errno 244) in case of a parity error. The decoding routine has, therefore, only to consider the error condition. If the read function returns without an error condition, a logical 0 was received from the time signal station. If the read function returns with an error, a logical 1 has been received. The resulting bit collecting routine is surprisingly simple, especially if no error checking is done:

```
int dcf77bit(int fd)
{
  char c;

  return read(fd, &c, 1) == 1 : 0 ? 1;
}
```

If a particular serial line device driver does not correctly handle the parity error bit of the communications controller, the return value of the read function must be considered.

The time decoding algorithm

At any time the decoder is started, it must synchronize with the DCF-77 time signal station. To do so, it waits for a delay between two bits that is significantly longer than one second. The following function uses a delay of 1.5 seconds:

```
#include <time.h>
#define CUR_TCK ((clock_t) _getsys(D_Ticks ,4))
#define 150_TCK (CLK_TCK + (CLK_TCK >> 1))

static int dcf77sync(int fd)
{
    register long last_tick, tick;
    register int this_bit;

    tick = CUR_TCK;
    do {
        last_tick = tick;
        this_bit = dcf77bit(fd);
        tick = CUR_TCK;
    } while (tick - last_tick < 150_TCK);
    return this_bit;
}
```

After this prolonged gap, the next bit to receive is bit zero of the subsequent time information packet. The algorithm then picks up the bits until all 59 data bits are collected. In case the algorithm falls out of synchronization and misses some bits, maybe due to bad reception of the radio signal, it detects the data loss by the time elapsed between two bits which must not be more than one second. In this case, the algorithm re-synchronizes with the DCF-77 signal and restarts.

Once the complete time information is received, it is immediately decoded and converted into an OS-9 standard time structure (struct sgtbuf, defined in time.h) which is returned to the caller after the bit zero of the following minute has been received. The time obtained can then be used in a subsequent call to the setime() function.

```
clock_t dcf77time(int fd, struct sgtbuf *t, int *wday, int *dst)
{
  static long last_tick = 0L;
  long tick;
  static int sec = 0;                /* Current second */
  static int bit[59];

  flush(fd);                         /* Flush any pending input */

  while (sec < 59) {
    bit[sec] = dcf77bit(fd);
    tick = CUR_TCK;

    if (tick - last_tick > (CLK_TCK + 5) {
```

```
      bit[0] = dcf77sync(fd);
      last_tick = CUR_TCK;
      sec = 1;
      t->t_year = 0;
    } else {
      last_tick = tick;
      t->t_second = sec++;
      if (sec < 60 && t->t_year)
        return 0L;
    }
  }

  t->t_second = (char) 0;
  sec = 0;

  t->t_minute = (char) bit[21] + (2 * bit[22]) + (4 * bit[23]) +
                (8 * bit[24]) + (10 * bit[25]) + (20 * bit[26]) +
                (40 * bit[27]);

  t->t_hour = (char) bit[29] + (2 * bit[30]) + (4 * bit[31]) +
              (8 * bit[32]) + (10 * bit[33]) + (20 * bit[34]);

  t->t_day = (char) bit[36] + (2 * bit[37]) + (4 * bit[38]) +
             (8 * bit[39]) + (10 * bit[40]) + (20 * bit[41]);

  if (wday != NULL)
        *wday = bit[42] + (2 * bit[43]) + (4 * bit[44]) - 1;

  t->t_month = (char) bit[45] + (2 * bit[46]) + (4 * bit[47]) +
               (8 * bit[48]) + (10 * bit[49]) - 1;

  t->t_year = (char) bit[50] + (2 * bit[51]) + (4 * bit[52]) +
              (8 * bit[53]) + (10 * bit[54]) + (20 * bit[55]) +
              (40 * bit[56]) + (80 * bit[57]);

  if (dst != NULL)
        *dst = bit[17];

  bit[sec] = dcf77bit(fd);       /* Wait for second 0 */
  last_tick = CUR_TCK;

  /* Return the tick that was valid when the minute started */

  return bit[sec++] == 1 ? last_tick - 20 : last_tick - 10;
}
```

If the time deocding function is re-entered within less than one second interval, it does not need to re-synchronize. In this case, the function waits for the next

data bit, stores it, increments t_second in the time structure and immediately returns to the caller. In consequence, the function needs at least two minutes only for the first call; if, however, called regularly, the function returns after about one second so that, for example, a display can continuously be updated.

We are always late

The above-described method to decode the DCF-77 data bits has one disadvantage: We are always late. When the read function returns, indicating that a new second just started, we are already late for 100 or 200 milliseconds which is the time that is needed to transmit the data bit. This delay must be taken into consideration when the exact time of the starting of a new second is needed. Nevertheless, calculating the correction factor is easy; if the bit just received is a logical 1, we are CLK_TCK / 5 ticks late; if it is logical 0, we are only CLK_TCK / 10 ticks late.

Adjusting the system time

While receiving and decoding the time information is comparingly simple, introducing the current time into the computer system may be a complex task. Generally, it is recommended to set the system time during the boot procedure, for example as part of the startup procedure. In this case, everything is simple and no problems will arise. If it is, however, intended to update the system time while the computer is running and possibly executing programs that rely on absolute time or on time intervals, serious problems may occur.

Changing the time at run-time

There are two generally different concepts to change the system time at run-time. The first concept gives maximum priority to the continuity of the time, i.e. the time may be compressed or stretched, but under no circumstances may a discrete time value get lost. The second concept regards time as a sequence of time units with fixed length which can neither be stretched nor be compressed, but it is allowed to miss or insert a time unit.

The distinction between the two different methods is necessary as in a real-time environment the time must not be changed without prior consideration of

the software that is running. Imagine a demon program that has to start other programs at a given time: If the continuity of the time is broken-up, a particular program may never be started. Such software would only run properly, if time adjustment is done by stretching or compressing the time axis.

Other software may not rely on the absolute time but on the accuracy of the system clock (tick) rate. If, in this case, the time is adjusted by speeding-up or slowing down the tick rate (i.e. stretching or compressing the time axis), this software probably will fail. Such software would only run properly, if time adjustment is done just by changing the time settings.

If both types of software simultaneously run on the same system, the time cannot be adjusted without producing unpredictable results. In this case, the system time should better not be adjusted at run-time.

Changing the absolute time

Changing the absolute time is very easy as OS-9 provides the F$STime function to set the absolute system time.

Adjusting the tick rate

Adjusting the tick rate at run-time is a delicate operation that OS-9 does not support per se. The tick rate can be changed by either inserting or omitting a system tick (resulting in a twice or half as long tick) or by changing the tick duration. In the latter case, the heartbeat of the system must be changed by reprogramming the clock device which is not possible in all cases. In the first case, the clock device driver must be modified.

References

[1] Klawitter, Gerd, *Time Signal Stations*, 11th edition, Meckenheim: Siebel, 1988.

[2] Oxé, Gerd, *Keep Your Clock Up To Date*, in "OS-9 International", 2/93, page 32.

Marc Balmer can be reached through the OS-9 International *office or by E-Mail at <balmer@ifi.unibas.ch>.*

Gerd Oxé can be reached by E-Mail at <gerd@tangens.pr.net.ch>.

Simulating dip-switches

Marc Balmer

In a software project for an embedded graphics controller, the software had to be developed at a time when the hardware was not yet available. A major function of the code relies on the settings of a dip-switch bank present on the target machine but not on the development system. In detail, the dip-switch allowed for the definition of the screen orientation to be either upright or upside-down.

It is, of course, possible to compile two different versions using C language preprocessor statements. This, however, has the disadvantage that the binary code for the target system cannot reliably be tested on the development system. It was, therefore, decided to produce a program that runs with and without a dip-switch bank. In detail, a library was written that contains functions to read out a physically existing dip-switch bank or to simulate it, if not existing. In addition, it was planned to allow for the simulation of the dip-switch setting also on the target system, so that different settings can be tested without actually altering the switches.

The most obvious way to simulate the dip-switches is to use an environment variable, e.g. "DIPSWITCH".

```
unsigned char dsw_get(void)
{
    register char *env;

    env = getenv("DIPSWITCH");
return env ? (unsigned char) atoi(env) : 0;
}
```

If, for example, the MSB position of the dip-switch has to be simulated, it is sufficient to enter the shell command "setenv DIPSWITCH 128" prior to starting the application software.

Unfortunately, this procedure can only be used for programs and not for device drivers since the latter have no access to the shell environment variables. On the other hand, it is a device driver and normally not a user program that would need to inspect the settings of a dip-switch bank.

The final solution was to use a data module that stores the current dip-switch settings and to provide another dsw_get() function that links to this module and either returns the simulated setting or inspects the memory location of the

hardware dip-switch bank. If the module is not found, a default value of 0 is assumed and returned.

The dsw_set_val() function is used to set a dip-switch value, the dsw_set_adr() function allows to set the memory location of the dip-switch bank. A flag in the data module indicates whether the dip-switch value is simulated or not. Please note that the dip-switch data module must be present in memory in any case, even if a hardware dip-switch bank is available.

The header file dsw.h contains all necessary definitions:

```
/* dsw.h */
#include <module.h>
#define DSW_MOD    "dsw.inf"

typedef struct {
    struct modhcom _mh;
    unsigned long dsw_adr;      /* Address of real dip-switch (if any) */
    unsigned char dsw_flags;    /* Flags, see below */
    unsigned char dsw_val;      /* Simulated value */
} mod_dsw;

/* Dipswitch flags */

#define DSW_SIM     0x01    /* Dip-switches are simulated */
#define DSW_NEG     0x02    /* Sense of dip-switch is inverted */
```

The utility program "dsw" is provided to manipulate and inspect the current dip-switch settings:

```
#include "dsw.h"

unsigned char dsw_get(void)
{
    register mod_dsw *m;
    register unsigned char dsw;

    if ((m = (mod_dsw *) modlink(DSW_MOD, 0)) == (mod_dsw *) -1)
        return 0;
    else {
        if (m->dsw_flags & DSW_SIM)
            dsw = m->dsw_val;
        else
            dsw = *((unsigned char *) m->dsw_adr);
    }

    if (m->dsw_flags & DSW_NEG)
```

```
        dsw = ~dsw;

    munlink(m);
    return dsw;
}
```

Another utility program contains the above-mentioned dsw_set_val() and dsw_set_addr() functions. In addition, the data module can be saved to disk so that it may be distributed together with a specific software version.

```
/*
 * dsw.c - DIP Switch simulation
 *
 * Copyright (C) 1994 by OS-9 International and Marc Balmer, CH-4055 Basel.
 * All rights reserved.
 */

#include <stdio.h>
#include <errno.h>

#include "dsw.h"

void main(int argc, char **argv)
{
    register unsigned char dsw = 0;
    register unsigned char n;
    int i, f, j;
    char *dsw_str = NULL;
    unsigned long arg;

    for (i = 1; i < argc; i++) {
        if (argv[i][0] == '-') {
            for (j = 1; argv[i][j] != 0; j++)
                switch (argv[i][j]) {
                    case '?':
                        usage();
                    case 'a':
                        if (argv[i][++j] == '=')
                            ++j;
                        sscanf(&argv[i][j], "%X", &arg);
                        dsw_set_adr(arg);
                        break;
                    case 'v':
                        if (argv[i][++j] == '=')
                            ++j;
                        sscanf(&argv[i][j], "%X", &arg);
                        dsw_set_val((unsigned char) arg);
                        break;
```

```
                case 'c':
                    dsw_clr_sim();
                    break;
            }
        } else
            dsw_str = argv[i];
    }

    if (dsw_str == NULL)
        dsw_print(dsw_get());
    else if (strlen(dsw_str) == 8) { /* Supports only 8-bit switch */
        for (n = 0; n < 8; n++)
            if (dsw_str[n] == '1')
                dsw += 1 << (7 - n);
        if (dsw_set_val(dsw) == -1)
            exit(_errmsg(errno,
            "Can't create dipswitch data module \"%s\".\n", DSW_MOD));
        dsw_print(dsw_get());
    }
}

void usage(void)
{
  fprintf(stderr, "Syntax: dsw [<opts>] [<settings>]\n");
  fprintf(stderr, "Function: display or set dip-switch settings\n");
  fprintf(stderr, "Options:\n");
  fprintf(stderr, "   -v=<hex> set dip-switch value to <hex>\n");
  fprintf(stderr, "   -a=<hex> set the dip-switch memory location to <hex>\n");
  fprintf(stderr, "   -c       clear dip-switch value (stop simulation)\n");
  exit(0);
}

/* Set dip-switch value */

int dsw_set_val(unsigned char dsw)
{
    register mod_dsw *m;
    register int unlink = 1;

    if ((m = (mod_dsw *) modlink(DSW_MOD, 0)) == (mod_dsw *) -1)
        if ((m = (mod_dsw *) _mkdata_module(DSW_MOD,
            sizeof(mod_dsw) - sizeof(struct modhcom),
            mkattrevs(MA_REENT, 0),
            MP_WORLD_MASK | MP_GROUP_MASK | MP_OWNER_MASK)) == (mod_dsw *) -1)
            return -1;
        else
            unlink = 0;

    m->dsw_flags |= DSW_SIM;
```

```
    m->dsw_val = dsw;
    if (unlink)
        munlink(m);
    return 0;
}

/* Set (real) dip-switch base address */

int dsw_set_adr(unsigned long adr)
{
    register mod_dsw *m;
    register int unlink = 1;

    if ((m = (mod_dsw *) modlink(DSW_MOD, 0)) == (mod_dsw *) -1)
        if ((m = (mod_dsw *) _mkdata_module(DSW_MOD,
            sizeof(mod_dsw) - sizeof(struct modhcom),
            mkattrevs(MA_REENT, 0),
            MP_WORLD_MASK | MP_GROUP_MASK | MP_OWNER_MASK)) == (mod_dsw *) -1)
            return -1;
        else
            unlink = 0;

    m->dsw_adr = adr;
    if (unlink)
        munlink(m);
    return 0;
}

/* Clear dip-switch simulation flag */
int dsw_clr_sim(void)
{
    register mod_dsw *m;
    register int unlink = 1;

    if ((m = (mod_dsw *) modlink(DSW_MOD, 0)) == (mod_dsw *) -1)
        if ((m = (mod_dsw *) _mkdata_module(DSW_MOD,
            sizeof(mod_dsw) - sizeof(struct modhcom),
            mkattrevs(MA_REENT, 0),
            MP_WORLD_MASK | MP_GROUP_MASK | MP_OWNER_MASK)) == (mod_dsw *) -1)
            return -1;
        else
            unlink = 0;

    m->dsw_flags &= ~DSW_SIM;
    if (unlink)
        munlink(m);
    return 0;
}
```

```
/* Print current dip-switch settings */

void dsw_print(unsigned char dipswitch)
{
    register unsigned char mask;

    printf("Current DIP Switch settings:\n\n");
    printf(" 7    6    5    4    3    2    1    0\n");
    mask = 0x80;
    while (mask) {
        printf("%s  ", dipswitch & mask ? "ON " : "OFF");
        mask >>= 1;
    }
    printf("\n");
}
```

Conclusion

Why does such a simple task like simulating a dip-switch merits the development of the above software? It was, of course, not the primary aim of this article to provide ready-to-use software but arguments for writing hardware-independent code. The OS-9 platform makes this easy and data modules are an important feature that may often be used for this purpose.

Marc Balmer can be reached through the OS-9 International *office or by E-Mail at <balmer@ifi.unibas.ch>.*

Can't execute "J"

Carsten Emde

Have you ever encountered the rather cryptic shell error message 'shell: can't execute "J"'? Here is an explanation for this message and a small C language program that provides a more helpful message. Whenever a command is entered at shell level, the first string is regarded as command and searched at the following places in the given order: Module directory, current execution directory, directories specified in the PATH environment variable, and current data directory. A file in the current execution directory and in the PATH directories may be both an executable OS-9 module or a procedure file; a file in the current data directory, however, may only be a procedure file. If the latter is not the case and an executable OS-9 module is found instead, the shell will try to execute it as a procedure file anyway. Since all valid OS-9 modules start with the illegal instruction code hexadecimal 4AFC, this number is erroneously interpreted as ASCII characters "J" and vertical bar with the sign-bit set. The latter is, obviously, regarded as an end-of-word character so that the shell searches for a program called "J" and tries to execute it. Since this fails, the above error message appears.

A remedy for this inconsistent behavior is given in the following program that is called "J" and that will display a more digestible error message.

```
#include <module.h>

/*
 *  this small program is intended to let OS-9 novices no longer
 *  suffer from the 'shell: can't execute "J"' message
 */
main(argc, argv)
int argc;
char *argv[];
{
  int i;

  if (argc == 2 && argv[0][0] == (char) (MODSYNC >> 8) &&
                   argv[0][1] == '\0' &&
                   argv[1][0] == (char) (MODSYNC & 0xff) &&
                   argv[1][1] == '\0')
    help();
  else
    usage();
}
```

```
/*
 * print explanation why there was a problem
 */
help()
{
  printf("shell: the requested program could not be forked, since it is\n");
  printf("located in the current data directory and, therefore, regarded\n");
  printf("as a procedure and not as a program.\n");
  printf("One of the following actions will solve the problem:\n");
  printf("1. load the program (load <prog> -d).\n");
  printf("2. copy the program into the current execution directory.\n");
  printf("3. copy the program into one of the PATH directories.\n");
  printf("4. make the current directory the execution directory.\n");
  printf("5. include the current directory into the PATH variable.\n");
  printf("6. enter the fully-qualified file name.\n");
  printf("7. use a less rudimentary shell.\n");
}


/*
 * print usage, if the program was started manually
 */
usage()
{
  printf("Sorry, this program (j) does not provide any meaningful\n");
  printf("user-accessible functionality.\n");
}
```

Carsten Emde can be reached by E-Mail at <carsten@ce.pr.net.ch>.

Parallel processing under OS-9

Carsten Emde

Whenever the result of a computer's calculation is required in less time and algorithms and compiler already ensure maximally optimized binary code, more computer power is needed. This can be realized by installing a more powerful CPU board, if available. If not available, however, parallel processing must be employed. The way this is done, depends — among other things — on the number of tasks involved in the particular computing procedure.

Single-task projects

A typical time-sensitive single-task project is, for example, a fore-casting program that, obviously, must have finished the calculations early enough before the relevant time period starts: A weather forecast for the next weekend would, of course, make no sense if only available on Sunday evening. Such problems require either more single CPU power or the availability of several CPUs and an adequate tool that separates the given algorithm into parallel tasks. For the OS-9 operating system, however, single-task computing power is limited to the maximum speed that can be achieved using Motorola's CISC processors family since compilers that transform a single task source into concurrently running procedures are normally not available for OS-9. Single-task projects that need more computing power than about 40 MIPS can, therefore, not be realized under OS-9.

Multi-tasking projects

On the other hand, many computer projects, especially under OS-9, already consist of several concurrently running procedures; such situation can much easier be transformed into parallel processing than the above described situation of single-task computing. The current article, therefore, focuses on parallel processing of a multi-tasking environment under OS-9. Another reason for this article is that a new principle of multi-processing has recently been made available: VMEbus boards with tightly-coupled processors. The term "tightly-coupled" means that the processors are connected to the same bus and have, therefore, common access to the entire memory. One processor is the master processor

that, by default, receives interrupts from the peripheral device controllers, the other processors have special control registers for reset and interrupt, but this is the only difference between them. Currently existing boards have two processors installed, but future boards may even have more. Thus, multi-processing does no longer require several CPU boards being connected to each other via VMEbus but may be done on one single board. As a consequence, a drastic increase in the performance of such systems can be achieved since all memory accesses are local and no longer limited by the VMEbus.

In general, there are three different ways to transform an OS-9 multi-tasking project into multi-processing: 1. using independent OS-9 systems that are connected via network, 2. installing a special kernel extension that distributes the active tasks to all processors instead of only one, and 3. using a driver interface to run programs without OS-9 in the additional CPUs. In addition, Microware is working on an OS-9000 multi-processor system ("Hydra project") but has not yet announced any definite plans for release [1].

Independent OS-9 systems

Several independent OS-9 systems can run simultaneously one per processor provided that the memory sections assigned to the processors do not overlap. This is normally achieved by individually defining a particular processor's memory in the 'init' configuration module. In a double-processor board with 32 MByte RAM one would, for example, define start address and memory size as 0 and 0x1000000 for the first processor, and as 0x1000000 and 0x1000000 for the second processor, respectively. After booting the first OS-9 system, a special download program may be started that boots the second CPU. If the second CPU does not need any specific mass storage, its OS9Boot file may contain a '/dd' device descriptor for a RAM floppy whose disk image is also provided as part of the OS9Boot file. The 'init' configuration module must then, of course, specify '/dd' as the primary disk device. Another, very elegant, feature is to incorporate the NFM file manager, drivers and descriptors into the OS9Boot file so that the newly booted OS-9 system may access all peripheral devices via OS9Net through the master processor. Since both OS-9 systems may access each other and the NFM file manager allows to access not only RBF devices but all other I/O devices including pipes, the network link may as well be used for synchronization purposes.

Advantages and disadvantages

The advantage of having two independent OS-9 systems running on one CPU board is that there is no need for specific software. Already existing software that was, for example, written for a master and a slave CPU board connected to each other via backplane net may easily be adapted to run on a double-processor board. But not all software projects may easily be transformed to communicate via network and not all customers are familiar with the configuration that is needed for such master/slave applications. It may also be difficult to decide in advance what part of the project should run on what processor. In addition, it must be mentioned that two OS-9 runtime licenses must be bought since two independent kernels are needed, irrespective of whether they run on one or on two CPU boards.

The "Doubler"

Recently, the Syac company (Sy.A.C. S.R.L., Trieste, Italy) has released OS-9 system modules that implement multi-processing in a single OS-9 system. Essentially, the existing OS-9 scheduling algorithm is modified so that the active tasks are distributed not only to one but to two processors. This software is called "Doubler" since it supports two processors. After installing the "Doubler" module, the single-task behavior of the system is not affected, i.e. a benchmark program such as the Dhrystone program has virtually the same performance with and without the "Doubler". If, however, the benchmark program is simultaneously started from two terminals or from two MGR windows, the performance can nearly be doubled. The increase in performance, however, depends from the particular application and may be less pronounced. The following values have, for example, been measured (68040, 33 MHz):

	Without "Doubler"	With "Doubler"	Percent
One Dhrystone	50.352	50.120	100
Two Dhrystones	24.975	49.309	197
Three Dhrystones	16.563	32.808	196
C compilation			ca. 160

Advantages and disadvantages

The advantage of the "Doubler" is, as in the first method, that any existing software can be used without restriction. Whenever at least two processes are active, the increase in performance takes place. This increase, however, is not always as big as in the above example since only user programs are executed concurrently. If a program heavily relies on kernel calls, the increase in performance may be less pronounced. This must, therefore, be considered when deciding between the two first methods of parallel-processing. The frequency of I/O calls, however, is not an important argument, since I/O is exclusively handled by the master processor in both methods.

Driver interface (TCP/SP)

Finally, a software package, the Tightly-Coupled Processors Support Package, has been developed that manages the additional CPU through a driver interface. This driver interface, although based on the SCF file manager, does not primarily provide read and write functionality. The main part of the driver is implemented in form of SetStat calls. There are, for example, specific calls to start and stop the additional processor, to install exception handlers and to provide communication channels using interrupts and signals. In addition, a 'fork' function is available that allows to run OS-9 program modules on the additional processor. This function fully emulates the OS-9 F$Fork kernel call, i.e. memory is allocated for static and global data and those pointer variables that refer to the program code or to static data are made position-independent. Bindings for the C language are available to facilitate the calling interface. The following example presents the code that is necessary to let the additional processor execute a loop for one second and to stop:

```
#include <modes.h>

extern int errno;

void prog(void);

void main(void)
{
  char          *cpu2name = "/cpu2";
  int           cpu2;

  if ((cpu2 = open(cpu2name, S_IREAD | S_IWRITE)) == -1)
```

```
    exit(_errmsg(errno, "can't open '%s' CPU device due to ",
      cpu2name));

  _ss_tcpsp_runlow(cpu2, (char *) 0, prog);
  /* no stack memory needed */

  sleep(1);

  _ss_tcpsp_stop(cpu2);
}

#asm
prog:
  cinva bc ; invalidate both caches
loop
  bra.s loop
#endasm
```

Advantages and disadvantages

The advantage of this method is that small and effective programs can run on an additional processor, that there is no overhead from the operating system and that no run-time licence is required. In consequence, the Tightly-Coupled Processors Support Package is ideally suitable for image processing (filters, reduction in bit depth, template matching, etc.) and number-crunching. The fact that no specific debugging facilities are available cannot be considered an important disadvantage, since both processors are connected to the same memory so that the development of the slave software including debugging can easily be done on the master processor. It must, however, be realized that — except inquiring system globals and low-level string output — kernel functions such as memory allocation, I/O etc. are not available. Standard software can, therefore, not easily be adapted to run on an additional processor using this method.

Conclusion

In comparison to some other operating systems, OS-9 is less well equipped with already existing and generally available support software for multi-processing. There are, however, at least three distinct ways to transform a multi-tasking environment into multi-processing. All of them have specific advantages so that in many cases where more OS-9 computing power is needed, an adequate method is available.

References

[1] Kemp, Douglas, *Peter Dibble at CERN*, in "OS-9 International", 1/93, page 13.

Carsten Emde can be reached by E-Mail at <carsten@ce.pr.net.ch>.

Of Mice And Men

Marc Balmer

Connecting a pointer device such as a mouse or a track ball to an OS-9 system is definitely not an easy task. This observation has probably been made by everyone who tried to install an OS-9 graphic systems such as the MGR Window Manager or X Window that relies on a pointer device.

There are mainly two reasons for this difficulty. First, generally available mice use a variety of different transmission protocols; this is normally not a problem, since most mice are designed to be used in conjunction with DOS computers and applicable drivers are normally delivered together with every mouse. Users of other operating system, however, must themselves provide adequate driver software. Secondly, mice manuals usually do not contain any pertinent information that is needed for writing a driver and the technical manual is either not available or, if available, contains mostly DOS-related information.

The current article, therefore, gives a detailed description of the most frequently used mouse protocols and appropriate recommendations of how to distinguish them between each other.

How The Mouse Works

At first glance, the mouse accomplishes a fairly simple task, namely to convert physical movement to numerical values — the mouse coordinates. At a closer view, however, the mouse is a rather complicated mechanical device with its own microprocessor, serial interface and movement tracking mechanism.

Most mice use an opto-mechanical principle: Whenever the mouse's position on a surface is changed, a rubber-coated ball rotates. Using two optical encoder disks, LEDs and photo-transistors, the movement of the mouse is encoded into a series of pulses for each axis separately with 90 degree phase difference. The number of pulses is a measure of the distance, the interpretation of the phase indicates the direction. The positional resolution and the speed range may be very high so that more than 100 dpi and up to 5 meters per seconds are possible.

To relinquish the host processor from counting the individual horizontal and vertical pulses, a microcontroller is built into the mouse that keeps track of the current position and that informs the main processor via the serial interface of mouse movements or buttons being depressed or released.

Hardware Prerequisites

There are a few hardware prerequisites that must be fulfilled before a mouse can be connected to a serial port. As the mouse is powered from the RTS and/or DTR line of the serial interface, at least the RTS line must be wired. Some line driver cicuitry does not provide enough current through the RTS line to drive the mouse properly; the DTR line must be wired as well in this case.

If the DTR is not wired and the RTS line does not provide enough current, the mouse must be powered by an external battery or mains adaptor. See figure 1 for the simple schematics.

If the mouse is powered from the DTR line, it is still desirable to connect the RTS line as well, because the RTS line may be used to automatically detect the type of a Microsoft compatible mouse.

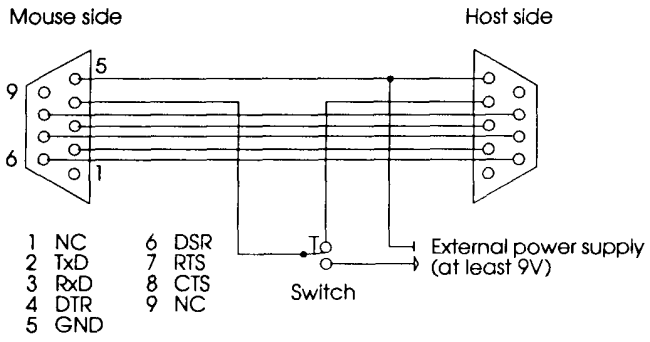

Figure 1: Powering the mouse from an external source.

Mouse Protocols

The most frequently used mouse protocol are 1. the Five Byte Packed Binary Format, 2. the Standard Microsoft Protocol (MS) and 3. The Extended Microsoft Protocol (M+).

The Five Byte Packed Binary Format

Everytime the position of the mouse is changed or a button is depressed or released, the mouse sends a 5-byte long data packet to the host computer through the serial interface.

The first byte has its most significant bit always set. Bits 3 through 6 are not used and are always cleared. Bits 0 through 2 represent the current button state. Bit 0 indicates the state of the right mouse button, bit 1 indicates the state of the middle mouse button and bit 2 represents the left button. Note that a value of 0 in bits 0 through 2 means that the corresponding button is currently depressed. In the normal idle state of the mouse with no buttons pressed, the first byte thus always reads hexadecimal 87.

Each of the following four bytes contain information about the relative change in the mouse position; they are tranmitted as two's complement binary numbers yielding each a range of -128 to +127. The second byte and the third byte represent the horizontal and the vertical offset, respectively, i.e. how far the mouse has been moved in horizontal and in vertical direction since the last report was generated and sent to the host. The last two bytes also contain horizontal and vertical offsets, but these numbers indicate how far the mouse has been moved since the current report was started. To get the complete offset, the first and second offset values of both horizontal and vertical displacement must be added to each other.

As the mouse only transmits relative motion data, it is up to the host's application program to maintain the absolute position of the mouse. Although there exist data formats that transmit absolute mouse positions, they should better be avoided.

The Standard Microsoft Protocol (MS)

The standard Microsoft data format uses seven data bits at 1200 bps. Each report is three bytes long and encodes the relative movement since the last report similar to the above Five Byte Packed Binary Protocol. In detail, the following definitions apply:

The first byte transmitted encodes the mouse button data in bit number 5 (left button) and bit number 6 (right button). Additionally, the first byte contains the two most significant bits of the relative movement in x-direction (bit 1 and 0) and y-direction (bit 3 and 2), respectively. To allow for synchronisation with the mouse, bit number 6 of the first byte is always set to logical 1, whereas in the subsequent bytes it is always set to logicl 0.

The second byte contains the remaining six bits of the relative horizontal movement. This byte can directly be used as a signed binary value. The last byte

encodes the six least significant bits of the relative vertical movement. These bits can also be used as a binary number.

To decode the full movement information carried in the data packet, the application software needs to add bit number 0 and bit number 1 of the first byte to the second data byte to get the relative x-movement. To decode the relative y-movement, bit number 3 and 2 of the first byte must be added to byte number 3.

When the middle button on a three-key mouse is pressed, bit 5 and 4 of the first byte are both set to logical 1.

The Extended Microsoft Protocol (M+)

The Standard Protocol (MS) cannot encode a situation where the third button is pressed additionally to one of the other buttons. Therefore, the Extended Microsoft Protocol (M+) has been defined that can handle three mouse buttons independently from each other. In all cases except when the middle mouse button is pressed, the Extended Protocol behaves exactly like the Standard Protocol. When the middle mouse button, however, is pressed or released a fourth byte is added to the data packet. This additional byte contains the state of the middle button in bit number 5. The remaining bits encode the device type which is logical 0 on all bits for a mouse device. The remaining values are reserved for future use.

Mouse types

Different mouse types differ in the number of buttons, the transmission protocol and whether they are programmable or not. To distinguish mice from each other, they can be put into two different categories: Mice that are Microsoft compatible and mice that are not.

Microsoft compatible mice

Three different Microsoft compatible mouse types exist, type M, type V and type W. Although only slightly different, the classification is needed, since the different behavior primarily affects transmission properties and recognition procedures.

Type M mice

Type M devices from Microsoft or 100% Microsoft compatible serial mice from other manufacturers use the MS protocol and operate at 1200 bps. They have no receive capability, i.e. type M mice generally cannot be programmed.

Type V mice

Type V mice are also Microsoft compatible but they support a third mouse button and, therefore, require another protocol – the Microsoft Plus (M+) protocol. Type V mice neither have any receive capability.

Type W mice

Type W mice are more versatile than type M and V mice. They support the MS protocol as well as the M+ and the Five Byte Packet Binary format. They can operate at 1200 bps or 9600 bps. Type W mice can accept data from the host over the serial line; in consequence, the behavior of a type W mouse can be programmed (sensitivity, baud rate, mouse protocol etc.).

Not Microsoft compatible mice

Type C mice

Type C mice are the most versatile mice. They can receive commands from the host and operate at 1200, 2400, 4800 or 9600 bps. Three data formats are supported by type C devices: MM Series protocol, Relative Bit Pad One Packet Binary and Five Byte Packed Binary.

Type C mice can operate in incremental stream mode or in prompt mode. Additional commands allow for system checking and information retrieval on the mouse type and revision number.

Detecting the mouse

As there are different mouse protocols available, it is important to properly identify the mouse type prior to using it. Every mouse has an option that allows for identification of the device. Unfortunately, the identification procedure is not the same for all mice.

Whereas type M, V and W devices respond to a state of change on the RTS line, type C mice respond to a status command which must be sent over serial line. To further complicate the identification process, type W mice must receive a command after RTS toggle function in order to properly being identified.

The RTS toggle function

All mice but type C mice respond to an RTS toggle by sending one or two characters to the host:

Mouse type	*Response*
Type M	"M"
Two-button type V	"M"
Three-button type V	"M3"
Two-button type W	"M"
Three-button type W	"M3"

The first character will be sent at last 20 ms after the rising edge of the RTS line. If a second character is to be sent, it will be transmitted at last 100 ms after the first character.

Mouse Identification

The process of completely identifying a mouse is somewhat more complicated. The following nine steps must be performed in order to automatically detect the mouse type:

1. Open the serial line in raw mode at 1200 bps with 7 data bits.
2. Toggle the RTS line.
3. If the response is "M" or "M3" proceed to the next step, else set speed to 9600 bps and return to **step 2**.

4 If the response is neither "M" nor "M3", it is not a type M, V or W mouse. Goto to **step 6** to see if it is a type C mouse.

5 Send the status command to the mouse. If it responds, it is a type W mouse. If it does not respond, it is either a type M or V device, depending on the RTS toggle response. As the mouse is identified, goto **step 9**.

6 Set the serial line to 1200 bps and 8 data bits.

7 Send the status command. If the mouse returns a proper status message, goto to **step 9**.

8 Multiply the serial line speed by two. If the speed exceeds 9600 bps, abort, the mouse cannot be identified else go back to **step 7**.

9 The mouse has successfully been identified. Return information.

Mouse Operating Modes

In addition to selecting the transmission protocol, different operating modes may be selected by sending appropriate commands to the mouse.

Stream vs. Prompt Mode

In stream mode, the mouse automatically sends a report to the host whenever the mouse is moved or a button is pressed or released. A simple analogue to the stream mode is an interrupt driven device which delivers its data to the operating system upon reception.

In prompt mode the mouse only sends a report to the host computer when asked for. Prompt mode can be compared to a polling device.

Polling a mouse contradicts the philosophy of OS-9 and real-time data processing; hence, the prompt mode is normally not recommended.

The Mouse Report Rate

There is, however, a problem when the mouse is in stream mode and the user moves the mouse quickly over the desktop: The mouse continuously sends position reports to the host at a very high rate and, thus, unnecessarily increases the system load.

To prevent the computer system from frequent serial line interrupts, the mouse report rate can be limited; a value between ten reports per second and up to 150 reports per seconds can be programmed. If the mouse position or state is not altered, of course no report is generated.

Conclusion

There are no specific requirements to connect a mouse to a computer system that OS-9 does not provide. The above-named procedure for mouse recognition, i.e. toggling the RTS line and waiting a defined time for one or more input bytes, can easily be realised by an OS-9 program or, better, by an OS-9 driver. The main prerequisite for writing such software is the availability of the technical data; this was the aim of the current article. One of the next issues of OS-9 International will present the driver software for a mouse subsystem that works with any of the described mouse types without requiring any specific configuration.

Marc Balmer can be reached through the OS-9 International *office or by E-Mail at <balmer@ifi.unibas.ch>.*

The EFFO 1994 general assembly

Reto Peter, EFFO secretary

The EFFO general assembly was held on Saturday, February 19, 1994, from 2:15 pm till 6:50 pm at the Gasthof zur Herberge in Teufenthal (near Aarau). All registered EFFO members have received written invitations including the proposed agenda and the proposed budget for 1994.

President Werner Stehling opened the 7th general assembly by reporting on the EFFO activities of the last year. The number of members has grown in the last year: there are now more than fifty members from Austria, Denmark, France, Germany, Italy, the Netherlands, Russia, Switzerland and the United States of America. EFFO's reputation has also improved; this is revealed by the fact that more professional users such as companies, software developers and university institutions have applied for EFFO membership.

Two companies from Switzerland and Germany have decided to include the EFFO software order form to all products. This support is very helpful and has certainly contributed to the increase in software orders. EFFO hopes that other companies follow.

The interest in the EFFO software is evenly distributed among the available public domain disks. EFFO will continue to regularly update the disks and to add new disks whenever possible.

During the last year, computer networking and electronic mail has again become more important. The various EFFO activities would not have been possible without these media.

Formal topics of the agenda

The profit and loss account and the balance for the last year were presented, checked by the auditor and accepted unanimously by the audience. The elections of the officers were performed, and all nominees were elected with the maximum of votes possible. Hence the members of the committee for 1994 are:

President	Werner Stehling	(as before)
Vice-president	Reto Peter	(as before)
Secretary / Registrar	Reto Peter	(as before)
Treasurer	Stephan Paschedag	(new)
Auditor	Carsten Emde	(new)

EFFO members from Eastern Europe are eligible, for the time being, to a membership together with a subscription to OS-9 International free of charge. No special rules, however, apply for software orders from Eastern Europe. The prices for public domain disks and annual subscriptions for EFFO membership were approved and remain unaltered.

Various items from the agenda

The next issue of OS-9 International has been completed (you will probably figure out this yourself). The quality of the magazine is relatively high. It, therefore, requires a lot of time to prepare an issue, and it will take a great deal of work to maintain this high standard in future issues.

One major goal of EFFO for 1994 is to increase the number of public domain disks. Hubert Nehring has announced to produce a Micro-emacs disk. Avi Cohen Stuart from the Netherlands has offered to produce a public domain disk that contains various packing and compression utilities.

Currently, a certain backlog in copying and distributing public domain disks exists. All pending orders should have been shipped by the publication date of this issue of OS-9 International.

The standardisation of the documentation of PD software is still an unresolved item. Carsten Emde proposes to reserve one of the next EFFO meetings to select a common layout. It was decided to put this as main item on the agenda of the EFFO meeting in April.

A German VMEbus company has offered to provide, free of charge, a 68040-based computer board with high-resolution 4-bit graphics, and OS-9 and MGR development software on a removable hard disk. Initially, it was planned to use this computer as the EFFO mail server. But everybody agreed on that it would be a waste to use this system as a mail server. It was decided to use this system as a general EFFO OS-9 system for internal use. In addition, the system is available for a limited time to every EFFO member for special projects requiring features or the power of this system. Power supply and VMEbus crate are still needed, but will be made available by EFFO members.

An EFFO member offered a used 68020-based system free of charge. This would be an ideal computer for a mailbox system. EFFO gladly accepted this offer.

EFFO expresses its sincere gratitude to Eltec and to Hans-Werner Bippus for their generous offers.

The public relation activities in 1993 were not successful at all. None of the magazines published the supplied information about EFFO.

EFFO budget for 1994 in Swiss Francs

	1994 income	expense	1993 income	bf expense
subscription of 30 single members a 80.00	2400.00		2400.00	
subscription of 10 group members a 150.00	1500.00		1500.00	
yield as a result of PD distribution	2000.00		2000.00	
yield from sale of organizers for documentation	500.00		450.00	
carry forward from 1993	3583.66			
60 subscriptions to OS-9 International		900.00		1000.00
reserve fund OS-9 International		1000.00		
subscription to InterEUNet		240.00		500.00
running costs for mailing service		360.00		
500 disks for PD distribution		500.00		500.00
20 organizers for documentation a 20.00		400.00		600.00
hardware reserve fund		1000.00		1000.00
public relations		500.00		500.00
general assembly 1995		250.00		250.00
guests		150.00		150.00
postage		200.00		250.00
envelopds, paper, toner		400.00		
travelling expenses of editors		500.00		
unexpected expenses		500.00		500.00
carry forward to 1995		3083.66		1100.00
total	9983.66	9983.66	6350.00	6350.00

The next EFFO meetings

The next EFFO meetings will be held at the restaurant "Gasthof zur Herberge" in CH-5723 Teufenthal at 7 p.m. The meetings are scheduled as follows:

Fri. 06.05.94 (meeting)
Fri. 03.06.94 (meeting)
Fri. 01.07.94 (meeting)

Every person interested in EFFO or OS-9 is kindly invited to attend the meetings.

What PD software is most needed?

EFFO staff members not only collect and distribute OS-9 software. They also put a lot of effort in porting software from other platforms to OS-9. The GNU C compiler ported by Stephan Paschedag or the Ghostscript Postscript-language interpreter ported by Carsten Emde may be seen as examples.

Porting and preparing software involves a lot of work and is a time–consuming task. Therefore, EFFO would like to coordinate such activities and wants to know what kind of software is most needed by the OS-9 community.

We kindly invite you to state your personal preferences; please contact EFFO through the OS-9 International office by mail or fax. Of course, you can also send an E-Mail message to one of the EFFO board members.

Reto Peter can be reached through the EFFO office.

_getsys(OS-9 International);

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 our office for detailed information on how to place an ad in OS-9 International.

Subscriptions

OS-9 International is exclusively available by subscription. We currently offer a subscription of six issues at the following subscription fees:

	CH and FL	*Europe*	*Overseas*
Six issues	CHF 45.00	CHF 68.00	CHF 83.00

To subscribe to OS-9 International send a letter or postcard with your address to OS-9 International (see cover page for address).

We are sure that you understand that subscriptions from outside Switzerland have to be prepaid. This must be done in Swiss currency drawn to our bank account: Account # 10-107,666.0, Marc Balmer, Swiss Bank Corporation, CH-4000 Basel, Switzerland. Your subscription will start upon receipt of your payment.

Subscribers in CH and FL will receive a bill mailed together with the subsequent issue.

Code Disks

All code presented in this issue is available in electronic form on 3.5" OS-9 universal format disk. This disk may be obtained by sending CHF 20.00 to OS-9 International. Please specify the issue for which you want to receive the code disk.

OS-9 Archive Sites

The following archive sites have been checked to carry up-to date OS-9 software and information:

FTP	chestnut.cs.wisc.edu (formerly cabrales.cs.wisc.edu)
FTP	lucy.ifi.unibas.ch

New phone numbers

Please note that our phone and fax numbers have changed on April, 24. 1994. The new numbers are:

Phone	+41 61 381 55 01
Fax	+41 61 381 55 02

OS-9 International

ISSN: 1019-6714
Published by Marc Balmer
Editors: Marc Balmer, Carsten Emde