S.Ha.R.K. Frequently Asked Questions (FAQ)

Paolo Gai (pj@hartik.sssup.it)

4th June 2001

Abstract

This file contains a small list of questions that many people asked to me when they tried to use the Kernel. I hope they will be useful to you, too. I would also thanks Luca, Tomas, Mikael, Danilo, and Massy for their suggestions and their questions.

Paolo (pj@sssup.it)

Contents

1 OSLib FAQs

1.1 Abort 64

What does ABORT 64 mean? The behaviour of Abort 64 is not explained into the documentation; I hope it will be fixed with the next release...

You get abort 64 when:

1. OsLib programmed with the Periodic Timer

   - you are programming the real-time clock in a periodic way (you returned a number >0 when you registered the modules)
   - you left the interrupt disabled for a long time, > 2 times the period of the real time clock, into an interrupt handler...

   It happens when:

   (a) an interrupt arrives

   (b) the interrupts are disabled, and you handle your module stuffs that takes too long

   (c) another interrupt arrives

   (d) when the interrupt routine ends it discovers that another interrupt has arrived and it aborts (look at the file ll/kl/cxsw-1.s, after the label goon:)

   How to fix it:

   - try the one-shot timer (you can disable the interrupts up to about 50 ms)
   - enlarge the system tich that you returned into the initialization function
   - decrease the time you left interrupt disabled (maybe the scheduler is too complex???)

2. OsLib programmed with the one-shot timer

   - you are programming the real-time clock in a one-shot way
   - your computer is slower than mine :-)

   It happens when:

   (a) The one-shot timer, after handling an interrupt, have to be programmed to raise the next interruption. If the next interruption is too close to the current time, you have to program the interruption after a while, to allow the CPU to exit from the current interrupt routine.

   (b) If the CPU is too slow, this minimum time may not be enough, so:

       - the interruption is programmed at the minimum time - the interrupt fires before the CPU exits from the interrupt routine
       - when the interrupt routine ends it discovers that another interrupt has fired, and it aborts (as case 1)

   How to fix it:

   - look at the file include/ll/i386/pit.h
   - go to the inline function pit_setconstant
   - look at the code and find something like

     ```
     if (c < 100) {
         tconst = 100;
     } else {
         tconst = c;
     }
     ```

     increase the value "100", recompile and try again :-)

1.2 How many events can be queued?

The maximum number of queued events is contained in the `MAX_EVENT` declaration in `include/ll/sys/ll/event.h`.

1.3 Can an OSLib event interrupt a task?

Yes. The tasks are usually executed with interrupts enabled; the OSLib events are generated by the timer interrupt and always preempt the running task unless it has disabled the interrupts using `kern_cli()` and `kern_sti()`.

2 Kernel FAQs

2.1 Task Creation problems

2.1.1 Panic!!! can't create main task... errno=135

The error code 135 is set when there is a guarantee error. Maybe the guarantee algorithm of your new Module isn't correct...

The error codes are stored into these files:

- `include/bits/errno.h`
- `include/ll/errno.h` (LAST_STDERRNO is defined here)

2.2 printf & kern_printf

> As i remember i must use kern_printf instead of ordinary printf(), is that correct?

`kern_printf` is just a define on a similar function of the OSLib... -> you can use it when you have to put something on the screen, keeping attenction that the scroll of the screen may take several usec!!!

there is also all the standard C library (`printf` and so on), but I have not yet used it...(it was written by Massimiliano Giorgi)...In any case, into the modules you have to use the `kern_printf`...

If you want to use the standard C function `printf` you have to first init the filesystem (as done in the init file `kernel/init/initfs.c`) then you can use all the standard C functions, but keep attenction that `printf` is more more more more slower than the `kern_printf`!!!!

2.3 Exceptions

2.3.1 Exception 6

Exception 6, (see `include/bits/errno.h`), is the exception that is raised by the `kern_event_post` (`include/kernel/func.h`) when you posted more events than the OSLib can handle. Normally this exception should never happen.

The scheduling modules should use only a few events for each task. To have more events, you shall modify the macro `MAX_EVENT` in `include/ll/sys/ll/event.h`.

2.4 Modules

2.4.1 How to create a private function for the Module

Sometimes there is the need to add a primitive to handle some module-specific behavior (see the CBS Fault Tolerant). The safest way you can write an external function is:

```
void MYMODULE_externalfunc()
{
```

```
  PID p;
  MYMODULE_level_des *lev;
  kern_cli();
  p = exec_shadow;
  lev = (MYMODULE_level_des *)
    level_table[proc_table[p].task_level];
  MYMODULE_level_des *lev =
    (MYMODULE_level_des *)(level_table[l]);
  if (lev->l.level_code == EDF_LEVEL_CODE &&
    lev->l.level_version == EDF_LEVEL_VERSION)
  {
    // your stuffs using lev->...
  }
  else
  {
    // ouch...  wrong level!!!
  }
  kern_sti();
}
```

2.5 EDF Module and event_need_reschedule()

Why in the module kernel/modules/edf.c into the routine EDF_timer_deadline there is an event_need_reschedule, while into the function EDF_task_activate not, also if it post the same event and the code is very similar?

Answer:

- EDF_timer_deadline is an OSLib event and it needs to reschedule after its end (it changed the ready queue status).
- EDF_task_activate is a Task call called by the task_activate() primitive (see kernel/activate.c). At the end of the primitive the system is always rescheduled. Moreover, task_activate is NOT an OSLib event, and a call of event_need_reschedule should not be done into a Task Call.

3 Compiling

3.1 math_private.h

> When i try to make i get this error: its in the `ll\libm\msun\src` directory a lot of files complain that `math_private.h` doesn't exist but it does!!

Ok... this is a knew problem, due to the 8.3 file format of DOS.

The files provided in the .ZIP and .TGZ are NOT in the 8.3 format!!! they have long names, because:

- the code must be compilable under Linux

- changing the name to 8.3 format implies changing a lot of files that we have inherited from linux without any modification...

the solution of the problems are:

- compile under Linux :-)
- do the following steps :
  - erase your current version of shark
  - go in DOS mode (not the DOS prompt in a window of windows 95, you must not have the support for long names!!!)
  - decompress the zip file using unzip -a (now the file like math_private.h become like "math_pri.h"; note that if you have decompressed from a DOS prompt, the long name is maintained, and when you go in DOS mode the name is not "math_pri.h" but something like "math_p~1.h")
  - now you can compile, because when the GCC find something like `#include <math_private.h>` it will search for the math_pri.h!!!

3.2 Can S.Ha.R.K. run under Linux?

... when you compile an application, the compiler (either on Linux or MS-DOS) give you a Multiboot compliant image.

If you want to run such application, you have to use a multiboot compliant bootloader (like, for example, GRUB) or the dos eXtender X. You cannot run an application from Linux directly, unless you use VMware or programs like that. (note that VMware works only with release 0.1 of OSLib... I will upgrade the S.Ha.R.K. code soon...

If you try to execute an application from Linux, you simply get segmentation fault!

3.3 Architecture undefined

to be completed...

4 Init Files

4.1 What value should be returned for the TICK into the initfile?

> the TICK definition in the init file, what effect does it have? Is the system working in that granularity or?

If you specify a TICK >0, the system works with the given tick granularity.

If TICK = 0 the system works with a one-shot timer (i.e., the interrupts are raised only when needed)...

5 Applications

5.1 Example applications

Some simple applications are available into the demos directory. the examples directory currently contains some old tests that will be moved (with more comments into the source files) to the demos directory as soon as possible.

If you need to build your own application you can simply use the template applications available on the website (the file names of the archives are like myapp.zip and myapp2.zip in the source/zip directory). We prepared a small compile-howto that shows how to build a sample application with a custom module WITHOUT modifying the standard kernel distribution...

You can find it into the download page (there is a link called "template application" and another near that called "how to compile"...

5.2 How can I create a task at a specified time?

There are two ways to create a task at a specific time:

- generate an event that creates the task using `task_create` and activate it using `task_activate`
- create a task A that creates the task load when activated, and then activate the task A using an OSLib event.

The first method has the problem of accounting the time spent into the `task_create` (the time spent in the event is accounted to the preempted running task, possibly causing a deadline miss or a wcet violation). In fact, note that if you create a task directly into an OSLib event the accounting of the elapsed time is done at reschedule time... so:

- a task executes
  - ->interrupt
    - the OSLib handles the event
      - the event calls the `task_create` is called (but the task accounting information is not done yet!!!)
      - the event activates the new task (the activation calls also `event_need_reschedule`)
    - ...other OSLib events may be handled
  - and NOW the system is rescheduled, and the wcet are updated!!! (This maybe a problem since the creation time depends on the guarantee algorithm)
  - ...then the interrupt routine finish
- ...and a task (maybe different from the previous preempted task) return to execution...

If the wcet accounting is a problem (`task_create` was not thought to be called in an event...) you can do that (I think THIS is the better solution):

- first, you have to register another level before yours, and create a task in that level: so, for example, you register
  - At level 0 you put RR, or EDF (if you want some form of guarantee). This level will handle the creation task A.
  - Your scheduling modules...
  - RR (for the main).
  - dummy.
- then, you create a task in the level 0 that simply doing that:

  ```
  void *taskXXX(void *arg)
  {
    for(;;) {
      task_create(mydisturbtask)
      task_activate(mydisturbtask)
      task_endcycle()
    }
  }
  ```
- Finally, you activate `taskXXX` in the OSLib event.

In this way, the capacity are managed correctly, and you can look at the overhead of the creation looking at the JET data of `taskXXX`.

5.3 How can I activate a lot of tasks together?

What can be done to activate 2, 10 or 100 tasks that arrives at the same time? You can do that in many ways...

- you may write something like

  ```
  task_activate(task1)
  ...
  task_activate(taskn)
  ```

  this works only if the running task has priority > than the others activated task (the first task, once activate, preempt the others!!!
- to fix this situation, you may write this

  ```
  task_nopreempt()
  task_activate(task1)
  ```

```
...
task_activate(taskn)
task_preempt()
```

task preempt and task_nopreempt simply disable the preemption in the system... (you may use it also for resource management... look at the NPP mutex module)

- another way is to give a group number to all the n task to be activated and then write something like that:

  `group_activate(group_number)`

  and this primitive do the second solution in an atomic and faster way...

- If you have also to create a group of tasks, look at the task group creation howto in the download page of the S.Ha.R.K. website!

5.4 How can the exception handlers be redefined to exit correctly from the Graphic mode?

5.4.1 If you only need to print a message...

Since 4 jun 2001 patch, there are two functions available, called
`int set_exchandler_grx();`
`int set_exchandler_text();`
these functions can be used to set up a handler for the `SIGHEXC` signal that prints a message with a little description of the exception numbers.[1]

You can call these functions in any point of your application. For example, look at `demos/base/fly.c`

5.4.2 If you need to redefine the S.Ha.R.K. exception handler

To correctly exit from the graphical mode, remember that an application can exit using either an exception (a signal handler is called) or a normal exit (for example, a `sys_end`()).

Remember that the `grx_close`() clean the video text memory, so if you want to print a debug information on the exception handler, you should write a signal handler that terminates with `sys_abort`(), and post a `grx_close`() with the flags `RUNLEVEL_BEFORE_EXIT` because the grx library runs in protected mode and not in real mode, and `NO_AT_ABORT` because in that way if you exit with an exception the `grx_close`() is put into the exception handlers and then, when `sys_abort` is called, the final `grx_close` is not called (if called, it cleans again the screen erasing our nice exception printings!!!).

You can write something like that:

[1] Exceprion numbers are listed into `include/bits/errno.h` .
The two functions simply redefines the `SIGHEXC` using the `sigaction` primitive. `set_exchandler_grx` differs from `set_exchandler_text` because it prints the message using a function posted with `sys_atrunlevel` with parameter `RUNLEVEL_AFTER_EXIT`.

```
void demo_exc_handler(int signo, siginfo_t *info, void *extra)
{
  struct timespec t;

  grx_close();

  /* Default action for an kern exception is */
  kern_cli();

  /* some nice silly printings */
  kern_printf("Goofy said:  Yuk!!!");

  sys_abort();
}

void my_close(void *arg)
{
  grx_close();
}

int main(int argc, char **argv) {

  ...

  struct sigaction action;

  action.sa_flags = SA_SIGINFO;
  action.sa_sigaction = demo_exc_handler;
  action.sa_handler = 0;
  sigfillset(&action.sa_mask);

  /* we block all the other signals...  */
  if (sigaction(SIGHEXC, &action, NULL) == -1)
  {
    perror("Error initializing signals...");
    sys_abort();
  }

  sys_atrunlevel(my_close, NULL, RUNLEVEL_BEFORE_EXIT|NO_AT_ABORT);
  ...
}
```

5.5 The MPEG demo

The MPEG demo was developed on my PC (celeron 366), and it should work :-). These are the steps you should do:

- compile the `example/fsdemo` directory.
- you need:
  - a fat16 filesystem
  - a few mpeg and bmp files (if you want the original file we used in the demo, send a mail to pj@sssup.it). You have to put the files here:
    - /temp/b/ph2a.bmp
    - /temp/b/ph3a.bmp
    - /temp/b/ph7a.bmp
    - /temp/m/earth-ci.mpg
- then you have to compile the demo using the provided makefile
- and run it using "x mplay"
- hit the enter key, then the keys a s d to load the bmp images.

5.6 How I can read and write from a FAT32 filesystem?

The filesystem currently supports only FAT16 filesystems on IDE drivers.

If you need to simply read/write some infos at the beginning or at the end of the application, you can use some hooks to DOS INT21h functions.

These functions are listed into `include/ll/i386/x-dos.h`, and allow the user to read and write a binary file. For more informations, see `demos/dosfs/README`

5.7 How can I use DOS_fwrite and DOS_fread?

See FAQ 5.6.