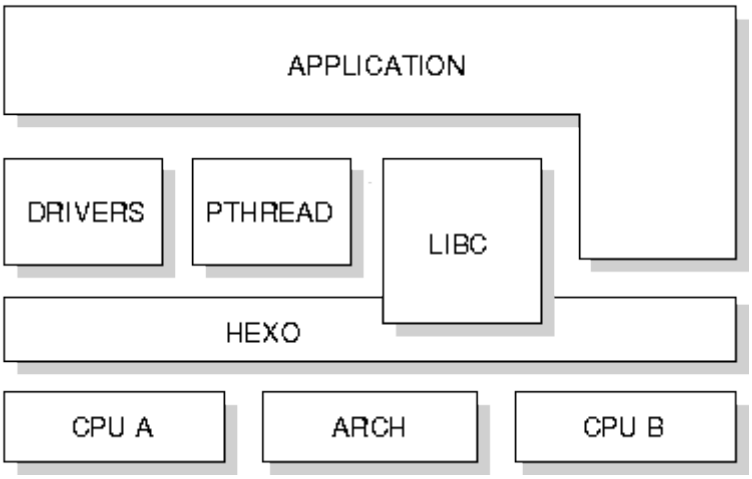
MutekH as Unix process quick start guide

This guide explain how to run MutekH embedded in a Unix process (so called emu architecture) in a way similar to ?user mode linux. This is the simplest way to run MutekH as it doesn't require any hardware platform. It will work if you running a Linux or Darwin host kernel on x86 or x86_64 processor(s).

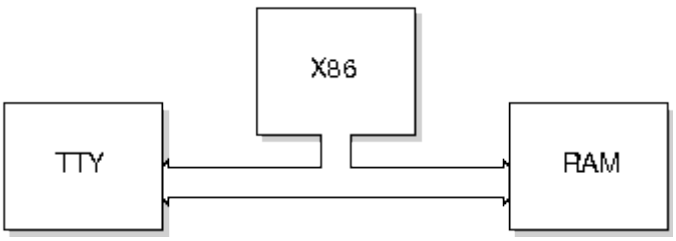
Overview

The MutekH operation system is built on top of the Hexo hardware abstraction layer. It's composed of several modules and libraries.



When compiled to run embedded in a Unix process, minimal hardware ressource are needed: The host processor and unix process memory are used as execution platform and a simple TTY driver redirect output to the Unix terminal.

It enables running MutekH natively on the host processor. This configuration suffer from several limitations regarding available peripherals, but it is usefull to test and debug algorithms.



Getting the sources

The example below show how to run MutekH using this configuration, it only requires to get the MutekH source code.

The MutekH source code is fully configurable and can be tweaked to adapt hardware platform and application needs. Configuration is handled by a dedicated tool which check dependencies and other relationships between the large set of available configuration tokens.

```
svn co https://www.mutekh.org/svn/trunk/mutekh/
```

Source tree is organized this way:

```
mutekh
|-- arch            contains hardware platforms modules for hexo
|-- cpu             contains processors modules for hexo
|-- doc             documentation
|-- drivers         device and filesystem drivers
|-- examples        Test and example programs
|-- gpct            container library, available as a separate project
|-- hexo            Hexo hardware abstraction layer
|-- mutek           hardware independant kernel code
|-- libc            standard C library
|-- libm            standard math library
|-- libnetwork      netwotk stack
|-- libpthread      posix thread library
|-- libvfs          virtual File System
...
|-- scripts         build system scripts
`-- tools           some usefull tools
```

More directories are actually available with other libraries and features.

Writing the example source code

Note: This example is available directly from `examples/hello` directory in source tree: trunk/mutekh/examples/hello

- Creating a new modules directory

```
mkdir hello
cd hello
```

- Writing the source code in `hello.c`

```
#include <pthread.h>

pthread_mutex_t m;
pthread_t a, b;

void *f(void *param)
{
  while (1)
    {
      pthread_mutex_lock(&m);
      printf("(%i) %s", cpu_id(), param);
      pthread_mutex_unlock(&m);
      pthread_yield();
    }
}
int main()
{
  pthread_mutex_init(&m, NULL);
  pthread_create(&a, NULL, f, "Hello ");
  pthread_create(&b, NULL, f, "World\n");
}
```

- Writing the `Makefile`

```
objs = hello.o
```

Writing the MutekH configuration

Standalone configuration file

Our configuration file is named `hello/config_emu`. Details about configuration file is explained later. This configuration file describe the following things:

- The application license, used to check license consistency for modules in use,
- The target hardware platform and processor
- Use of the POSIX threads library
- Use of terminal output
- Declaration of a new "hello" modules

The MutekH source code is split in modules. We now have to declare our new module to have it compiled along with the kernel by the build system. As modules may be located out of the source tree, we have to specify the module directory.

```
# New source code module to be compiled
  CONFIG_MODULES examples/hello:%CONFIGPATH

# Application license
  CONFIG_LICENSE_APP_LGPL

# Platform types
  CONFIG_ARCH_EMU
  CONFIG_ARCH_EMU_LINUX

# Processor types
  CONFIG_CPU_X86_EMU

  ...
```

The complete configuration file is available here: trunk/mutekh/examples/hello/config_emu.

Have a look to the BuildSystem page for more information about configuration system and configuration file format. The MutekH API reference manual describes all available configuration tokens.

Generic configuration file

The flat and standalone configuration file described above is specific to the target emu architecture, GNU/Linux host operating system and x86 32bits processor.

It's possible to write a more generic configuration file which relies on common files to target more platforms.

The complete generic configuration file is available here: trunk/mutekh/examples/hello/config.

Compiling the application along with MutekH

Getting the cross-compilers

You may have some success in compiling MutekH/emu using your readily available host GNU compiler. If it doesn't work because of incompatible configuration, please use a MutekH toolchain. You can rely on the

`tools/crossgen.mk` script which comes along with MutekH to build some GNU toolchains or download a precompiled toolchain. See BuildingExamples page.

Compiling

Simply type something like:

```
make CONF=examples/hello/config_emu
```

or to use the generic configuration file:

```
make CONF=examples/hello/config BUILD=emu-linux-x86
```

The later allows targeting other emu platforms like MacOs X (darwin) and x86_64 processors and other hardware platforms. See BuildingExamples for details.

Once the compilation process has finished, the executable binary is available.

Execution

Simply execute the program as a normal unix executable:

```
$ ./kernel-emu-x86-emu.out
(0) Hello (0) World
(0) Hello (0) World
(0) Hello (0) World
(0) Hello (0) World
...
```

Other more advanced topics and guides are available from the Main page.