# MutekH quick start guide for SoCLib platform

This quickstart guide only present 2 target plaforms among those supported by MutekH. These platforms are linux/darwin user process and the SoCLib hardware simulator. Other supported platforms are multiprocessors IBMPC/x86 and some micro-controllers.

MutekH can be compiled for Mips, Arm, PowerPC, x86 and Avr porcessors.

# MutekH configurations

The MutekH kernel 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.

This document present two source code configurations:

1. A first MutekH build designed to run embedded in a unix process.

   

   This is the simplest way to have MutekH running as it does not need extra hardware or simulator. It enables running MutekH natively on the host processor. This configuration suffer from several limitations regarding available peripheral, but it is usefull to test and debug algorithms.

   

   The first example below show how to run MutekH using this configuration.

2. The second example below explain how to setup a SoCLib hardware simulator with 4 RISC processor (Mips, Arm or PowerPC).

   

# Part 1 : Running MutekH in a UNIX process

This first example only require to get the MutekH source code.

## Getting the sources

```
svn co -r 1024 https://www-asim.lip6.fr/svn/mutekh/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
|-- libc            standard C library
|-- libm            standard math library
|-- libnetwork      netwotk stack
|-- libpthread      posix thread library
|-- libvfs          virtual File System
|-- mutek           hardware independant kernel code
|-- 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` in source tree.

- 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 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.

```
# Application license
  CONFIG_LICENSE_APP_LGPL

# Platform types
  CONFIG_ARCH_EMU

# Processor types
  CONFIG_CPU_X86_EMU

# Mutek features
  CONFIG_PTHREAD
  CONFIG_MUTEK_CONSOLE

# Device drivers
  CONFIG_DRIVER_CHAR_EMUTTY

# Code compilation options
  CONFIG_COMPILE_DEBUG

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

## Compiling the application along with MutekH

Simply type:

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

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

```
kernel-emu-x86-emu.out
```

## 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
...
```

# Part2 : Running MutekH in a multiprocessor SoCLib simulator

## Getting SoCLib

We now need to have a working SoCLib install. SoCLib installation is explained here: ?https://www.soclib.fr/trac/dev/wiki/InstallationNotes

# SoCLib platform description

The SoCLib source tree contains a platform dedicated to this tutorial:
`soclib/soclib/platform/topcells/caba-vgmn-mutekh_tutorial/`.

# Getting the cross-compilers

You can rely on the `tools/crossgen.mk` script which comes along with MutekH to build some GNU cross-toolchains:

```
$ tools/crossgen.mk
$ tools/crossgen.mk all TARGET=mipsel-unknown-elf
```

# MutekH Configuration

Note: This example is readily available in the `examples/hello` directory in the MutekH source tree.

The MutekH configuration for the 4 Mips processors platform is in the `hello/config_soclib_mipsel` file:

```
# Application license
  CONFIG_LICENSE_APP_LGPL

# Platform types
  CONFIG_ARCH_SOCLIB

# Processor types, Mips little endian with multiprocessor support up to 4 cpus
  CONFIG_CPU_MIPS
  CONFIG_CPU_MIPS_VERSION 32
  CONFIG_CPU_ENDIAN_LITTLE
  CONFIG_SMP
  CONFIG_CPU_MAXCOUNT 4

  CONFIG_CPU_RESET_HANDLER

# Mutek features
  CONFIG_PTHREAD
  CONFIG_MUTEK_CONSOLE

# Device drivers
  CONFIG_DRIVER_CHAR_SOCLIBTTY
  CONFIG_DRIVER_ICU_SOCLIB
  CONFIG_ARCH_DEVICE_TREE

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

# definitions of the memory sections where to put things
  CONFIG_ROM_ADDR 0x60100000
  CONFIG_ROM_SIZE 0x00100000

  CONFIG_RAM_ADDR 0x62600000
  CONFIG_RAM_SIZE 0x00100000

# Add an hardware enumerator
  CONFIG_FDT
  CONFIG_DRIVER_ENUM_FDT
```

You may have noticed the processor definition change: we are now building for a 4 little-endian Mips processors platform.

Have a look to the BuildSystem page for more information about configuration system.

# Platform description

This hardware platform uses now hardware enumeration (plug and play), so the `CONFIG_ARCH_DEVICE_TREE` token is defined in the configuration file. It will let the kernel get the platform layout description from a FlattenedDeviceTree which will be built-in.

Therefore, we have to provide the platform description FlattenedDeviceTree and add it to the Makefile to have it compiled in. The `hello/Makefile` file must contain:

```
objs = hello.o platform-mips.o
```

The current FlattenedDeviceTree source file `platform-mips.dts` contains:

```
/dts-v1/;

/ {
        model = "MutekH_Tutorial";
        compatible = "MutekH_Tutorial";
        #address-cells = <1>;
        #size-cells = <1>;

        cpus {
                #address-cells = <1>;
                #size-cells = <0>;
                Mips,32@0 {
                        name = "Mips,32";
                        device_type = "cpu";
                        reg = <0>;
                        icudev_type = "cpu:mips";
                };

                Mips,32@1 {
                        name = "Mips,32";
                        device_type = "cpu";
                        reg = <1>;
                        icudev_type = "cpu:mips";
                };

                Mips,32@2 {
                        name = "Mips,32";
                        device_type = "cpu";
                        reg = <2>;
                        icudev_type = "cpu:mips";
                };

                Mips,32@3 {
                        name = "Mips,32";
                        device_type = "cpu";
                        reg = <3>;
                        icudev_type = "cpu:mips";
                };

        };

        tty@0 {
            device_type = "soclib:tty";
                tty_count = <1>;
                reg = <0x90600000 0x10>;
                icudev = &{/icu@0};
                irq = <0>;
```

```
            };

            icu@0 {
                device_type = "soclib:icu";
                    input_count = <2>;
                    reg = <0x20600000 0x20>;
                    icudev = &{/cpus/Mips,32@0};
                    irq = <0>;
            };

            timer@0 {
                device_type = "soclib:timer";
                    reg = <0x01620000 0x10>;
                    icudev = &{/icu@0};
                    irq = <1>;
            };

            memory@0 {
                    device_type = "memory";
                    cached;
                    memreg: reg = <0x61100000 0x00100000>;
            };

            memory@1 {
                    device_type = "memory";
                    memreg: reg = <0x62600000 0x00100000>;
            };

            chosen {
                    console = &{/tty@0};
                    timer = &{/timer@0};
            };

    };
```

# Compiling the application along with MutekH

The MutekH kernel and the application may be built out of the source tree.

Change to the SoCLib platform directory and apply the following steps to experiment with out of tree compilation. You have to setup the following variables:

`MUTEKH_DIR`
: Path to MutekH source tree

`APP`
: Path to application source

`CONFIG`
: MutekH configuration file name

```
$ cd soclib/soclib/platform/topcells/caba-vgmn-mutekh_tutorial
$ make MUTEKH_DIR=~/mutekh/ APP=~/mutekh/examples/hello CONFIG=config_soclib_mipsel all
```

This will build the MutekH kernel along with the application. You can still build MutekH separately as explained in the first part. The simulator can then be built using:

```
$ cd soclib/soclib/platform/topcells/caba-vgmn-mutekh_tutorial
$ make system.x
```

# Execution

The simulator needs the MutekH executable file name and the processor type and the number of processors of this type:

```
$ ./system.x mutekh/kernel-soclib-mips.out:mips32:4
```

You may want to refer to other articles available from the main page to go further with MutekH.