

# MutekH quick start guide for SoCLib platform

---

Dans le cadre du projet SoCLib/ANR seules ces deux types de configuration sont documentées. D'autres configurations sont possibles dont, par exemple, celle adaptée aux cartes mères de PC/x86. Elles ne sont pas documentées ici, mais sont présentes dans le dépôt svn de MutekH.



MutekH est compilable sur plusieurs processeurs de la bibliothèque SoCLib comme le Mips et le PowerPC. Il est également compilable sur intel X86 en natif ou en émulation dans un processus Unix.

## Les configurations de MutekH

MutekH est entièrement configurable pour s'adapter à la fois au matériel sur lequel il s'exécute mais aussi aux besoins de l'application. Il existe une centaine de paramètres permettant de définir une configuration. Chaque paramètre peut prendre plusieurs valeurs. Les paramètres ne sont pas indépendants les uns vis à vis des autres, deux paramètres peuvent s'exclure ou dépendre l'un de l'autre. MutekH facilite la définition d'une configuration en vérifiant les règles de dépendances et la cohérence des valeurs.

Dans ce document nous allons voir deux configurations.

1. Une configuration pour une plateforme x86 émulée dans un processus unix.



L'intérêt de cette plateforme est qu'elle permet de mettre au point une application en bénéficiant des performances de la machine hôte. Elle permet également d'exécuter une application sans que l'installation de SoCLib ne soit nécessaire. L'inconvénient est que le seul périphérique disponible est le terminal.



Les premières applications de ce document utilisent cette configuration. Au delà de son aspect jouet, cette configuration est utile à chaque fois que l'on s'intéresse à la mise au point d'un algorithme et que l'on a pas besoin de coprocesseurs, ni d'interruption.

2. Une configuration pour un cluster à 4 processeurs (Mips ou PowerPC) autour d'un NoC/VCI. Cette configuration permet de créer une application pour un simulateur de SoC en systemC/SoCLib.  
DESSIN DE LA PLATEFORME: PROC+RAM+LOCK+MULTITTY+...

## Premiers pas sur une plateforme émulée

Pour ce premier essai seules les sources de MutekH sont nécessaires.

## Récupération des sources

svn co -r 471 [?https://www-asim.lip6.fr/svn/mutekh/trunk/mutekh](https://www-asim.lip6.fr/svn/mutekh/trunk/mutekh)

créé un répertoire mutekh contenant les sources et les documentations de MutekH. L'organisation des sources suit celle des bibliothèques.

```
mutekh
|-- arch           code dépendant des plateformes
|-- cpu           code dépendant des processeurs
|-- doc           documentation
|-- drivers       pilotes des périphériques
|-- gpct          API de gestion des ensembles
|-- hexo          API d'Hexo
```

```

|-- libc           API standard C
|-- libnetwork    API de la pile de protocole réseau
|-- libpthread    API des threads Posix
|-- libssl        API des appels système Unix
|-- libvifs       API Virtual File System
|-- mutekh
|-- scripts
`-- tools

```

## Écriture du premier programme

Le programme de test va être placé dans un sous-répertoire du répertoire de mutekh.

### 1. Création du répertoire de test, dans le répertoire mutekh

```

mkdir benches benches/hello
cd benches/hello

```

### 2. Écriture du programme suivant dans le fichier hello.c

```

#include <pthread.h>

pthread_mutex_t m;
pthread_t a, b;

void *f(void *param)
{
    while (1)
    {
        pthread_mutex_lock(&m);
        printf("(%) %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, "Wolrd\n");
}

```

## Configuration de Mutekh

Tapez le fichier de configuration suivant dans le fichier benches/config\_emu Nous verrons plus loin la signification des configurations. Disons rapidement que cette configuration indique:

- la licence de l'application (et la vérification que les composants sont compatibles avec ce choix),
- que la plateforme cible s'exécute dans un processus Unix sur une plateforme X86
- que l'application utilise les Pthreads.
- que les affichages se font sur le terminal.

```

# 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
```

## 4. Compilation de l'application et de MutekH

La compilation se fait en tapant:

```
make CONF=benchs/config_emu LIBAPP=benchs/hello/hello.o
```

Le Makefile compile les sources du système et de l'application en tenant compte du fichier de configuration. Le résultat de cette compilation est:

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

## 5. Exécution

L'exécution du programme kernel-emu-x86-emu.out rend normalement: {{{(0) Hello (0) World (0) Hello (0) World (0) Hello (0) World (0) Hello (0) World ... }}}}

## Plus loin sur une plateforme SoCLib

### Récupération des sources de SoCLib

renvoie au site de soclib à la page d'installation

### Description de la plateforme SoCLib

Pour ce premier contact avec MutekH, nous avons préparé une plateforme à 4 processeurs. Cette plateforme se trouve dans les sources de SoCLib dans le répertoire soclib/soclib/platform/topcells/mutekh\_basic/.



## Configuration de MutekH

La configuration de MutekH pour une plateforme simulé de 4 processeurs va être placée dans le fichier benches/config\_mips:

```
# 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_ENDIAN_LITTLE
```

```

CONFIG_SMP
CONFIG_CPU_MAXCOUNT 4

# Mutek features
CONFIG_PTHREAD
CONFIG_MUTEK_CONSOLE

# Device drivers
CONFIG_DRIVER_CHAR_SOCLIBTTY
CONFIG_DRIVER_ICU_SOCLIB

```

On note que la définition des processeurs changent. On indique qu'il s'agit de 4 mips de type bigendian. Le système peut donner des informations sur les options en tapant la commande:

```
make CONF=benchs/config_mips showconfig TOKEN=CONFIG_SMP
```

L'ensemble des configurations possibles (une centaine) peut être obtenu par la commande ci-après:

```
make CONF=benchs/config_mips listconfig
```

Le résultat:

Configuration token name	Declare location
CONFIG_ARCH_EMU	(arch/emu/emu.config:2)
CONFIG_ARCH_IBMPC	(arch/ibmpc/ibmpc.config:2)
CONFIG_ARCH_SIMPLE	(arch/simple/simple.config:2)
+ CONFIG_ARCH_SOCLIB	(arch/soclib/soclib.config:2)
CONFIG_CLUSTER	(arch/arch.config:8)
m CONFIG_CPU_MAXCOUNT	(cpu/cpu.config:2)
+ CONFIG_CPU_MIPS	(cpu/mips/mips.config:2)
CONFIG_CPU_MIPS_ABI_EABI	(cpu/mips/mips.config:40)
CONFIG_CPU_MIPS_ABI_N32	(cpu/mips/mips.config:34)
+ CONFIG_CPU_MIPS_ABI_O32	(cpu/mips/mips.config:22)
CONFIG_CPU_MIPS_ABI_O64	(cpu/mips/mips.config:28)
v CONFIG_CPU_MIPS_VERSION	(cpu/mips/mips.config:16)
...	
(+) defined, (p) provided, (m) mandatory, (v) value.	

## Compilation de l'application et de MutekH

La plateforme mutek\_basic s'adapte automatiquement à la configuration des sources de MutekH (type de cpu, ...). Il faut donc ajouter un lien symbolique dans le répertoire de la plateforme soclib mutek\_basic vers les sources de MutekH.

```
cd ~/soclib/soclib/platform/topcells/mutekh_basic
ln -s ~/mutekh
```

Par ailleurs, Mutekh doit connaître les adresses liées au matériel de la plateforme. Il faut donc ajouter des liens symboliques dans le répertoire de mutekh vers la description des adresses de la plateforme:

```
cd ~/mutekh
ln -s ~/soclib/soclib/platform/topcells/mutekh_basic/soclib_addresses.h
ln -s ~/soclib/soclib/platform/topcells/mutekh_basic/soclib_addresses.ldscript.m4
```

On commence par compiler le système MutekH avant de compiler la plateforme. La compilation de MutekH impose que les cross tools suivants soient disponibles:

- mipsel-unknown-elf-gcc

- mipsel-unknown-elf-ld

```
make CONF=benchs/config_mips LIBAPP=benchs/hello/hello.o
```

Lancer la compilation de la plateforme:

```
cd soclib/soclib/platform/topcells/mutekh_basic  
make
```

## Exécution

```
./system.x ~/mutekh/kernel-soclib-mips.out
```