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 présentent dans le dépot 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 configuraion. Chaque paramètre peut prendre plusieurs valeurs. Les paramètres ne sont pas indépandants 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 cohérences 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.

Premiers pas sur une plateforme émulée

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

Récuperation des sources

```
svn co -r 997 https://www-asim.lip6.fr/svn/mutekh/trunk/mutekh
```

crée 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
|-- examples        Test and example programs
|-- 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
|-- libsrl
|-- libunix         API des appels système Unix
|-- libvfs          API Virtual File System
|-- mutek
|-- scripts
`-- tools
```

Ecriture du premier programme

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

Note: Cet exemple est disponible directement dans le sous repertoire `examples/hello`.

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

   ```
   mkdir hello
   cd hello
   ```
2. Ecriture 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("(%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");
   }
   ```
3. Ecriture d'un `Makefile`

   ```
   objs = hello.o
   ```

Configuration de MutekH

Tapez le fichier de configuration suivant dans le fichier `hello/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.
- que l'on déclare un nouveau module nommé "hello".

Le code source de MutekH est découpé en modules. Nous devons declarer un noveau module que nous appelerons "hello" pour que notre application soit compilée en même temps que le reste du système. Le chemin de ce module doit être spécifié, dans notre cas ce chemin est identique à celui du fichier de configuration que nous écrivons.

```
# 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
```

4. Compilation de l'application et de MutekH

La compilation se fait depuis le répertoire de MutekH en tapant:

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

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 programmme kernel-emu-x86-emu.out rend normalement:

```
(0) Hello (0) World
(0) Hello (0) World
(0) Hello (0) World
(0) Hello (0) World
...
```

Aller plus loin avec une plateforme SoCLib

Récupération des sources de SoCLib

Il faut disposer d'une installation fonctionnelle de SoCLib, on vous renvoie au site de soclib à la page d'installation: ?https://www.soclib.fr/trac/dev/wiki/InstallationNotes

Description de la plafeforme SoCLib

Pour ce premier contact avec MutekH, nous avons préparé un plateforme à 4 processeurs. Cette plateforme se trouve dans les plateformes de démo de SoCLib dans le répertoire `soclib/soclib/platform/topcells/caba-vgmn-mutekh_tutorial/`.

Obtenir les compilateurs criosés

Il est possible d'installer facilement les outils de compilation pour les différents processeurs en utilisant le script `tools/crossgen.mk` fourni avec mutek:

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

Configuration de MutekH

Note: Cet exemple est disponible directement dans le sous repertoire `examples/hello`.

La configuration de MutekH pour une plateforme simulée de 4 processeurs va etre placée dans le fichier `hello/config_soclib_mipsel`:

```
# 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
```

On note que la définition des processeurs change. On indique qu'il s'agit de 4 mips de type little-endian. Le systeme peut donner des informations sur les options en tapant la commande:

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

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

```
make CONF=hello/config_mips listconfig
```

Voir BuildSystem pour plus d'informations

Description de la plateforme

On voit qu'on a ajouté un token `CONFIG_ARCH_DEVICE_TREE` dans le fichier de configuration, celui-ci sert à dire qu'on a une définition de la plateforme sous forme d'un FlattenedDeviceTree qui accompagne le kernel.

Il faut en fait compiler cette description dans le kernel, en l'ajoutant dans la Makefile. Le fichier `hello/Makefile` contient alors:

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

Il faut ensuite ajouter un fichier `platform-mips.dts`, au format accepté par l'utilitaire `dtc`, contenant la définition de la plateforme. Ce format est issu de l'IEEE1275 (Open Firmware).

```
/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};
        };

};
```

Compilation de l'application et de MutekH

La plateforme mutek_tutorial s'adapte automatique à la configuration des sources de MutekH (type de cpu, ...). De plus, on peut compiler le noyau depuis le répertoire de la plateforme, donc en dehors de l'arborescence de MutekH.

La Makefile fournie contient tout le code nécessaire. Elle a besoin de 3 variables:

`MUTEKH_DIR`
: Le répertoire contenant les sources de MutekH

`APP`

Le répertoire contenant les sources de l'application (ici le répertoire de hello)

`CONFIG`
: Le fichier de configuration, dans le répertoire de la plateforme

La compilation de MutekH impose que les cross tools suivants soient disponibles:

- mipsel-unknown-elf-gcc
- mipsel-unknown-elf-ld
- ?

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

Cette commande compile successivement le noyau, puis la plateforme.

Exécution

Le simulateur prend en argument le kernel à charger ainsi que le type et le nombre de processeurs. Dans notre cas, il est dans `mutekh/kernel-soclib-mips.out` et s'exécute sur une plateforme avec 4 processeurs mips32:

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