HAL - CPU
part 1

MI074 - 4

HAL : Hardware Abstraction Layer

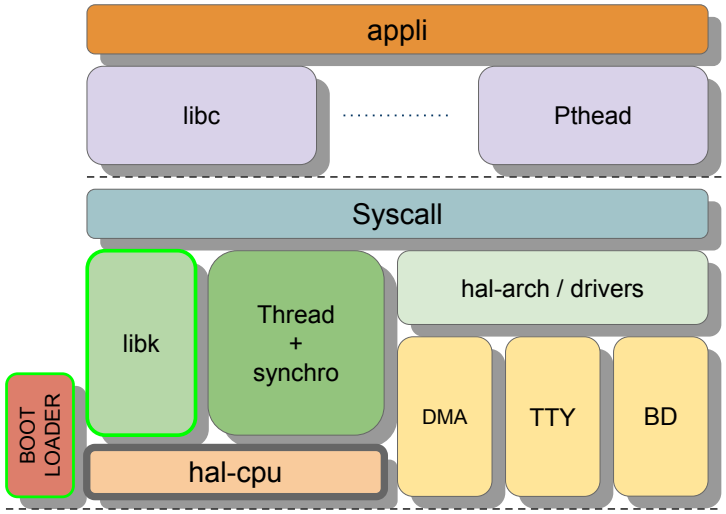
(selon wikipedia)

La couche d'abstraction matérielle (Hardware Abstraction Layer ou HAL) est une spécification et un utilitaire logiciel qui *traque* les périphériques du système informatique.

Le but du HAL est d'éviter aux développeurs d'implémenter manuellement le code spécifique à un périphérique. À la place, ils peuvent utiliser une couche connectable qui fournit des informations à propos du dit périphérique, tel que cela se passe par exemple lorsqu'un utilisateur branche ou débranche un périphérique USB.

Cette couche implémente un certain nombre de fonctions spécifiques au matériel : interfaces d'entrées-sorties, contrôleur d'interruptions, caches matériels, mécanismes de communication multiprocesseur... Elle isole ainsi le noyau du système des spécificités des plates-formes matérielles.

Architecture de l'OS final



HAL : Hardware Abstraction Layer

Une HAL est définie par un ensemble de structures de données et de fonctions qui permettent au noyau d'accéder au matériel de manière uniforme indépendamment du matériel réel.

La HAL est défini à travers plusieurs interfaces

- CPU Interface Kernel avec le CPU
- ARCH Interface Kernel avec la plateforme et les périphériques

Nous commencer par l'interface CPU et partiellement avec l'ARCHitecture (sans les drivers).

Organisation du code du système

```
|-- arch          HAL architecture
|   `-- soclib      spécifique à la plateforme (p.ex. segmention.h)
|                     et les drivers des périphériques
|
|-- cpu           HAL processeur
|   `-- mipsel      spécifique au mips (p.ex. accès registres)
|
|-- kernel        code du noyau
|
`-- libk          bibliothèque C pour le noyau
                  str..., malloc, etc.
```

Notes: dans linux

- **arch** contient le code spécifique au CPU
- **drivers** contient le code de la plateforme et les drivers
- **mm** contient le gestionnaire de mémoire du noyau

HAL : CPU et ARCH

- L'interface de la HAL cpu et arch est présente dans deux .h dans le répertoire kernel.
- Puisque la plupart des services sont courts, ils sont pour la plupart décrits dans des fonctions inlinées, donc dans un .h et pas dans .c
- dans kernel, on a :
  hal_cpu.h et hal_arch.h
  qui inclue le fichier présent dans cpu/mipsel
  hal_cpu_code.h
  et le fichier présent dans arch/soclib
  hal_arch_code.h
- C'est dans le Makefile en fonction des paramètres CPU et ARCH que l'on indique quel hal_cpu_code.h ou hal_arch_code inclure.

HAL-ARCH

Accès minimaliste à la console
Pour pouvoir afficher des messages alors que le système n'est pas installé

Opérations atomiques et lock
Utilise le service offert par le processeur s'il existe
ou un périphérique
ou ...

Declaration des pilotes
Déclare tous les types de péripériques que sait gérer l'OS

Initialisation de l'architecture
initialisation des pilotes de périphérique (vu plus tard)

HAL-ARCH

```
#ifndef _HAL_ARCH_H_
#define _HAL_ARCH_H_

// ---------------------------------------- minimalist kernel console access
static int  kgetc    (void);
static int  kputc    (int c);

// ---------------------------------------- spinlock API
typedef unsigned spinlock_t;

static inline void spin_init         (spinlock_t *lock);
static inline void spin_destroy      (spinlock_t *lock);
static inline int  spin_trylock      (spinlock_t *lock);
static inline void spin_lock         (spinlock_t *lock);
static inline void spin_unlock       (spinlock_t *lock);
static inline void spin_lock_noirq   (spinlock_t *lock, unsigned * status);
static inline void spin_unlock_noirq (spinlock_t *lock, unsigned status);

#include <hal_arch_code.h>

#endif
```

Il manque la déclaration des périphériques.

HAL-CPU

Registres spéciaux
numéro de proc, timestamp

Irq du CPU
masquage, démasquage.

Opérations atomiques et lock
addition, trylock, lock, unlock, ...

Caches
invalidation de ligne de cache data

Contexte du processeur
création, destruction, chargement, sauvegarde.

HAL-CPU

```
#ifndef  _HAL_CPU_H_
#define  _HAL_CPU_H_

// ------------------------------------------------------------ Special Register
#define CPUID cpu_get_id()
#define CPUTIME cpu_time_stamp()
static inline unsigned cpu_get_id          (void);
static inline unsigned cpu_time_stamp      (void);

// ------------------------------------------------------------ IRQ register/enable/disable/restore
static inline void cpu_get_highest_irq     (void);
static inline void cpu_disable_single_irq  (unsigned irq_num, unsigned * old);
static inline void cpu_enable_single_irq   (unsigned irq_num, unsigned * old);
static inline void cpu_disable_all_irq     (unsigned * old);
static inline void cpu_enable_all_irq      (unsigned * old);
static inline void cpu_restore_irq         (unsigned old);

// ------------------------------------------------------------ Spinlock trylock/unlock/init
static inline void cpu_spin_init           (unsigned * lock);
static inline void cpu_spin_lock           (unsigned * lock);
static inline unsigned cpu_spin_trylock    (unsigned * lock);
static inline void cpu_spin_unlock         (unsigned * lock);
static inline void cpu_spin_destroy        (unsigned * lock);

// ------------------------------------------------------------ Cache operations
#define CACHE_LINE_SIZE  ...
static inline void cpu_invalid_dcache_line (void *ptr);

// ------------------------------------------------------------ cpu context
#define CONTEXT_SIZE     ...
extern void cpu_context_init               (unsigned ctx[], unsigned mode_usr, unsigned stack_ptr,
                                            unsigned entry_func, unsigned exit_func,
                                            unsigned arg1);
extern void cpu_context_load               (unsigned *ctx);
extern unsigned cpu_context_save           (unsigned *ctx);
extern void cpu_context_restore            (unsigned *ctx, unsigned val);

#include <hal_cpu_code.h>

#endif
```

HAL-CPU: redéfinition des types entier

http://en.wikipedia.org/wiki/C_data_types

- Tous les processeurs n'ont pas la même largeur de mot, donc la largeur des types entiers dépend du CPU (16/32/64)
  - int : mot machine
  - long : 4 octets
- La règle est que char ≤ short ≤ int ≤ long ≤ long long
- Il faut rédéfinir les types standards dans cpu/mipsel : stdint.h
  N=8 | 16 | 32 | 64
- Les types sont définis dans **stdint.h**

Type category	Signed types			Unsigned types		
	Type	Minimum value	Maximum value	Type	Minimum value	Maximum value
Exact width	intN_t	INTN_MIN	INTN_MAX	uintN_t	0	UINTN_MAX

HAL-CPU: redéfinition des types entier

```
#ifndef _STDINT_H_
#define _STDINT_H_

#ifndef NULL
#define NULL (void*)0
#endif

typedef unsigned long long uint64_t;
typedef unsigned int       uint32_t;
typedef unsigned short     uint16_t;
typedef unsigned char      uint8_t;

typedef long long          int64_t;
typedef int                int32_t;
typedef short              int16_t;
typedef char               int8_t;

typedef unsigned int       size_t;
typedef int                ssize_t;

#endif
```

Les limites sont définis dans le fichier: limits.h
mais gcc le fait seul en fonction du processeur pour lequel il compile

```
#define MB_LEN_MAX 16
#define SCHAR_MAX  127
#define UCHAR_MAX  255
#define CHAR_MAX    UCHAR_MAX
#define CHAR_MAX    SCHAR_MAX
#define SHRT_MAX   32767
#define USHRT_MAX  65535
#define INT_MIN    (-INT_MAX - 1)
#define INT_MAX    2147483647
#define UINT_MAX   4294967295U
#define LONG_MAX    9223372036854775807L
#define LONG_MAX    2147483647L
#define LONG_MIN   (-LONG_MAX - 1L)
#define ULONG_MAX   18446744073709551615UL
#define ULONG_MAX   4294967295UL
#define LLONG_MAX   9223372036854775807LL
#define LLONG_MIN   (-LLONG_MAX - 1LL)
#define ULLONG_MAX 18446744073709551615ULL
```

HAL-CPU : Registres spéciaux

dans hal-cpu.h

- static inline unsigned cpu_get_id(void);
- static inline unsigned cpu_time_stamp(void);

dans hal-cpu-code.h

```
static inline unsigned cpu_get_id(void)
{
    register unsigned proc_id;
    asm volatile ("mfc0  %0, $0":"=r" (proc_id));
    return proc_id;
}
static inline unsigned cpu_time_stamp(void)
{
    register unsigned cycles;
    asm volatile ("mfc0  %0, $9":"=r" (cycles));
    return cycles;
}
```

HAL-CPU : spinlock du CPU

```
static inline void cpu_spin_init(unsigned * lock){
    __asm__ volatile (
        "sync             \n" // to be sure that previous store are acheived
        "sw $0, (%0)      \n" // unlock
        "sync             \n" // to empty the write buffer
        ::"r" (lock));
}

static inline bool_t cpu_spin_trylock(unsigned * lock){
    register bool_t state;
    __asm__ volatile (
        ".set noreorder   \n" // do not modify the sequence
        "lw    %0, (%1)   \n" // load lock state
        "bnez  %0, 32     \n" // forgive whenever lock is busy (i.e. != 0)
        "nop              \n" // delayed slot
        "ll    %0, (%1)   \n" // load lock value in %0 and try to register in memory at %1
        "nop              \n" // delayed slot
        "bnez  %0, 16     \n" // forgive whenever lock is busy
        "ori   %0, 1      \n" // preload 1
        "sc    %0, (%1)   \n" // sc returns in %0 <- 1 if free, 0 if busy
        "nop              \n" // delayed slot
        "xori  %0, %0,  1 \n" // trylock returns 0 if free, 1 if busy
        ".set  reorder    \n" // return to normal mode
        :"=r" (state)
        :"r" (lock));
    return state;
}

static inline void cpu_spin_lock(unsigned * lock) {
    while ((cpu_spinlock_trylock(lock)));
}

static inline void cpu_spin_unlock(unsigned * lock) {
    cpu_spinlock_init(lock);
}

static inline void cpu_spin_destroy(unsigned * lock){}
```

HAL-CPU: Interruptions du CPU

```
static inline void cpu_disable_single_irq_mask(unsigned irq_mask, unsigned * old){
    register unsigned old_sr, new_sr;
    __asm__ volatile ("mfc0   %0, $12  \n":"=r" (old_sr));  // get old_SR
    if (old) *old = old_sr;
    new_sr = old_sr & ~irq_mask;
    __asm__ volatile ("mtc0   %0, $12  \n"::"r" (new_sr));  // set new_SR
}

static inline void cpu_enable_single_irq_mask(unsigned irq_mask, unsigned * old){
    register unsigned old_sr, new_sr;
    __asm__ volatile ("mfc0   %0, $12  \n":"=r" (old_sr));  // get old_SR
    if (old) *old = old_sr;
    new_sr = old_sr | irq_mask;
    __asm__ volatile ("mtc0   %0, $12  \n"::"r" (new_sr));  // set new_SR
}

static inline void cpu_disable_single_irq(unsigned irq_num, unsigned * old){
    cpu_disable_single_irq_mask((1 << (10 + irq_num)),old);
}

static inline void cpu_enable_single_irq(unsigned irq_num, unsigned * old){
    cpu_enable_single_irq_mask((1 << (10 + irq_num)),old);
}

static inline void cpu_disable_all_irq(unsigned * old){
    cpu_disable_single_irq_mask(1,old);
}

static inline void cpu_enable_all_irq(unsigned * old){
    cpu_enable_single_irq_mask(1,old);
}

static inline void cpu_restore_irq(unsigned old){
    __asm__ volatile ("mtc0 %0, $12"::"r" (old));
}
```