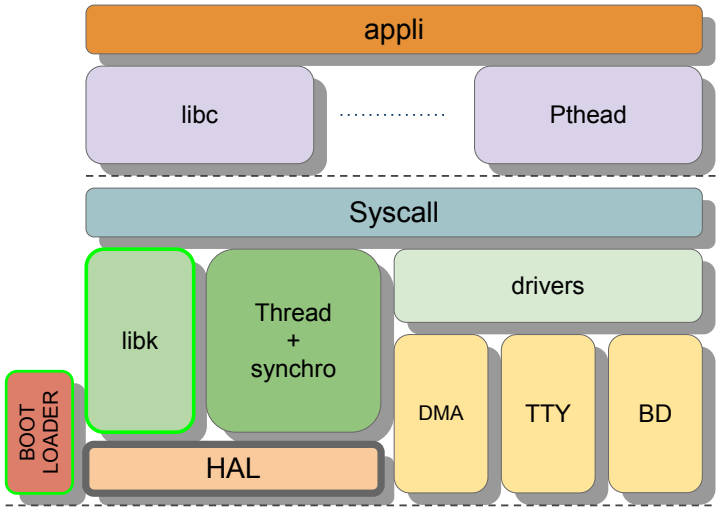
HAL - registers
Couches basses du noyau

MI074 - 4

Architecture de l'OS final



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.

Les fichiers de la HAL

```
.
|-- arch
|   `-- soclib
|       |-- __boot_loader.c          2
|       |-- hal_arch_code.c
|       |-- hal_arch_code.h
|       |-- kldscript.h
|       `-- segmentation.h
|-- build
|-- cpu
|   `-- mipsel
|       |-- hal_cpu_code.c           1 (boot.s)
|       |-- hal_cpu_code.h
|       `-- stdint.h
|-- kernel
|   |-- config.h
|   |-- __do_init.c                  3
|   |-- hal_arch.h
|   `-- hal_cpu.h
|
|
|-- libk
|   |-- ctype.c
|   |-- ctype.h
|   |-- kbzero.c
|   |-- kgetc.c
|   |-- kgets.c
|   |-- klist.h
|   |-- kmalloc.c
|   |-- kmalloc.h
|   |-- kmemmove.c
|   |-- kprintf.c
|   |-- krand.c
|   |-- kstdio.h
|   |-- kstdlib.h
|   |-- kstrcmp.c
|   |-- kstrcpy.c
|   |-- kstrlen.c
|   |-- kstrtol.c
|   `-- libk.h
`-- Makefile
```

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

- Interface Kernel avec le CPU
- Interface Kernel avec la plateforme (ARCHitecture)
- Interface Kernel avec les périphériques

Nous commencer en partie par l'interface CPU et l'ARCHitecture.
Nous aborderons les périphériques dans un prochain cours

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.

Organisation du code du système

```
|-- arch          HAL architecture
|   `-- soclib      spécifique à la plateforme (p.ex. segmention.h)
|                     et les drivers des periphériques
|
|-- 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-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  tty_getc    (void);
static int  tty_putc    (char 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

```
#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_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  // context table 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

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: 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: 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));
}
```

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){}
```

contexte du CPU

Ensemble des informations qui permettent de définir un contexte (un état) du CPU.

Il y a deux types d'informations:

1. celles qui servent au démarrage du thread
   - adresse de la fonction d'entrée
   - ...
2. celles qui servent en régime stationnaire.
   - table de stockage des registres
   - ...

Usage des registres (cas du mips)

- Les registres que la fonction appelante doit sauvegarder :

```
$at        : $1
$v0-$v1    : $2-$3
$a0-$a3    : $4-$7
$t0-$t7    : $8-$15
$t8-$t9    : $24-$25
```

- Les registres que la fonction appelée doit sauvegarder :

```
$s0-$s7    : $16-$23
$sp        : $29
$ra        : $31
```

HAL-CPU: context

Le contexte d'un CPU est simplement un tableau de registres

```
#define s(a)            #a
#define v(a)            s(a)

// -------------------------
// CPU CONTEXT
// -------------------------
#define     S0_16       0
#define     S1_17       1
#define     S2_18       2
#define     S3_19       3
#define     S4_20       4
#define     S5_21       5
#define     S6_22       6
#define     S7_23       7
#define     SP_29       8
#define     S8_30       9
#define     RA_31       10
#define     C0_SR       11
#define     EXIT_FUNC   12
#define     ARG1        13
#define     LOADABLE    14
```

```
void cpu_context_init(unsigned ctx[],
                      unsigned mode_usr,
                      unsigned stack_ptr,
                      unsigned entry_func,
                      unsigned exit_func,
                      unsigned arg1)
{
    ctx[C0_SR]      = (mode_usr) ? 0xFC13 : 0xFC03;
    ctx[SP_29]      = stack_ptr;
    ctx[RA_31]      = entry_func;
    ctx[EXIT_FUNC]  = exit_func;
    ctx[ARG1]       = arg1;
    ctx[LOADABLE]   = 1;
    // required to be sure that all data are written
    __asm__ volatile ("sync");
}
```

- Registres persistants seulement car les registres temporaires seront sauvés dans la pile du threads qui demande le changement de contexte.
- Pointeur de pile sur la dernière case occupée
- Adresse de retour qui est au départ l'adresse de la fonction du thread
- Registre status qui sera restauré (U ou K)
- Adresse de la fonction de sortie du thread
- Argument du thread
- Booléen indiquant si un thread vient d'être créé

registre status

dans MIPS_vol3 p. 53

contenu des 16 bits de poids faible du registre **SR**:

IM[7:0]	0	0	0	UM	0	ERL	EXL	IE

Cette version du processeur MIP32 n'utilise que 12 bits du registre SR :

IE : Interrupt Enable
EXL : Exception Level
ERL : Reset Level
UM : User Mode
IM[7:0] : Masques individuels pour les six ligne d'nterruption matérielles (bits IM[7:2]) et pour les 2 interruptions logicielles (bits IM[1:0])

La commutation de tâches

- Sauvegarder le contexte du Thread courant
- Élire un nouveau Thread à partir de la liste des Threads à l'état prêt (READY) du processeur courant, selon la politique d'ordonnancement de ce processeur.
- Restaurer le contexte du Thread élu

Commutation de tâches : cpu_context_save()

```
if(cpu_context_save() == 0)
{
    // restore
}
de sortie du restore
```

Quand cpu_context_save() est appelée, elle retourne 0, ce qui veut dire que les instructions du restore vont être exécutées

MAIS

Quand le contexte du thread est restauré, $2 contiendra autre chose que 0, et les instructions de retour du restore sont alors exécutées.

Sauvegarde/restauration du *contexte*

- La sauvegarde de *contexte* est effectuée par la fonction cpu_context_save().
- Lors de la sauvegarde, la fonction cpu_context_save() écrit la valeur 0 dans le registre $2 (registre résultat) avant la sauvegarde du contexte.
- La fonction cpu_context_save() retourne la valeur 1 lorsque le thread sera restauré : la fonction cpu_context_restore()

HAL-CPU: context

```
__asm__(// unsigned cpu_context_save(
        //          struct cpu_context_s * ctx)
"   .section .text, \"ax\", @progbits   \n"
"   .align 2                            \n"
"   .globl  cpu_context_save            \n"
"   .ent    cpu_context_save            \n"
"                                       \n"
"cpu_context_save:                      \n"
"   mfc0  $2,   $12                     \n"
"   sw    $16,  "v(S0_16)"*4($4)        \n"
"   sw    $17,  "v(S1_17)"*4($4)        \n"
"   sw    $18,  "v(S2_18)"*4($4)        \n"
"   sw    $19,  "v(S3_19)"*4($4)        \n"
"   sw    $20,  "v(S4_20)"*4($4)        \n"
"   sw    $21,  "v(S5_21)"*4($4)        \n"
"   sw    $22,  "v(S6_22)"*4($4)        \n"
"   sw    $23,  "v(S7_23)"*4($4)        \n"
"   sw    $29,  "v(SP_29)"*4($4)        \n"
"   sw    $30,  "v(S8_30)"*4($4)        \n"
"   sw    $31,  "v(RA_31)"*4($4)        \n"
"   sw    $2,   "v(C0_SR)"*4($4)        \n"
"   li    $2,   0                       \n"
"   jr    $31                           \n"
"                                       \n"
"   .end    cpu_context_save            \n"
);
```

```
__asm__(//void cpu_context_restore(
        //     struct cpu_context_s * ctx,
        //     unsigned val)
"   .section .text, \"ax\", @progbits   \n"
"   .align 2                            \n"
"   .globl  cpu_context_restore         \n"
"   .ent    cpu_context_restore         \n"
"                                       \n"
"cpu_context_restore:                   \n"
"    lw    $27,  "v(LOADABLE)"*4($4)    \n"
"    lw    $26,  "v(C0_SR)"*4($4)       \n"
"    lw    $29,  "v(SP_29)"*4($4)       \n"
"    bne   $27,  $0,  cpu_context_load  \n"
"    lw    $31,  "v(RA_31)"*4($4)       \n"
"    lw    $16,  "v(S0_16)"*4($4)       \n"
"    lw    $17,  "v(S1_17)"*4($4)       \n"
"    lw    $18,  "v(S2_18)"*4($4)       \n"
"    lw    $19,  "v(S3_19)"*4($4)       \n"
"    lw    $20,  "v(S4_20)"*4($4)       \n"
"    lw    $21,  "v(S5_21)"*4($4)       \n"
"    lw    $22,  "v(S6_22)"*4($4)       \n"
"    lw    $23,  "v(S7_23)"*4($4)       \n"
"    lw    $30,  "v(S8_30)"*4($4)       \n"
"    mtc0  $26,  $12                    \n"
"    move  $2,   $5                     \n"
"    jr    $31                          \n"
"                                       \n"
"cpu_context_load:                      \n"
"    sw    $0,   "v(LOADABLE)"*4($4)    \n"
"    lw    $27,  "v(RA_31)"*4($4)       \n"
"    lw    $31,  "v(EXIT_FUNC)"*4($4)   \n"
"    lw    $4,   "v(ARG1)"*4($4)        \n"
"    mtc0  $27,  $14                    \n"
"    mtc0  $26,  $12                    \n"
"    addiu $29,  $29,  -4               \n"
"    eret                               \n"
"                                       \n"
"   .end    cpu_context_restore         \n"
);
```