HAL & context
Couches basses du noyau

MI074 -5

Plan

Le but de la séance est de terminer la couche HAL pour le CPU

context
création, destruction, chargement, sauvegarde, ...
registres spéciaux
numéro de proc, timestamp, numéro thread, ...
irq du CPU
masquage, démasquage, ...
spinlock
lock, unlock, ...
caches
invalidation de ligne de cache data

irq du cpu

```
#undef  CPU_IRQ_NR
#define CPU_IRQ_NR   6

static inline void cpu_disable_single_irq_mask(uint_t irq_mask, uint_t * old)
{
    register uint_t 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(uint_t irq_mask, uint_t * old)
{
    register uint_t 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(uint_t irq_num, uint_t * old) {
    cpu_disable_single_irq_mask((1 << (10 + irq_num)),old);
}
static inline void cpu_enable_single_irq(uint_t irq_num, uint_t * old) {
    cpu_enable_single_irq_mask((1 << (10 + irq_num)),old);
}
static inline void cpu_disable_all_irq(uint_t * old) {
    cpu_disable_single_irq_mask(1,old);
}
static inline void cpu_enable_all_irq(uint_t * old) {
    cpu_enable_single_irq_mask(1,old);
}
static inline void cpu_restore_irq(uint_t old) {
    __asm__ volatile ("mtc0 %0, $12"::"r" (old));
}
```

cpu_spinlock

```
static inline void cpu_spinlock_init(uint_t * 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_spinlock_trylock(uint_t * lock)
{
    register bool_t state;
    __asm__ volatile (
        ".set noreorder   \n" // do not modify the sequence
        "lw    %0, (%1)   \n" // load lock state
        "bnez  %0, 24     \n" // forgive whenever lock is busy (i.e. != 0)
        "nop              \n" // delayed slot
        "ll    %0, (%1)   \n" // load lock value in $2 and try to register in memory
        "bnez  %0, 12     \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
        "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_spinlock_lock(uint_t * lock){
    while ((cpu_spinlock_trylock(lock)));
}
static inline void cpu_spinlock_unlock(uint_t * lock) {
    cpu_spinlock_init(lock);
}
static inline void cpu_spinlock_destroy(uint_t * lock) {
    cpu_spinlock_init(lock);
}
```

HAL : Contexte d'exécution sur CPU

```
struct cpu_context_s;                          // structure opaque context
static inline void cpu_context_init(           // initialisation de ctx
                    struct cpu_context_s * ctx,
                    uint_t mode_usr,
                    uint_t stack_ptr,
                    uint_t entry_func,
                    uint_t exit_func,
                    uint_t thread_ptr,
                    uint_t arg1);

extern void cpu_context_load(struct cpu_context_s * ctx);
extern uint_t cpu_context_save(struct cpu_context_s * ctx);
extern void cpu_context_restore(struct cpu_context_s * ctx, uint_t val);
static inline void cpu_context_destroy(struct cpu_context_s * ctx);
```

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
```

Nommage des registres : mips_regs.h

```
#define      AT          0
#define      V0          1
#define      V1          2
#define      A0          3
#define      A1          4
#define      A2          5
#define      A3          6
#define      T0          7
#define      T1          8
#define      T2          9
#define      T3          10
#define      T4          11
#define      T5          12
#define      T6          13
#define      T7          14
#define      T8          15
#define      LO          16
#define      T9          17
#define      SR          18
#define      HI          19
```

```
#define      RA          20
#define      EPC         21
#define      GP          22
#define      S0          23
#define      S1          24
#define      S2          25
#define      S3          26
#define      S4          27
#define      S5          28
#define      S6          29
#define      S7          30
#define      CR          31
#define      SP          32
#define      S8          33
#define      SAVE_REG_NB 34
```

contexte du CPU

cpu_context_s

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
   - ...

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])

cpu_context_s

```
struct cpu_context_s {
    uint_t s0_16;          // 0
    uint_t s1_17;          // 1
    uint_t s2_18;          // 2
    uint_t s3_19;          // 3
    uint_t s4_20;          // 4
    uint_t s5_21;          // 5
    uint_t s6_22;          // 6
    uint_t s7_23;          // 7
    uint_t sp_29;          // 8
    uint_t fp_30;          // 9
    uint_t ra_31;          // 10
    uint_t c0_sr;          // 11
    uint_t thread_ptr;     // 12
    uint_t exit_func;      // 13
    uint_t arg1;           // 14
    uint_t loadable;       // 15
} __attribute__ ((packed));
```

```
static inline void cpu_context_init(
    struct cpu_context_s *ctx,
    uint_t mode_usr,
    uint_t stack_ptr,
    uint_t entry_func,
    uint_t exit_func,
    uint_t thread_ptr,
    uint_t arg1)
{
    ctx->c0_sr = (mode_usr)?0xFC12:
0xFC02;
    ctx->sp_29 = stack_ptr;
    ctx->ra_31 = entry_func;
    ctx->exit_func = exit_func;
    ctx->thread_ptr = thread_ptr;
    ctx->arg1 = arg1;
    ctx->loadable = 1;
}
```

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

Sauvegarde/restauration du *contexte*

- La sauvegarde de *contexte* est effectuée par la fonction cpu_context_save().
- Le contexte (matériel i.e du CPU) du thread courant est sauvegardé dans la structure du thread (le champ PWS).
- 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()

Commutation de tâches : cpu_context_save()

```
1:      if(cpu_context_save())
2:      {
3:          /* instructions A */
4:      }
5:      else
6:      {
7:          /* instructions  B */
8:      }
```

Quand cpu_context_save() est appelée, elle retourne 1, ce qui veut dire que les instructions A vont être exécutées

MAIS

Quand le contexte du thread est restauré, $2 contiendra 0, et les instructions B sont alors exécuter

hal_cpu.S : cpu_context_load

```
# void cpu_context_load(struct cpu_context_s * ctx)
# ------------------------------------------------------------------
cpu_context_load:
        .globl  cpu_context_load
        .ent    cpu_context_load
        sw    $0,   15*4($4)        # reset loadable flag
        lw    $26,  11*4($4)        # get mode
        mtc0  $26,  $12             # set SR
        lw    $26,  12*4($4)        # get thread pointer
        mtc0  $26,  $4,   2         # set thread pointer
        lw    $29,  8*4($4)         # get stack ptr
        lw    $31,  13*4($4)        # get exit_func
        lw    $26,  10*4($4)        # get entry_func
        mtc0  $26,  $14             # set EPC with entry_func
        lw    $4,   14*4($4)        # get thread arg1
        addiu $29,  $29,  -4        # entry_func has one arg
        eret                        #
        .end    cpu_context_load
```

hal_cpu.S : cpu_context_save

```
.section .text, "ax", @progbits
.align 2

# uint_t cpu_context_save(struct cpu_context_s * ctx)
# ------------------------------------------------------------------
cpu_context_save:
        .globl  cpu_context_save
        .ent    cpu_context_save
        sw    $16,  0*4($4)         # save registers
        sw    $17,  1*4($4)         #
        sw    $18,  2*4($4)         #
        sw    $19,  3*4($4)         #
        sw    $20,  4*4($4)         #
        sw    $21,  5*4($4)         #
        sw    $22,  6*4($4)         #
        sw    $23,  7*4($4)         #
        mfc0  $26,  $4,   2         # get current thread pointer
        sw    $29,  8*4($4)         #
        mfc0  $2,   $12             # get SR
        sw    $30,  9*4($4)         #
        sw    $31,  10*4($4)        #
        sw    $2,   11*4($4)        # save SR
        sw    $26,  12*4($4)        # save current thread pointer
        li    $2,   0               # return 0 after save
        jr    $31                   #
        .end    cpu_context_save
```

hal_cpu.S : cpu_context_restore

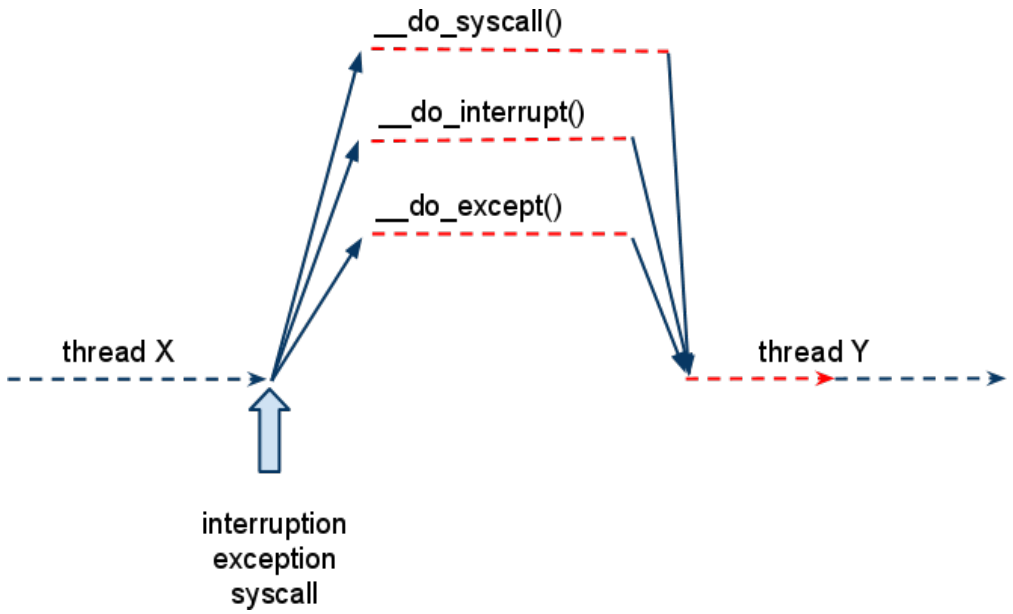
```
# void cpu_context_restore(struct cpu_context_s * ctx, uint_t val)
# ------------------------------------------------------------------
cpu_context_restore:
        .globl  cpu_context_restore
        .ent    cpu_context_restore
        lw    $27,  15*4($4)        # get loadable flag
        bne   $27,  $0,  cpu_context_load # if set then context_load
        lw    $26,  11*4($4)        # get mode
        mtc0  $26,  $12             # restore SR
        lw    $26,  12*4($4)        # get thread pointer
        mtc0  $26,  $4,   2         # set thread pointer
        lw    $29,  8*4($4)         # get stack ptr
        lw    $16,  0*4($4)         # retore registers
        lw    $17,  1*4($4)         #
        lw    $18,  2*4($4)         #
        lw    $19,  3*4($4)         #
        lw    $20,  4*4($4)         #
        lw    $21,  5*4($4)         #
        lw    $22,  6*4($4)         #
        lw    $23,  7*4($4)         #
        lw    $30,  9*4($4)         #
        lw    $31,  10*4($4)        #
        move  $2,   $5              #
        jr    $31                   #
        .end    cpu_context_restore
```

kentry : l'entrée du système

- On entre dans le système à la suite de 3 événements:
    1. une interruption provenant d'un périphérique.
    2. une exception détectée par le matériel
    3. une demande de service par l'utilisateur

- L'entrée du système est à l'adresse kentry
  Le code est spécifique au CPU

Les points d'entrée du noyau



Les points d'entrée du noyau: interruption 1/2

Contrairement aux SYSCALL les interruptions surviennent n'importe où.
Quand une interruption survient, on termine l'instruction puis :

- Passage en mode kernel (sauf si on y est déjà)
- Sauvegarde du mode (status) d'origine
- *Sauvegarde du pointeur de pile user et passage à la pile kernel*
- Sauvegarde dans la pile kernel tous les registres temporaires + SR + l'adresse de retour (EPC)
- Fabrication des paramètres this, proc_id et irq_state
- Appel de la fonction __do_interrupt (interruptions masquées)

```
void __do_interrupt(unsigned int proc_id,
                    unsigned int irq_state)
```

- Restauration des registres temporaires
- *Restauration du pointeur de pile, si on venait du mode user*
- Retour dans le mode d'origine
- Retour dans la séquence interrompue

Les points d'entrée du noyau: interruption 2/2

Le traitement des interruptions n'est pas interruptible.

- Cela ne permet pas de faire des traitements longs.

- MAIS
  cela évite le débordement de la pile kernel.
- ET
  cela oblige le traitement en deux temps des événements
  1. immédiat : acquittement + déplacement des données
  2. plus tard : traitement des données

- CE QUI
  nécessite la définition d'un gestionnaire d'événements

- QUI PERMET
  les traitements de manière retardés et possiblement déportés

Les points d'entrée du noyau: exception 1/2

Les exceptions sont des événements asynchrones issus de la plateforme, qui ne sont pas masquables (à la différence des interruptions).

Cela peut être:
- une violation de privilège
- un accès non aligné à la mémoire
- une division par 0
- un accès à une adresse non mappée
- une instruction inconnue
- un page-fault ou un miss tlb de la MMU
- etc...

Les points d'entrée du noyau: exception 2/2

Dès qu'une exception survient :

- Passage en mode kernel (sauf si on y est déjà)
- Sauvegarde du mode d'origine
- *Sauvegarde du pointeur de pile user et usage de la pile kernel*
- Sauvegarde dans la pile kernel tous les registres + SR + l'adresse de retour
- Fabrication des paramètres
- Appel de la fonction __do_exception (interruptions masquées)

```
reg_t __do_exception(uint_t cpu_id,
                     uint_t * reg)
```

- Restauration des registres
- *Restauration du pointeur de pile, si on venait du mode user*
- Retour dans le mode d'origine
- Retour dans la séquence interrompue

kentry.S

```
#include <mips_regs.h>

#-----------------------------------------------------------------------------
# Kernel entry point (Exception/Interrupt/System call) for MIPS32 ISA compliant
# processors. The base address of the segment containing this code
#-----------------------------------------------------------------------------
     .section   .kentry,"ax",@progbits
     .extern    __do_interrupt
     .extern    __do_exception
     .ent       kentry
     .set       noat
     .set       noreorder
     .org       0x180

#number of arguments of called functions
#define NBA         2
```

kentry.S

```
# Kernel Entry point
#-------------------------------------------------------------
kentry:

    # alloc memory in stack to be able to save all registers
    addiu   $29,    $29,    -(SAVE_REG_NB+NBA)*4  # max registers to save + args

    # just save temporary registers
    sw      $1,     (NBA+AT)*4($29)
    sw      $2,     (NBA+V0)*4($29)
    sw      $3,     (NBA+V1)*4($29)
    sw      $4,     (NBA+A0)*4($29)
    sw      $5,     (NBA+A1)*4($29)
    sw      $6,     (NBA+A2)*4($29)
    sw      $7,     (NBA+A3)*4($29)
    sw      $8,     (NBA+T0)*4($29)
    sw      $9,     (NBA+T1)*4($29)
    sw      $10,    (NBA+T2)*4($29)
    sw      $11,    (NBA+T3)*4($29)
    sw      $12,    (NBA+T4)*4($29)
    sw      $13,    (NBA+T5)*4($29)
    sw      $14,    (NBA+T6)*4($29)
    sw      $15,    (NBA+T7)*4($29)
    sw      $24,    (NBA+T8)*4($29)
    sw      $25,    (NBA+T9)*4($29)
    sw      $28,    (NBA+GP)*4($29)
    sw      $31,    (NBA+RA)*4($29)
    mflo    $1
    mfhi    $2
    mfc0    $3,     $14                 # Read EPC
    mfc0    $5,     $13                 # read CR (used later)
    sw      $1,     (NBA+LO)*4($29)
    sw      $2,     (NBA+HI)*4($29)
    sw      $3,     (NBA+EPC)*4($29)    # Save EPC
```

kentry.S

```
    # test cause register jump to cause_int if it is an interrupt
    andi    $6,     $5,    0x3          # apply interrupt mask (used later)
    beq     $6,     $0,     cause_int
    mfc0    $4,     $0                  # CPU_ID, 1th arg

# Exception
# ----------------------------------------------

    # since this is an exception, save all the remaining registers
    mfc0    $1,     $12                 # Read current SR (used later)
    addiu   $2,     $29,    (SAVE_REG_NB+NBA)*4
    sw      $1,     (NBA+SR)*4($29)     # Save SR
    sw      $2,     (NBA+SP)*4($29)
    sw      $5,     (NBA+CR)*4($29)
    sw      $16,    (NBA+S0)*4($29)
    sw      $17,    (NBA+S1)*4($29)
    sw      $18,    (NBA+S2)*4($29)
    sw      $19,    (NBA+S3)*4($29)
    sw      $20,    (NBA+S4)*4($29)
    sw      $21,    (NBA+S5)*4($29)
    sw      $22,    (NBA+S6)*4($29)
    sw      $23,    (NBA+S7)*4($29)
    sw      $30,    (NBA+S8)*4($29)

    # call function __do_exception( cpu_id, regs_table)
    la      $27,    __do_exception
    or      $5,     $0,     $29     # regs_tbl, 2th arg
    jr      $27
    addiu   $5,     $5,     NBA*4
```

kentry.S

```
# Interrupt
# ----------------------------------------------
cause_int:

    # call function __do_interrupt( cpu_id, irq_state)
    la      $27,    __do_interrupt
    srl     $5,     $5,     10      # extract irq state
    jal     $27
    andi    $5,     $5,     0x3F    # 6 HW IRQ LINES, 2th arg is irq_state
```

kentry.S

```
# Kentry exit
# ---------------------------------------------

    # only restore temporary register
    lw      $1,     (NBA+LO)*4($29)
    lw      $2,     (NBA+HI)*4($29)
    lw      $3,     (NBA+EPC)*4($29)
    mtlo    $1
    mthi    $2
    mtc0    $3,     $14
    lw      $1,     (NBA+AT)*4($29)
    lw      $2,     (NBA+V0)*4($29)
    lw      $3,     (NBA+V1)*4($29)
    lw      $4,     (NBA+A0)*4($29)
    lw      $5,     (NBA+A1)*4($29)
    lw      $6,     (NBA+A2)*4($29)
    lw      $7,     (NBA+A3)*4($29)
    lw      $8,     (NBA+T0)*4($29)
    lw      $9,     (NBA+T1)*4($29)
    lw      $10,    (NBA+T2)*4($29)
    lw      $11,    (NBA+T3)*4($29)
    lw      $12,    (NBA+T4)*4($29)
    lw      $13,    (NBA+T5)*4($29)
    lw      $14,    (NBA+T6)*4($29)
    lw      $15,    (NBA+T7)*4($29)
    lw      $24,    (NBA+T8)*4($29)
    lw      $25,    (NBA+T9)*4($29)
    lw      $30,    (NBA+GP)*4($29)
    lw      $31,    (NBA+RA)*4($29)
    addiu   $29,    $29,    (SAVE_REG_NB+NBA)*4  # max registers to save + args
    eret

    .set reorder
    .set at
    .end kentry
#-------------------------------------------------------------------------------
```