Programme en mode User

MI074 - 8

Questions

Que va-t-il falloir faire pour permettre la programmation en mode user ?

- Quelle est l'interface entre les mondes user et système ?
- Quel est l'impact sur la plateforme ?
- Où mettre le programme pour le lancer depuis le système ?
- Comment passer du mode user au mode système
- Comment compiler et faire l'édition de lien ?
- Comment lancer un thread utilisateur ?

Combien de codes

- code OS
  - kernel le noyau
  - libk les fonctions de services
  - arch spécifique à la plateforme
  - cpu spécifique au processeur
- code des bibliothèques de l'utilisateur
  - libc les fonctions standards
  - pthread la gestion des threads et des synchro
  - crt0 code de lancement d'une application
- code des applications
  - une seule application à la fois

Interface entre les deux mondes

- des appels systèmes:
  - services standardisés proposé par le système pour l'application (ou les applications).
- Un appel système c'est :
  - un numéro de service
  - des paramètres pour le service
  - un résultat qui prend la forme d'un effet de bord
  - un état de retour (0 si tout va bien le plus souvent)
  - un numéro d'erreur pour connaitre la raison de l'erreur (errno)
- exemple ?

Passage du mode usr au mode sys

- C'est un appel de fonction

```
typeOut
service(typeIn0 a0, typeIn1 a1, typeIn2 a2, typeIn3 a3)
```

  - Il n'y a pas plus de 4 paramètres et un résultat
  - avec un effet de bord sur la variable errno
  - mais la fonction va être réalisée par le noyau
- Quand on appelle une fonction en C,
  - Les registres temporaires sont perdus
  - Les registres persistants sont intacts
  - Il ne faut pas inliner pour connaitre l'état des registres
  - On va écrire tout le corps de la fonction en assembleur avoir des garanties sur l'usage des registres.

```
typeOut __attribute__((noinline))
service(typeIn0 a0, typeIn1 a1, typeIn2 a2, typeIn3 a3)
   SYSCALL(service_no)
```

syscall.h

- interface entre l'utilisateur et le noyau
  - déclaration des devices (fb, hd, dma)
  - declaration des appels systèmes

```
enum SYSCALL_E {
    WRITE,
    READ,
    IOCTL,
    CPU_ID,                 // cpu identifier
    CPU_TSC,                // cpu Time Stamp Counter
    PTHREAD_CREATE,
    PTHREAD_EXIT,
    ...
    ERRNO_LOCATION,         // return &errno in user space of the current thread
    SYSCALL_UNDEF
};

#define SYSCALL_NR  16

#ifdef __KERNEL__
#   define SYS(name) sys_##name
#else
#   define SYS(name) __attribute__((noinline)) name
#endif

// ------------------------------------------------------------------------------- sys_drivers.c
extern int        SYS(write)               (int fd, const void *buf, size_t size);
extern int        SYS(read)                (int fd, const void *buf, size_t size);
extern int        SYS(ioctl)               (int fd, int request, void *arg);
// ------------------------------------------------------------------------------- sys_pthread.c
extern int        SYS(pthread_create)       (pthread_t *pthread, const pthread_attr_t *attr,
                                            void *(*start_routine)(void*), void *arg);
extern void       SYS(pthread_exit)        (void * value_ptr);
...
// ------------------------------------------------------------------------------- sys_info.c
extern unsigned * SYS(errno_location)      (void);
extern unsigned   SYS(cpu_id)              (void);
extern unsigned   SYS(cpu_tsc)             (void);
```

Passage du mode usr au mode sys

```
// for MIPS : $2 must contain the service number
// we will force syscall to return from the current function with the result
// the return statetement is required by gcc but it will never execute here #define
SYSCALL(service) {   \
   __asm__ volatile (            \
   "    li $2, %0 \n"            \
   "    syscall \n"          \
   ::   "i"(service));           \
   return 0;                     \
}
```

```
80003308 <write>:
80003308: 2402000c  li      v0,12
8000330c: 0000000c  syscall
80003310: 03e00008  jr      ra
80003314: 00001021  move    v0,zero
```

Par exemple:

```
syscall.h
    extern int __attribute__((noinline)) write(int fd, char *buffer, size_t size);
syscall.c
    int write(int fd, char *buffer, size_t size) SYSCALL(WRITE)
```

app.c

```
type f () {
    ...
    int res = write(tty2, "bonjour", 8);
    ...
}
```

syscall.c

```
#include <syscall.h>
#include <stdint.h>
#include <hal_cpu.h>

int write (int fd, const void *buf, size_t size)              SYSCALL(WRITE)
int read (int fd, const void *buf, size_t size)               SYSCALL(READ)
int ioctl (int fd, int request, void *argp)                   SYSCALL(IOCTL)

unsigned cpu_id (void)                                        SYSCALL(CPU_ID)
unsigned cpu_tsc (void)                                       SYSCALL(CPU_TSC)

int pthread_create (pthread_t *thread, const pthread_attr_t *attr,
                    void *(*start_routine)(void*), void *arg)  SYSCALL(PTHREAD_CREATE)
void pthread_exit (void * value_ptr)                          SYSCALL_VOID(PTHREAD_EXIT)
int pthread_join (pthread_t thread, void **value_ptr)         SYSCALL(PTHREAD_JOIN)
int pthread_yield (void)                                      SYSCALL(PTHREAD_YIELD)

int pthread_mutex_lock (pthread_mutex_t *mutex)               SYSCALL(PTHREAD_MUTEX_LOCK)
int pthread_mutex_trylock (pthread_mutex_t *mutex)            SYSCALL(PTHREAD_MUTEX_TRYLOCK)
int pthread_mutex_unlock (pthread_mutex_t *mutex)             SYSCALL(PTHREAD_MUTEX_UNLOCK)

unsigned * errno_location (void)                              SYSCALL(ERRNO_LOCATION)
```

syscall

syscall.h

```
enum SYSCALL_E {
    WRITE,
    READ,
    IOCTL,
    CPU_ID,
    CPU_TSC,
    PTHREAD_CREATE,
    PTHREAD_EXIT,
    ...
    ERRNO_LOCATION,
    SYSCALL_UNDEF
};
```

ksyscall.c

```
#include <syscall.h>
#include <libk.h>

int undefined_syscall( int a0, int a1, int a2, int a3, int
service)
{
    kprintf("Unused syscall n°%d a0=%x a1=%x a2=%x a3=%x\n",
            service, a0, a2, a2, a3);
    return service;
}

void * syscall_vector[SYSCALL_NR] =
{
    [WRITE                      ] = &sys_puts,
    [REAS                           ] = &sys_write,
    ...
    [ERRNO_LOCATION                 ] = &sys_errno_location,
    [SYSCALL_UNDEF ... SYSCALL_NR-1 ] = &undefined_syscall
};
```

sys_driver.c

```
int sys_write (int fd, const void *buf, size_t size)
{
    dev_request_t req, *req1 = &req;
    switch (fd) {case DEV_IO: case DEV_ERR: case DEV_LOG:
        req.src = (void *)buf;
        req.size = (uint32_t)size;
        dev_tty(fd)->op.generic.write(dev_tty(fd),req1);
        return req.err;
    }
    return 0;
}
```

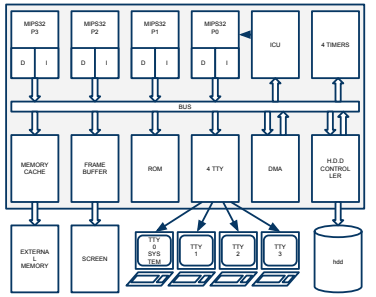
kentry

```
__asm__(
".section   .kentry,\"ax\",@progbits              \n"
".ent        kentry                               \n"
".set        noat                                 \n"
".org        0x180                                \n"
"                                                 \n"
"kentry:                                          \n"
"   mfc0    $26,    $13                           \n" // read CR
"   andi    $26,    $26,    0x3C                  \n" // apply cause mask
"   li      $27,    0x20                          \n" // syscall code
"   bne     $26,    $27,    not_syscall           \n" // that is not a syscall
"                                                 \n"
"syscall:                                         \n" // $4, $5, $6, $7 contains syscall args
"   addiu   $29,    $29,    -7*4                  \n" // for $31 SR + service + args
"   mfc0    $27,    $12                           \n" // read SR
"   sw      $2,     4*4($29)                      \n" // save service
"   sw      $27,    5*4($29)                      \n" // save SR
"   sw      $31,    6*4($29)                      \n" // save $31 return of funct with syscall
"   la      $26,    syscall_vector                \n" // function to call
"   andi    $2,     $2,     "v(SYSCALL_NR)"-1     \n" // SYSCALL_NB must a power of 2
"   sll     $2,     $2,     2                     \n" // compute syscall index
"   addu    $26,    $26,    $2                    \n" // in syscall_vector
"   lw      $26,    ($26)                         \n" // get syscall function address
"   li      $27,    0xFF00                        \n" // UM=0 ERL=0 EXL=0 IE=0
"   mtc0    $27,    $12                           \n" // SR <- kenel-mode, without INT
"   jalr    $26                                   \n" // call syscall function
"   lw      $27,    5*4($29)                      \n" // get old SR
"   lw      $31,    6*4($29)                      \n" // restore $31
"   mtc0    $27,    $12                           \n" // restore SR
"   mtc0    $31,    $14                           \n" // EPC <- $31 since using eret
"   addiu   $29,    $29,    7*4                   \n" // restore stack pointer
"   eret                                          \n" // equiv to jr $31
"                                                 \n"
"not_syscall:                                     \n"
"   addiu   $29,    $29, -("v(SAVE_REG_NB+NBA)")*4 \n" // saved regs + function args
"   sw      $1,     ("v(NBA+AT)")*4($29)          \n" // save temporary registers
"   sw      $2,     ("v(NBA+V0)")*4($29)          \n"
```

Passage du mode usr au mode sys

- entrée dans kentry
  - *on teste si on est en mode user*
    - *si oui on change de pile*
  - on teste si c'est un syscall
    - si oui
      on sauve l'adresse de retour, SR, et le service
      on passe en mode Kernel sans IT
      on saute à __do_syscall
      on restaure l'adresse de retour, SR
      on sort avec eret
      on restaure la pile
  - on teste si c'est une interruption
    - ...

Ou se trouve le code de l'utilisateur



- Où mettre le code ?
  - sur le disque
  - même sans système de fichiers

- Il faut compiler séparement

Au commencement

__do_init
- initialise l'architecture
- crée les thread idle
- Le cpu0 crée le thread init
  - charge le programme utilisateur en mémoire
  - lance programme
    - thread utilisateur : fonction start()
    - qui lance la fonction main()

code user : crt0.h crt0.c

```
#ifndef _CRT0_H_
#define _CRT0_H_

typedef void * (start_function_t)(void);

struct header_app_s {
    start_function_t *  start;
    uint32_t            magic;
    void *              data_end;
    void *              ram_end;
};

extern int main();          // main funtion of application
extern unsigned __data_end; // last address of data section
extern unsigned __ram_end;  // last address of ram section
extern unsigned __bss;      // first addr of bss section
extern unsigned __bss_end;  // last addr of bss section

#endif
```

```
#include <stdlib.h>
#include <crt0.h>

__attribute__((section(".header")))
struct header_app_s header = {
    &_start,
    0xDEADBEEF,
    &__data_end,
    &__ram_end
};

int errno;

void _start(void)
{
    for(int *p = (int*)&__bss; p != (int*)&__bss_end; *p++ = 0);
    malloc_init(header.data_end, header.ram_end);
    main();
}
```

Au commencement ...

```
#define BLOCKSIZE 512

void *th_init(void *arg)
{
    unsigned *ram0 = (unsigned *)RAM_BASE;
    unsigned *ram1 = (unsigned *)(RAM_BASE+BLOCKSIZE);

    struct header_app_s *header = (struct header_app_s *)RAM_BASE;

    struct dev_request_s req0 = {.src = 0, .dst = ram0, .size = 1 };
    dev_bd(0)->op.generic.read(dev_bd(0), &req0);

    ASSERT_ERROR(header->magic == CONFIG_MAGIC_APP,"no application loaded");

    unsigned nbblock = ((unsigned)header->data_end - (unsigned)ram1)/BLOCKSIZE;
    struct dev_request_s req1 = {.src = (void*)1, .dst = ram1, .size = nbblock};
    dev_bd(0)->op.generic.read(dev_bd(0), &req1);

    // il faut creer le thread USER et basculer dedans
    header->start();

    return NULL;
}
```

ldscript de l'utilisateur

```
#include <segmentation.h>

MEMORY
{
    ram   : ORIGIN = RAM_BASE, LENGTH = RAM_SIZE_USER
}

__ram_end = RAM_BASE + RAM_SIZE_USER ;

SECTIONS
{
    .ram   : {
        *(.header)
        *(.text)
        . = ALIGN(4); __data = .;
        *(.data*) *(.rodata*)
        . = ALIGN(4); __bss = .;
        *(.scommon*) *(.bss*) *(COMMON*)
        . = ALIGN(4); __bss_end = .;
        . = ALIGN(512); __data_end = .;
    } > ram
}

// all following command can have several arguments separated by space
// SEARCH_DIR for all directories where librairies are placed, eq. to -L
// GROUP      force librairies to be read, eq. to -l
// INPUT      force object file to be read

SEARCH_DIR(LIB_DIR)
GROUP(LIB_DIR/libc.a)
INPUT(LIB_DIR/crt0.o LIB_DIR/syscall.o)
```