Prova pratica di Calcolatori Elettronici

C.d.L. in Ingegneria Informatica, Ordinamento DM 270

24 febbraio 2017

1. Siano date le seguenti dichiarazioni, contenute nel file `cc.h`:

```
struct st1 { int vi[4]; };
struct st2 { char vd[4]; };
class cl {
        char v1[4]; char v3[4]; long v2[4];
public:
        cl(st1 ss);
        cl(st1& s1, int ar2[]);
        cl elab1(char ar1[], st2 s2);
        void stampa() {
                for (int i = 0; i < 4; i++) cout << (int)v1[i] << ' '; cout << endl;
                for (int i = 0; i < 4; i++) cout << (int)v2[i] << ' '; cout << endl;
                for (int i = 0; i < 4; i++) cout << (int)v3[i] << ' '; cout << endl << endl;
        }
};
```

Realizzare in Assembler GCC le funzioni membro seguenti.

```
cl::cl(st1 ss)
{
        for (int i = 0; i < 4; i++) {
                v1[i] = ss.vi[i]; v2[i] = ss.vi[i] / 2;
                v3[i] = 2 * ss.vi[i];
        }
}
cl::cl(st1& s1, int ar2[])
{
        for (int i = 0; i < 4; i++) {
                v1[i] = s1.vi[i]; v2[i] = s1.vi[i] / 4;
                v3[i] = ar2[i];
        }
}
cl cl::elab1(char ar1[], st2 s2)
{
        st1 s1;
        for (int i = 0; i < 4; i++) s1.vi[i] = ar1[i] + i;
        cl cla(s1);
        for (int i = 0; i < 4; i++) cla.v3[i] = s2.vd[i];
        return cla;
}
```

2. Colleghiamo al sistema delle periferiche PCI di tipo `ce`, con vendorID `0xedce` e deviceID `0x1234`. Ogni periferica `ce` usa 32 byte nello spazio di I/O a partire dall'indirizzo base specificato nel registro di configurazione BAR0, sia b.

   La periferiche `ce` sono periferiche di ingresso in grado di operare in PCI Bus Mastering. Sono inoltre in grado di eseguire tutte le necessarie trasformazioni da indirizzi virtuali a fisici, utilizzando le stesse strutture dati della MMU del processore, purchè non incontrino bit P a zero durante la traduzione.

   I registri accessibili al programmatore, tutti di 4 byte, sono i seguenti:

   1. **VPTRHI** (indirizzo b): parte più significativa dell'indirizzo virtuale di destinazione (sempre 0 nei sistemi a 32bit);
   2. **VPTRLO** (indirizzo $b + 4$): parte meno significativa dell'indirizzo virtuale di destinazine;
   3. **CNT** (indirizzo $b + 8$): numero di byte da trasferire;
   4. **CR3** (indirizzo $b + 12$): indirizzo fisico del direttorio (32bit) o tabella di livello 4 (64bit);
   5. **STS** (indirizzo $b + 16$): registro di stato;
   6. **CMD** (indirizzo $b + 20$): registro di comando.

   Ogni volta che si scrive il valore 1 nel registro CMD, la periferica tenta di scrivere CNT byte in memoria a partire dall'indirizzo virtuale contenuto in VPTRHI, VPTRLO. Gli indirizzi verranno tradotti utilizzando la tabella di corrispondenza puntata dal registro CR3 dell'interfaccia. Se l'interfaccia incontra degli errori durante la traduzione (per es. un bit P a zero), interrompe il traferimento e setta il bit 2 di STS. In ogni caso la periferica invia una richiesta di interruzione al completamento dell'operazione (o perché non ha più byte da trasferire, o perché ha riscontrato un errore).

   Le interruzioni sono sempre abilitate. La lettura del registro di stato funziona da risposta alle richieste di interruzione.

   Modificare i file `io.s` e `io.cpp` in modo da realizzare la primitiva

   ```
   bool cedmaread(natl id, natl quanti, char *buf)
   ```

   che permette di leggere `quanti` byte dalla periferica numero `id` (tra quelle di questo tipo), copiandoli nel buffer `buf`. La primitiva restituisce `false` se il trasferimento è stato interrotto per errori, `true` altrimenti.

   Controllare tutti i problemi di Cavallo di Troia.

   Per descrivere le periferiche `ce` aggiungiamo le seguenti strutture dati al modulo I/O:

   ```
   struct des_ce {
       natw iVPTRHI, iVPTRLO, iCNT, iCR3, iSTS, iCMd;
       natl sync;
       natl mutex;
       bool error;
   };
   des_ce array_ce[MAX_CE];
   natl next_ce;
   ```

   La struttura `des_ce` descrive una periferica di tipo `ce` e contiene al suo interno gli indirizzi dei vari registri, l'indice di un semaforo inizializzato a zero (`sync`), l'indice di un semaforo inizializzato a 1 (`mutex`) e un booleano per memorizzare il successo o fallimento dell'ultima operazione.

   I primi `next_ce` elementi del vettore `array_ce` contengono i descrittori, opportunamente inizializzati, delle periferiche di tipo `ce` effettivamente rilevate in fase di avvio del sistema. Ogni periferica è identificata dall'indice del suo descrittore.

   Nota: il modulo sistema mette a disposizione del modulo di I/O le primitive

```
natl resident(void addr, natl quanti);
void nonresident(natl id);
```

La primitiva `resident()` permette di caricare in memoria e rendere temporaneamente non rimpiazzabili tutte le pagine virtuali (e le relative tabelle) che contengono gli indirizzi da `addr` a `addr+quanti` (escluso). La primitiva restituisce un identificatore di operazione che può essere successivamente passato a `nonresident()` per rendere nuovamente rimpiazzabili le pagine (e le tabelle) in questione. In caso di errore restituisce `0xffffffff`.

```
/**
 * File: cc.h
 *       Contains the declaration for the data structures used in the exercise.
 *
 * Author: Rambod Rahmani <rambodrahmani@autistici.org>
 *         Created on 17/09/2019.
 */

#include <iostream>

using namespace std;

struct st1
{
    int vi[4];
};

struct st2
{
    char vd[4];
};

class cl
{
    char v1[4];
    char v3[4];
    long v2[4];

public:
    cl(st1 ss);

    cl(st1& s1, int ar2[]);

    cl elab1(char ar1[], st2 s2);

        void stampa()
    {
        for (int i = 0; i < 4; i++)
        {
            cout << (int)v1[i] << ' ';
        }
        cout << endl;

        for (int i = 0; i < 4; i++)
        {
            cout << (int)v2[i] << ' ';
        }
        cout << endl;

        for (int i = 0; i < 4; i++)
        {
            cout << (int)v3[i] << ' ';
        }
        cout << endl << endl;
    }
};
```

```
/**
 * File: es1.cpp
 *       Contains the C++ code to be translated into Assembly (es1.s file).
 *
 * Author: Rambod Rahmani <rambodrahmani@autistici.org>
 *         Created on 17/09/2019.
 */

#include "cc.h"

cl::cl(st1 ss)
{
    for (int i = 0; i < 4; i++)
    {
        v1[i] = ss.vi[i];
        v2[i] = ss.vi[i] / 2;
        v3[i] = 2 * ss.vi[i];
    }
}

cl::cl(st1& s1, int ar2[])
{
    for (int i = 0; i < 4; i++)
    {
        v1[i] = s1.vi[i];
        v2[i] = s1.vi[i] / 4;
        v3[i] = ar2[i];
    }
}

cl cl::elab1(char ar1[], st2 s2)
{
    st1 s1;

    for (int i = 0; i < 4; i++)
    {
        s1.vi[i] = ar1[i] + i;
    }

    cl cla(s1);

    for (int i = 0; i < 4; i++)
    {
        cla.v3[i] = s2.vd[i];
    }

    return cla;
}
```

```
#*****************************************************************************
# File: es1.s
#       Contains the Assembly translation for es1.cpp.
#
# Author: Rambod Rahmani <rambodrahmani@autistici.org>
#         Created on 14/09/2019.
#*****************************************************************************

#-----------------------------------------------------------------------------
.TEXT
.GLOBAL _ZN2clC1E3st1                                        # cl::cl(st1 ss)
#-----------------------------------------------------------------------------
# activation record:
# ------------------
#  i            -36
#  ss [MSB]     -32
#  ss [LSB]     -16
#  &this        -8
#  %rbp         0
#-----------------------------------------------------------------------------
_ZN2clC1E3st1:
# set stack locations labels:
    .set this, -8
    .set ss,   -32
    .set i,    -36

# prologue: activation record:
    pushq %rbp
    movq  %rsp, %rbp
    subq  $40, %rsp                 # reserve stack space for actual arguments

# copy actual arguments to the stack:
    movq %rdi, this(%rbp)
    movq %rsi, ss(%rbp)
    movq %rdx, ss+8(%rbp)

# for loop initialization:
    movl $0, i(%rbp)                # i = 0

for:
    cmpl $4, i(%rbp)                # check if i < 4
    jge  finefor                    # end for loop (i >= 4)

# for loop body:
    movq   this(%rbp), %rdi         # &this -> %rdi
    movslq i(%rbp), %rcx            # i -> %rcx
    leaq   ss(%rbp), %rsi           # &ss -> %rsi
    movslq (%rsi, %rcx, 4), %rax    # ss.vi[i] -> %rax
    movb   %al, (%rdi, %rcx, 1)     # v1[i] = ss.vi[i]
    sar    $1, %rax                 # ss.vi[i]/2 -> %rax
    movq   %rax, 8(%rdi, %rcx, 8)   # v2[i] = ss.vi[i]/2
    movslq (%rsi, %rcx, 4), %rax    # ss.vi[i] -> %rax
    sal    $1, %rax                 # 2*ss.vi[i] -> %rax
    movb   %al, 4(%rdi, %rcx, 1)    # v3[i] = 2*ss.vi[i]

    incl i(%rbp)                    # i++
    jmp  for                        # loop again

finefor:

    movq  this(%rbp), %rax          # return initialized object address
    leave                           # movq %rbp, %rsp; popq %rbp
    ret

#-----------------------------------------------------------------------------
.GLOBAL _ZN2clC1ER3st1Pi                          # cl::cl(st1& s1, int ar2[])
#-----------------------------------------------------------------------------
# activation frame:
# -----------------
#  i            -28
```

```
#  &ar2         -24
#  &s1          -16
#  &this        -8
#  %rbp         0
#-------------------------------------------------------------------------------
_ZN2clC1ER3st1Pi:
# set stack locations labels:
    .set this, -8
    .set s1,   -16
    .set ar2,  -24
    .set i,    -28

# prologue: activation frame:
    pushq %rbp
    movq  %rsp, %rbp
    subq  $32, %rsp

# copy actual arguments to the stack:
    movq %rdi, this(%rbp)
    movq %rsi, s1(%rbp)
    movq %rdx, ar2(%rbp)

# for loop initialization:
    movl $0, i(%rbp)                # i = 0

for1:
    cmpl $4, i(%rbp)                # check if i < 4
    jge  finefor1                   # end for loop (i >= 4)

# for loop body:
    movq   this(%rbp), %rdi         # &this -> %rdi
    movslq i(%rbp), %rcx            # i -> %rcx
    movq   s1(%rbp), %rsi           # &s1 -> %rsi
    movslq (%rsi, %rcx, 4), %rax    # s1.vi[i] -> %rax
    movb   %al, (%rdi, %rcx, 1)     # v1[i] = s1.vi[i]
    sar    $2, %rax                 # s1.vi[i]/4 -> %rax
    movq   %rax, 8(%rdi, %rcx, 8)   # v2[i] = s1.vi[i]/4
    movq   ar2(%rbp), %rsi          # &ar2 -> %rsi
    movl   (%rsi, %rcx, 4), %ebx    # ar2[i] -> %ebx
    movl   %ebx, 4(%rdi, %rcx, 1)   # v3[i] = ar2[i]

    incl i(%rbp)                    # i++
    jmp  for1                       # loop again

finefor1:

    leave                           # movq %rbp, %rsp; popq %rbp
    ret

#-------------------------------------------------------------------------------
.GLOBAL _ZN2cl5elab1EPc3st2                  # cl cl::elab1(char ar1[], st2 s2)
#-------------------------------------------------------------------------------
# activation frame:
# -----------------
#  cla.v1/v3    -88
#  cla.v2[0]    -80
#  cla.v2[1]    -72
#  cla.v2[2]    -64
#  cla.v2[3]    -56
#  s1 [MSB]     -48
#  s1 [LSB]     -40
#  i            -32
#  s2           -28
#  &ar1         -24
#  &this        -16
#  &indo        -8
#  %rbp         0
#-------------------------------------------------------------------------------
_ZN2cl5elab1EPc3st2:
# set stack locations labels:
```

```
    .set indo, -8
    .set this, -16
    .set ar1,  -24
    .set s2,   -28
    .set i,    -32
    .set s1,   -48
    .set cla,  -88

# prologue: activation frame:
    pushq %rbp
    movq  %rsp, %rbp
    subq  $88, %rsp                 # reserve stack space for actual arguments

# copy actual arguments to the stack:
    movq %rdi, indo(%rbp)
    movq %rsi, this(%rbp)
    movq %rdx, ar1(%rbp)
    movl %ecx, s2(%rbp)

# for loop 1 initialization:
    movl $0, i(%rbp)                # i = 0

for2:
    cmpl $4, i(%rbp)                # check if i < 4
    jge  finefor2                   # end for loop (i >= 4)

# for loop 1 body:
    movslq i(%rbp), %rcx            # i -> %rcx
    movq   ar1(%rbp), %rdi          # &ar1 -> %rdi
    movb   (%rdi, %rcx, 1), %al     # ar1[i] -> %al
    movsbq %al, %rax                # ar1[i] -> %rax
    addq   %rcx, %rax               # ar1[i] + i -> %rax
    leaq   s1(%rbp), %rsi           # &s1 -> %rsi
    movl   %eax, (%rsi, %rcx, 4)    # s1.vi[i] = ar1[i] + i;

    incl i(%rbp)                    # i++
    jmp  for2                       # loop again

finefor2:

# cl cla(s1):
    leaq cla(%rbp), %rdi
    movq s1(%rbp), %rsi
    movq s1+8(%rbp), %rdx
    call _ZN2clC1E3st1

# for loop 2 initialization
    movl $0, i(%rbp)                # i = 0

for3:
    cmpl $4, i(%rbp)                # check if i < 4
    jge  finefor3                   # end for loop (i >= 4)

# for loop 2 body
    movslq i(%rbp), %rcx            # i -> %rcx
    leaq   cla(%rbp), %rdi          # &cla -> %rdi
    leaq   s2(%rbp), %rsi           # &s2 -> %rsi
    movb   (%rsi, %rcx, 1), %al     # s2.vd[i] -> %al
    movb   %al, 4(%rdi, %rcx, 1)    # cla.v3[i] = s2.vd[i]

    incl i(%rbp)                    # i++
    jmp  for3                       # loop again

finefor3:

# copy cla into the memory space addressed by indo
    leaq cla(%rbp), %rsi            # source address
    movq indo(%rbp), %rdi           # destination address
    movabsq $5, %rcx                # repetitions
    rep movsq                       # execute transfer
```

```
# return intialized object address
    movq indo(%rbp), %rax

    leave                           # movq %rbp, %rsp; popq %rbp
    ret

#*****************************************************************************
```

```
/**
 * File: prova1.cpp
 *       This file contains a developer harness test for es1.s.
 *
 *       Compile with:
 *          g++ -o es1 -fno-elide-constructors es1.s prova1.cpp
 *
 *       Test your result with:
 *          ./es1 | diff - es1.out
 *
 * Author: Rambod Rahmani <rambodrahmani@autistici.org>
 *         Created on 17/09/2019.
 */

#include "cc.h"

/**
 * Developer harness test.
 *
 * @param   argc    command line arguments counter.
 * @param   argv    command line arguments.
 *
 * @return          execution exit code.
 */
int main(int argc, char * argv[])
{
    st1 s1 = { 1,2,3,4 };

    st2 s2 = { 5,6,7,8 };

    char a1[4] = { 11,12,13,14 };

    int a2[4] = {15,16,17,18 };

    cl cla1(s1); cla1.stampa();

    cl cla2(s1, a2); cla2.stampa();

    cla1 = cla2.elab1(a1, s2); cla1.stampa();

    //system("pause");
}
```

```
1 2 3 4
0 1 1 2
2 4 6 8

1 2 3 4
0 0 0 1
15 16 17 18

11 13 15 17
5 6 7 8
5 6 7 8
```

```
// EXTENSION 2017-02-24

#define  MAX_RES  10

// EXTENSION 2017-02-24

// EXTENSION 2017-02-24

/**
 * User module primitives interrupt types definitions.
 */

/**
 * natl resident(addr base, natq size);
 */
#define TIPO_RES     0x59

/**
 * void nonresident(natl id);
 */
#define TIPO_NONRES  0x5a

/**
 * natq countres();
 */
#define TIPO_CRES    0x5b

// EXTENSION 2017-02-24
```

```
// EXTENSION 2017-02-24

/**
 *
 */
extern "C" natq countres();

// EXTENSION 2017-02-24

// EXTENSION 2017-02-24

/**
 * Reads the required amount of bytes from the specified CE device.
 *
 * @param  id      CE device ID;
 * @param  quanti  number of bytes to be transferred;
 * @param  buf     destination buffer virtual address.
 */
extern "C" bool cedmaread(natl id, natl quanti, char *buf);

// EXTENSION 2017-02-24
```

```
# EXTENSION 2017-02-24

#-------------------------------------------------------------------------------
.GLOBAL countres                                    # extern "C" natq countres();
#-------------------------------------------------------------------------------
countres:
    int $TIPO_CRES
    ret

#-------------------------------------------------------------------------------
.GLOBAL cedmaread  # extern "C" bool cedmaread(natl id, natl quanti, char *buf);
#-------------------------------------------------------------------------------
cedmaread:
    int $IO_TIPO_CEDMAREAD
    ret

# EXTENSION 2017-02-24
```

```
# EXTENSION 2017-02-24

#--------------------------------------------------------------------------------
.global resident
#--------------------------------------------------------------------------------
resident:
    nt $TIPO_RES
    ret

#--------------------------------------------------------------------------------
.global nonresident
#--------------------------------------------------------------------------------
nonresident:
    int $TIPO_NONRES
    ret

#--------------------------------------------------------------------------------
.global readCR3                # returns che content of the CR3 register in %eax
#--------------------------------------------------------------------------------
readCR3:
    movq %cr3, %rax
    retq

# EXTENSION 2017-02-24


# SOLUTION 2017-02-24

    # fill IDT gate for IO_TIPO_CEDMAREAD interrupts
    fill_io_gate    IO_TIPO_CEDMAREAD   a_cedmaread

# SOLUTION 2017-02-24

# SOLUTION 2017-02-24

#--------------------------------------------------------------------------------
.EXTERN c_cedmaread                                          # C++ implementation
#--------------------------------------------------------------------------------
a_cedmaread:
    .cfi_startproc
    .cfi_def_cfa_offset 40
    .cfi_offset rip, -40
    .cfi_offset rsp, -16
    cavallo_di_troia %rdx           # check destination buffer starting address
    cavallo_di_troia2 %rdx %rsi     # check destination buffer length
    call c_cedmaread                # call C++ implementation
    iretq                           # return from interrupt
    .cfi_endproc

# SOLUTION 2017-02-24
```

```
// EXTENSION 2017-02-24

/**
 * Checks if the address virtual pages [base, size) are permanent or can be make
 * permanent in the physical memory.
 *
 * @param  base  virtual pages starting address
 * @param  size  virtual pages length
 *
 * @return  an ID which can be used as argument of the nonresident(natl id)
 *          primitive to undo the operation.
 */
extern "C" natl resident(addr base, natq size);

/**
 * Undoes the operations performed by the resident(addr, natq) primitive.
 */
extern "C" void nonresident(natl id);

// EXTENSION 2017-02-24


// EXTENSION 2017-02-24

/**
 * Maximum number of CE devices to be initialized at boot.
 */
static const int MAX_CE = 16;

/**
 * CE device descriptor. Each CE PCI device (identified by the vendor ID 0xedce
 * and device ID 0x1234) occupies 32 bytes in the I/O space starting from the
 * address specified in the BAR0 configuration register. Theese PCI devices are
 * capable of transferring data in PCI Bus Mastering.
 */
struct des_ce
{
    // destination buffer virtual address MSB
    ioaddr iVPTRHI;

    // destination buffer virtual address LSB
    ioaddr iVPTRLO;

    // number of bytes to be transferred
    ioaddr iCNT;

    // directory (32 bit) / tab4 (64 bit) physical address to be used to
    // translate the provided destination buffer virtual address
    ioaddr iCR3;

    // status register: reading it is considered as interrupt request ak
    // in case of errors the transfer is interrupted and the status register bit
    // n. 2 is set to 1
    ioaddr iSTS;

    // command regsiter address: writing 1 here starts the transfers
    ioaddr iCMD;

    // synchronization semaphore
    natl sync;

    // mutex semaphore
    natl mutex;

    // last transfer operation result
    bool error;
};

/**
 * Initialized CE devices array.
```

```
 */
des_ce array_ce[MAX_CE];

/**
 * Next CE device to be initialized.
 */
natl next_ce;

/**
 * Reads the content of the CPU CR3 register.
 */
extern "C" addr readCR3();

// EXTENSION 2017-02-24


// SOLUTION 2017-02-24

/**
 * Transfers from the CE device having the provided id the number of specified
 * bytes into the destination buffer.
 *
 * @param  id      CE device ID;
 * @param  quanti  number of bytes to be transferred;
 * @param  buf     destination buffer virtual address.
 */
extern "C" bool c_cedmaread(natl id, natl quanti, char *buf)
{
    // tranfer result
    bool rv;

    // check if the given CE device id is valid
    if (id >= next_ce)
    {
        // print warning log message
        flog(LOG_WARN, "Invalid CE device: %d", id);

        // abort calling process
        abort_p();
        }

    // retrieve CE device descriptor
    des_ce *ce = &array_ce[id];

    // wait for the CE device mutex sempahore
    sem_wait(ce->mutex);

    // print log message for debugging purposes
    flog(LOG_DEBUG, "virt %p len %d", buf, quanti);

    // reset error flag to false before starting a new transfer
    ce->error = false;

    // check if the provided buffer virtual address is or can be made resident
    // in memory
    natl rid = resident(buf, quanti);
    if (rid == 0xffffffff)
    {
        // if not, the transfer can not be completed
        rv = false;
        goto out;
    }

    // write destination buffer virtual address MSB
    outputl((natq)buf, ce->iVPTRLO);

    // write destination buffer virtual address LSB
    outputl((natq)buf >> 32, ce->iVPTRHI);
```

```
    // write number of bytes to be transferred
    outputl(quanti, ce->iCNT);

    // retrieve CPU CR3 register content and write it in the
    // CE device CR3 register
    outputl((natq)readCR3(), ce->iCR3);

    // clear status register: might contain previous errors
    outputl(0, ce->iSTS);

    // write 1 in the command register: start transfers
    outputl(1, ce->iCMD);

    // wait for the sync sempahore: DMA transfers completed
    sem_wait(ce->sync);

    // check if there was any error during DMA transfers
    rv = !ce->error;

    // undo the previous resident operation
    nonresident(rid);

out:
    // notify mutex semaphore
    sem_signal(ce->mutex);

    // return transfer result
    return rv;
}

/**
 * Called every time the CE device having the given id sends an interrupt
 * request.
 */
extern "C" void estern_ce(int id)
{
    // retrieve CE device descriptor
    des_ce *ce = &array_ce[id];

    // status register buffer byte
    natl b;

    // without the infinite for loop when the wfi() function sends the EOI and
    // schedules a new process, the next time this process is waken the function
    // will end
    for (;;)
    {
        // read from the CE device status register: interrupt request ak
        inputl(ce->iSTS, b);

        // check bit n. 2 of the status register for errors
        ce->error = (b & 2);

        // notify synchronization semaphore
        sem_signal(ce->sync);

        // send EOI and schedule a new process
        wfi();
    }
}

// SOLUTION 2017-02-24

// EXTENSION 2017-02-24

/**
 * Initializes the CE devices available on the PCI bus 0. A maximum number of
 * MAX_CE device can be initializes which descriptors will be place in the
 * array_ce array. All other CE devices will be ignored.
 */
```

```
bool ce_init()
{
    // loop through PCI device on bus 0 having the required vendor and device ID
    for (natb bus = 0, dev = 0, fun = 0;
         pci_find_dev(bus, dev, fun, 0xedce, 0x1234);
         pci_next(bus, dev, fun))
    {
        // check if the maximum number of CE devices is not exceeded
        if (next_ce >= MAX_CE)
        {
            // if so, print a warning log message
            flog(LOG_WARN, "Too many CE devices");

            // just return to the caller
            break;
        }

        // retrieve CE device descriptor
        des_ce *ce = &array_ce[next_ce];

        // retrieve BAR0 register content
        natw base = pci_read_confl(bus, dev, fun, 0x10);

        // retrieve base address
        base &= ~0x1;

        // set VPTRHI register address: base address
        ce->iVPTRHI = base;

        // set VPTRLO register address: base + 4
        ce->iVPTRLO = base + 4;

        // set CNT register address: base + 8
        ce->iCNT = base + 8;

        // set CR3 register address: base + 12
        ce->iCR3 = base + 12;

        // set status register address: base + 16
        ce->iSTS = base + 16;

        // set command register address: base + 20
        ce->iCMD = base + 20;

        // initialize synchronization semaphore
        ce->sync = sem_ini(0);

        // initialize mutex semaphore
        ce->mutex = sem_ini(1);

        // retrieve device APIC pin number
        natb irq = pci_read_confb(bus, dev, fun, 0x3c);

        // activate external process for CE device interrupt requests
        activate_pe(estern_ce, next_ce, PRIO, LIV, irq);

        // print info log message with the CE device details
        flog(LOG_INFO, "ce%d %2x:%1x:%1x base=%4x IRQ=%d", next_ce, bus, dev, fun, base,
irq);

        // increase initialized CE devices counter
        next_ce++;
    }

    // return CE devices initialization successful
    return true;
}

// EXTENSION 2017-02-24
```

```
////////////////////////////////////////////////////////////////////////////
//                  INIZIALIZZAZIONE DEL SOTTOSISTEMA DI I/O                  //
////////////////////////////////////////////////////////////////////////////

// inizializza i gate usati per le chiamate di IO
//
extern "C" void fill_io_gates(void);

extern "C" natl end;
// eseguita in fase di inizializzazione
//
extern "C" void cmain(int sem_io)
{

        fill_io_gates();
        mem_mutex = sem_ini(1);
        if (mem_mutex == 0xFFFFFFFF) {
                flog(LOG_ERR, "impossible creare semaforo mem_mutex");
                abort_p();
        }
        unsigned long long end_ = (unsigned long long)&end;
        end_ = (end_ + DIM_PAGINA - 1) & ~(DIM_PAGINA - 1);
        heap_init((void *)end_, DIM_IO_HEAP);
        if (!console_init())
                abort_p();
        if (!com_init())
                abort_p();
        if (!hd_init())
                abort_p();

// EXTENSION 2017-02-24

    // initialize CE devices
    if (!ce_init())
    {
        // abort the calling process in case of errors
        abort_p();
    }
// EXTENSION 2017-02-24

        sem_signal(sem_io);
        terminate_p();
}
```

```
# EXTENSION 2017-02-24

    carica_gate  TIPO_CRES  a_countres  LIV_UTENTE

# EXTENSION 2017-02-24

        // primitive per il livello I/O
        carica_gate     TIPO_APE        a_activate_pe   LIV_SISTEMA
        carica_gate     TIPO_WFI        a_wfi           LIV_SISTEMA
        carica_gate     TIPO_FG         a_fill_gate     LIV_SISTEMA
        carica_gate     TIPO_P          a_panic         LIV_SISTEMA
        carica_gate     TIPO_AB         a_abort_p       LIV_SISTEMA
        carica_gate     TIPO_TRA        a_trasforma     LIV_SISTEMA

# EXTENSION 2017-02-24

    carica_gate  TIPO_NONRES  a_nonresident  LIV_SISTEMA
    carica_gate  TIPO_RES     a_resident     LIV_SISTEMA

# EXTENSION 2017-02-24


# EXTENSION 2017-02-24

#-----------------------------------------------------------------------------
.EXTERN c_countres                                          # natq countres();
#-----------------------------------------------------------------------------
a_countres:
    .cfi_startproc
    .cfi_def_cfa_offset 40
    .cfi_offset rip, -40
    .cfi_offset rsp, -16
    call c_countres                 # call C++ implementation
    iretq                           # return from interrupt
    .cfi_endproc

#-----------------------------------------------------------------------------
.EXTERN c_nonresident                                  # void nonresident(natl id);
#-----------------------------------------------------------------------------
a_nonresident:
    .cfi_startproc
    .cfi_def_cfa_offset 40
    .cfi_offset rip, -40
    .cfi_offset rsp, -16
    call salva_stato                # save current process state
    call c_nonresident              # call C++ implementation
    call carica_stato               # load new process state
    iretq                           # return from interrupt
    .cfi_endproc

#-----------------------------------------------------------------------------
.extern c_resident                          # natl resident(addr base, natq size);
#-----------------------------------------------------------------------------
a_resident:
    .cfi_startproc
    .cfi_def_cfa_offset 40
    .cfi_offset rip, -40
    .cfi_offset rsp, -16
    call salva_stato                # save current process state
    cavallo_di_troia %rdi           # check addressed pages base
    cavallo_di_troia2 %rdi %rsi     # check addressed pages length
    call c_resident                 # call C++ implementation
    call carica_stato               # load new process state
    iretq                           # return from interrupt
    .cfi_endproc

# EXTENSION 2017-02-24
```

```
////////////////////////////////////////////////////////////////////////////
//                          PAGINE FISICHE                                //
////////////////////////////////////////////////////////////////////////////

// avremo un descrittore di pagina fisica per ogni pagina fisica della parte
// M2.  Lo scopo del descrittore e' di contenere alcune informazioni relative
// al contenuto della pagina fisica descritta. Tali informazioni servono
// principalmente a facilitare o rendere possibile il rimpiazzamento del
// contenuto stesso.
struct des_frame {
        int     livello;        // 0=pagina, -1=libera

// EXTENSION 2017-02-24

        natl    residente;      // pagina residente sse > 0

// EXTENSION 2017-02-24

        // identificatore del processo a cui appartiene l'entita'
        // contenuta nel frame.
        natl    processo;
        natl    contatore;      // contatore per le statistiche
        // blocco da cui l'entita' contenuta nel frame era stata caricata
        natq    ind_massa;
        // per risparmiare un po' di spazio uniamo due campi che
        // non servono mai insieme:
        // - ind_virtuale serve solo se il frame e' occupato
        // - prossimo_libero serve solo se il frame e' libero
        union {
                // indirizzo virtuale che permette di risalire al
                // descrittore che punta all'entita' contenuta nel
                // frame. Per le pagine si tratta di un qualunque
                // indirizzo virtuale interno alla pagina. Per le
                // tabelle serve un qualunque indirizzo virtuale la
                // cui traduzione passa dalla tabella.
                vaddr   ind_virtuale;
                des_frame*      prossimo_libero;
        };
};


// EXTENSION 2017-02-24

natq pf_count = 0;

// EXTENSION 2017-02-24

void stat();

bool c_routine_pf()
{
        vaddr ind_virt = readCR2();
        natl proc = esecuzione->id;

        stat();

// EXTENSION 2017-02-24

        pf_count++;
        //flog(LOG_DEBUG, "page fault a %p", ind_virt);

// EXTENSION 2017-02-24

        for (int i = 3; i >= 0; i--) {
                tab_entry d = get_des(proc, i + 1, ind_virt);
                bool bitP = extr_P(d);
                if (!bitP) {
                        des_frame *df = swap(proc, i, ind_virt);
                        if (!df)
```

```
                                return false;
                }
        }
        return true;
}

// EXTENSION 2017-02-24

/**
 *
 */
struct res_des
{
    addr base;
    natq size;
    natl proc;
};

/**
 *
 */
res_des array_res[MAX_RES];

/**
 *
 */
natl alloca_res(addr base, natq size)
{
    res_des *r = 0;

    natl id = 0xffffffff;

    for (int i = 0; i < MAX_RES; i++)
    {
        r = &array_res[i];

        if (r->proc == 0)
        {
            id = i;

            break;
        }
    }

    if (r)
    {
        r->base = base;

        r->size = size;

        r->proc = esecuzione->id;
    }

    return id;
}

/**
 *
 */
bool res_valido(natl id)
{
    return (id < MAX_RES) && (esecuzione->id == array_res[id].proc);
}

/**
 *
 */
void rilascia_res(natl id)
{
    array_res[id].proc = 0;
```

```
}

/**
 *
 */
extern "C" natq c_countres()
{
    natq c = 0;

    for (natq i = 0; i < N_DF; i++)
    {
        des_frame* ppf = &vdf[i];

        if (ppf->livello >= 0 && ppf->residente > 0)
        {
            c++;
        }
    }

    return c | (pf_count << 32);
}

// decrementa i campi resident per tutte le tabelle o pagine
// di livello i che coprono gli indirizzi [base, stop)
void undo_res(natq base, natq stop, int i)
{
    natl proc = esecuzione->id;

    // per capire quali tabelle/pagine di livello j dobbiamo
    // rendere non residenti calcoliamo:
    // vi: l'indirizzo virtuale di partenza della prima pagina di livello
    //     i+1 che interseca [base, base+size)
    // vf: l'indirizzo virtuale di partenza della prima pagina di livello
    //     i+1 che si trova oltre a e non interseca [base, base+size)
    natq mask = dim_region(i) - 1;
    vaddr vi = base & ~mask;
    vaddr vf = (stop + mask) & ~mask;

    for (natq v = vi; v != vf; v += dim_region(i))
    {
        // otteniamo il descrittore che punta a questa tabella/pagina
        natq& d = get_des(proc, i + 1, v);
        // se prima era residente, deve essere presente, quindi
        // possiamo estrarre l'indirizzo fisico e ottenere da questo
        // il puntatore al descrittore di pagina fisica
        des_frame *ppf = descrittore_frame(extr_IND_FISICO(d));
        ppf->residente--;
    }
}

/**
 * @param  base
 * @param  size
 */
extern "C" void c_resident(addr base, natq s)
{
    natl proc = esecuzione->id, id;
    int i;
    natq v, a = (natq)base;
    des_proc *self = des_p(proc);

    self->contesto[I_RAX] = 0xFFFFFFFF;

    if (a < ini_utn_p || a + s < a || a + s > fin_utn_p)
    {
        flog(LOG_WARN, "parametri non validi: %p, %p", a, s);
        return;
    }

    for (i = 3; i >= 0; i--)
```

```
    {
                natq mask = dim_region(i) - 1;
                vaddr vi = a & ~mask;
                vaddr vf = (a + s + mask) & ~mask;
                //flog(LOG_DEBUG, "liv %d: vi %p vf %p", i, vi, vf);
                for (v = vi; v != vf; v += dim_region(i)) {
                        natq& d = get_des(proc, i + 1, v);
                        des_frame *ppf;
                        if (!extr_P(d)) {
                                ppf = swap(proc, i, v);
                                if (!ppf)
                                        goto error;
                        } else {
                                ppf = descrittore_frame(extr_IND_FISICO(d));
                        }
                        ppf->residente++;
                }
        }
        id = alloca_res(base, s);
        if (id == 0xffffffff)
                goto error;

        self->contesto[I_RAX] = id;
        return;

error:
        for (int j = 3; j >= i + 1; j--)
                undo_res(a, a + s, j);
        undo_res(a, v, i);
}

/**
 *
 */
extern "C" void c_nonresident(natl id)
{
        res_des *r;

        if (!res_valido(id)) {
                flog(LOG_WARN, "id non valido: %d", id);
                c_abort_p();
                return;
        }
        r = &array_res[id];

        natq a = (natq)r->base;
        natq s = r->size;

        for (int i = 3; i >= 0; i--) {
                undo_res(a, a + s, i);
        }
        rilascia_res(id);
}

// EXTENSION 2017-02-24
```