```
////////////////////////////////////////////////////////////////////////////
//                          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
```