2023-09-13

Definiamo una nuova zona della memoria virtuale di ogni processo, che mappa semplicemente della memoria fisica in lettura/scrittura. Il primi processi, creati direttamente dal nucleo, partono con una zona che contiene solo zeri. I processi possono modificare liberamente la propria zona, e ciascuno vede solo le proprie modifiche. Ogni volta che un processo padre crea un processo figlio (tramite `activate_p()` o `activate_pe()`), il figlio riceve una copia della zona del padre, visibile agli stessi indirizzi e con il contenuto attuale. Da quel punto in poi padre e figlio possono modificare la propria copia indipendentemente.

Invece di copiare davvero tutto il contenuto della zona ad ogni creazione di processo, adottiamo l'ottimizzazione del *Copy-On-Write* (COW): al momento della creazione del processo figlio disabilitiamo le scritture per tutti gli indirizzi appartenenti alla zona del padre, quindi facciamo in modo che la tabella di livello 4 del processo figlio punti direttamente alle tabelle di livello 3 del TRIE del padre che mappano la zona. In questo modo i due processi condivideranno tutte le pagine della zona e tutto il sotto-TRIE che le mappa nelle loro memorie virtuali. Quando il processo padre o figlio cercheranno di scrivere nella zona, la MMU genererà una eccezione di page-fault. Invece di terminare il processo, la routine di gestione dell'eccezione (già in gran parte realizzata) copierà la pagina interessata in una nuova pagina, riabilitarà il permesso di scrittura nella pagina solo per il processo che ha causato il fault e poi farà ripartire il processo, che ripeterà l'istruzione interrotta. Si noti che, per abilitare il permesso di scrittura solo per il processo che ha causato il fault, la routine dovrà anche creare opportune copie delle eventuali tabelle del TRIE che risultassero ancora condivise con altri processi.

Il meccanismo appena descritto comporta che le pagine e le tabelle relative alla zona possono in generale essere condivise da un numero variabile di processi (si pensi a un processo che crea un figlio, che poi ne crea un altro, che ne crea un altro e così via, e si pensi a cosa succede quando uno di questi processi tenta di scrivere in una pagina della zona, prima o dopo aver creato un figlio). Dobbiamo stare attenti a deallocare solo le tabelle e le pagine della zona che non sono più condivise con altri processi. Per sapere quali tabelle o pagine sono condivise, manteniamo un contatore `nshared` per ciascuna di esse (lo mettiamo nel `des_frame` del frame che contiene la tabella o la pagina). Il contatore deve contare il numero di processi attivi che hanno la tabella o la pagina nel proprio TRIE, e deve essere aggiornato opportunamente ogni volta che un processo nasce o muore, oppure viene generato un page fault in scrittura sulla zona. La tabella/pagina va deallocata (`rilascia_tab` o `rilascia_frame`) se e solo se il suo contatore passa a zero.

```
struct des_frame {
    union {
        struct {
            /// numero di entrate valide (se il frame contiene una tabella)
```

```
            natw nvalide;
            /// numero di processi che condividono questo frame (se cow)
            natw nshared;
        };
        /// prossimo frame libero (se il frame è libero)
        natl prossimo_libero;
    };
};

void init_frame()
{
    ...
    memset(voidptr_cast(primo_frame_libero * DIM_PAGINA), 0xaa, N_M2 * DIM_PAGINA);
    ...
}
```

Per realizzare il meccanismo aggiungiamo il campo `nshared` ai descrittori frame e introduciamo le seguenti funzioni:

- `bool crea_cow(paddr dest)` (già realizzata): crea la prima versione della zona, nel TRIE di radice `dest`, e la riempie di zeri.
- `void copia_cow(paddr srd, paddr dest)` (da realizzare): chiamata durante la creazione di un processo, copia la zona dal TRIE di radice `src` al TRIE di radice `dest`, usando la tecnica COW e aggiornando opportunamente i contatori `nshared`;
- `bool aggiorna_cow(vaddr v)` (realizzata in parte): chiamata dalla routine di page fault; aggiorna il TRIE del processo corrente per abilitare la scrittura nella pagina che contiene `v` secondo la tecnica COW; va realizzata la parte che gestisce i contatori `nshared`;
- `void distruggi_cow()` (da realizzare): chiamata durante la distruzione di un processo; rilascia tutte le risorse associate alla zona COW del processo corrente, purché non siano ancora condivise con altri processi.

```
natl pf_counter = 0;    ///< numero di page fault
vaddr last_pf_v;    ///< indirizzo virtuale dell'ultimo page fault
vaddr last_pf_rip;  ///< istruzione che ha causato l'ultimo page fault
natq  last_pf_error;    ///< codice di errore dell'ultimo page fault

extern "C" void gestore_eccezioni(int tipo, natq errore, vaddr rip)
{
    ...
    if (tipo == 14) {
        vaddr v = readCR2();
        pf_counter++; // solo per debug
        if (pf_counter) {
            if (last_pf_v == v && last_pf_rip == rip && last_pf_error == errore) {
```

```
                panic("PAGE FAULT NON RISOLTO");
            }
            last_pf_v = v;
            last_pf_rip = rip;
            last_pf_error = errore;
        }
        if (aggiorna_cow(v))
            return;
    }
    ...
}

const vaddr ini_utn_w = norm(I_UTN_W * PART_SIZE); ///< base di della zona cow
const vaddr fin_utn_w = ini_utn_w + PART_SIZE * N_UTN_W; ///< limite della zona cow

bool crea_cow(paddr dest)
{
    vaddr v = map(dest, ini_utn_w, ini_utn_w + DIM_USR_COW, BIT_US,
            [](vaddr v) {
                paddr f = alloca_frame();
                memset(voidptr_cast(f), 0, DIM_PAGINA);
                // nshared è inizialmente zero per tutti i frame
                return f;
            });
    if (v != ini_utn_w + DIM_USR_COW) {
        unmap(dest, ini_utn_w, v,
            [](vaddr, paddr p, int) {
                rilascia_frame(p);
            });
        return false;
    }
    return true;
}

des_proc* crea_processo(void f(natq), natq a, int prio, char liv)
{
    ...
    copia_cow(readCR3(), p->cr3);
}
void distruggi_processo(des_proc* p)
{
    distruggi_cow();
    ...
}
```

Modificare il file `sistema.cpp` aggiungendo le parti mancanti.

Attenzione: si ricordi che le tabelle hanno anche un proprio contatore, `nvalide`, che conta il numero di entrate con bit P a 1. Quando si copia o modifica una tabella bisogna mantenere la consistenza del contatore `nvalide`. Inoltre, prima di deallocare una tabella, `nvalide` deve essere zero. Funzioni utili per la manipolazione del contatore `nvalide`: `copy_des`, `inc_ref`, `dec_ref`, `set_des`.

```
void copia_cow(paddr src, paddr dest)
{
    for (tab_iter it(src, ini_utn_w, DIM_USR_COW); it; it.next()) {
        tab_entry& e = it.get_e();
        if (!(e & BIT_P))
            fpanic("indirizzo cow %lx non mappato", it.get_v());
        if (e & BIT_RW) {
            e &= ~BIT_RW;
            if (it.get_l() == 1)
                invalida_entrata_TLB(it.get_v());
        }
        paddr f = extr_IND_FISICO(e);
        vdf[f/DIM_PAGINA].nshared++;
    }
    copy_des(src, dest, I_UTN_W, N_UTN_W);
}
bool rilascia_frame_condiviso(paddr f, int liv)
{
    natl other_refs = --vdf[f/DIM_PAGINA].nshared;
    if (!other_refs) {
        if (liv > 1) {
            set_des(f, 0, 512, 0);
            rilascia_tab(f);
        } else {
            rilascia_frame(f);
        }
    }
    return liv == MAX_LIV || !other_refs;
}
bool aggiorna_cow(vaddr v)
{
    if (esecuzione->livello != LIV_UTENTE || v < ini_utn_w || v >= ini_utn_w + DIM_USR_COW)
        return false;

    vaddr b = base(v, 0);
    for (tab_iter it(esecuzione->cr3, b, DIM_PAGINA); it; it.next()) {
        tab_entry& e = it.get_e();
        if (!(e & BIT_P))
            fpanic("indirizzo cow %lx non mappato", b);
```

```
        if (e & BIT_RW)
            continue;
        paddr new_frame;
        paddr old_frame = extr_IND_FISICO(e);
        if (it.get_l() > 1) {
            new_frame = alloca_tab();
            if (!new_frame)
                return false;
            copy_des(old_frame, new_frame, 0, 512);
        } else {
            new_frame = alloca_frame();
            if (!new_frame)
                return false;
            memcpy(voidptr_cast(new_frame),
                   voidptr_cast(old_frame), DIM_PAGINA);
            invalida_entrata_TLB(it.get_v());
        }
        set_IND_FISICO(e, new_frame);
        e |= BIT_RW;

        vdf[new_frame/DIM_PAGINA].nshared = 1;
        rilascia_frame_condiviso(old_frame, it.get_l());
    }
    return true;
}
void distruggi_cow()
{
    tab_iter it(esecuzione->cr3, ini_utn_w, DIM_USR_COW);
    for (it.post(); it; it.next_post()) {
        tab_entry& e = it.get_e();
        if (!(e & BIT_P))
            continue;
        int liv = it.get_l();
        paddr f = extr_IND_FISICO(e);
        if (rilascia_frame_condiviso(f, liv)) {
            e = 0;
            dec_ref(it.get_tab());
        }
    }
}
```