Prova pratica di Calcolatori Elettronici

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

25 giugno 2025

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

```
struct st {
        int  vv2[4];
        char vv1[4];
};
class cl {
        st s;
public:
        cl(char v[]);
        void elab1(st& ss, int d);
        void stampa()
        {
                for (int i = 0; i < 4; i++)
                        cout << (int)s.vv1[i] << ' ';
                cout << '\t';
                for (int i = 0; i < 4; i++)
                        cout << s.vv2[i] << ' ';
                cout << endl;
                cout << endl;
        }
};
```

   Realizzare in Assembler GCC le funzioni membro seguenti.

```
void cl::elab1(st& ss, int d)
{
        for (int i = 0; i < 4; i++) {
                if (d >= ss.vv2[i])
                        s.vv1[i] += ss.vv1[i];
                s.vv2[i] = d - i;
        }
}
```

2. Modifichiamo il nucleo per aggiungere parte del supporto per lo swap-in e swap-out dei processi utente. Un singolo processo utente può registrarsi come "swapper" e ottenere l'accesso alle primitive `swap_out()`, `swap_in()` e `get_swap_ev()`. Introduciamo inoltre il concetto dei processi "incompleti" e "rimossi".

   - Normalmente, la primitiva `activate_p()` fallisce se non riesce a creare la parte utente/privata (pila utente) del nuovo processo. La modifichiamo in modo che termini comunque con successo, ma il nuovo processo si trovi in uno stato "incompleto".

- Lo swapper può, in qualunque momento, chiedere lo swap-out un altro processo invocando la primitiva `swap_out()`, a cui passa l'id del processo vittima P e un buffer in cui la primitiva copia il contenuto attuale della parte utente/privata di P. Il processo P diventa "rimosso".

In qualunque momento, lo swapper può invocare la primitiva `swap_in()` che provvede a completare o ricaricare un processo. Nel caso di ricarica, lo swapper deve passare alla `swap_in()` un buffer con il contenuto da ripristinare.

Sia i processi incompleti che quelli rimossi non possono essere messi in esecuzione, quidi modifichiamo lo schedulatore in modo che li salti. Se c'è un processo swapper registrato, lo schedulatore deve notificarlo dell'esistenza di questi processi. Lo swapper può ricevere queste notifiche invocando la primitiva `get_swap_ev()`, che gli restituisce il pid di un processo incompleto o rimosso che era stato saltato dallo schedulatore.

Per realizzare questo meccanismo aggiungiamo i seguenti campi ai descrittori di processo:

```
memstate_t memstate;
bool scheduled;
```

Il campo `memstate` registra lo stato della memoria utente/privata del processo e può valere: `M_OK` (memoria creata e caricata); `M_INCOMPLETE` (memoria non creata); `M_SWAPPED_OUT` (memoria creata, ma rimossa). Il campo `scheduled` vale true se il processo è stato saltato dallo schedulatore (perché la sua `memstate` non era `M_OK`).

Aggiungiamo inoltre la seguente struttura dati, di cui allochiamo un'unica istanza globale (variabile `swap_info`):

```
struct des_swapper {
natl id;
bool waiting;
struct des_proc *tonotify;
};
```

Il campo `id` contiene il pid dello swapper (`0xFFFFFFFF` se non esistente); il campo `waiting` vale `true` se lo swapper è bloccato in attesa di eventi nella `get_swap_ev()`; il campo `tonotify` è una lista di processi (ordinata per priorità) il cui pid deve essere inviato allo swapper (i processi restano in questa lista, e il loro pid viene re-inviato, fino a quando lo swapper non li completa o ricarica).

Aggiungiamo infine le seguenti primitiva (abortiscono il processo in caso di errore, controllano problemi di Cavallo di Troia):

- `swapper()` (già realizzata): registra il processo invocante come swapper. È un errore se lo swapper esiste già.
- `bool swap_out(natl pid, char *buf)` (già realizzata): esegue lo swap-out del processo `pid`. Se il processo era incompleto, non fa niente. È un errore se viene invocata da un processo non registrato come swapper, se il processo `pid` è un processo di sistema, o è lo swapper stesso; restituisce `false` se il processo non esiste e `true` in tutti gli altri casi.
- `bool swap_in(natl pid, const char *buf)` (realizzata in parte): esegue lo swap-in o il completamento del processo `pid`. Nel caso di completamento, `buf` non è usato e può essere anche `nullptr`. Se necessario reinserisce il processo in coda pronti, gestendo eventuali preemption. È un errore se viene invocata da un processo non registrato come swapper o se il processo `pid` è un processo di sistema; restituisce `false` se il processo non esiste o se la creazione/ripristino della pila fallisce, `true` in tutti gli altri casi.
- `get_swap_ev()` (già realizzata): restituisce il pid di un processo che era stato saltato dallo schedulatore e che risulta ancora incompleto o rimosso. Sospende il chiamante se non ci sono processi in questa condizione. È un errore se viene invocata da un processo non registrato come swapper.

Modificare il file `sistema.cpp` in modo da realizzare le parti mancanti.