2023-06-28

Sincronizzazione

Aggiungiamo al sistema la primitiva

```
bool sem_wait_to(natl sem, natl to);
```

che funziona come la `sem_wait()`, ma in più permette di specificare un *time-out* `to` per il caso in cui il semaforo `sem` non contenga gettoni. Più in dettaglio: se il semaforo `sem` contiene almeno un gettone, il processo lo prende senza bloccarsi; altrimenti si sospende e a quel punto possono succedere due cose:

1. prima che `to` sia trascorso, qualche processo esegue una `sem_signal()` che risveglia il processo; in quel caso il processo prende il gettone e `sem_wait_to()` restituisce `true`;
2. trascorre il tempo `to` senza che nessuno risvegli il processo tramite una `sem_signal()`; in questo caso il processo si risveglia (senza aver preso un gettone da `sem`) e `sem_wait_to()` restituisce `false`.

È ammesso il caso `to==0`, e in quel caso la `sem_wait_to()` si comporta nel seguente modo: se il semaforo contiene un gettone lo prende e restituisce `true`, altrimenti non fa niente e restituisce `false`.

Per realizzare la nuova primitiva agiungiamo un campo `natl waiting_on` ai descrittori dei processi. Il campo contiene un valore diverso da `0xFFFFFFFF` solo se il processo è sospeso in attesa che si verifichi uno dei due eventi elencati sopra. La primitiva `sem_wait_to()`, se necessario, inserisce il processo sia nella coda del semaforo, sia nella coda del timer (`p_sospesi`). Modifichiamo poi la funzione `c_sem_signal()` in modo che gestisca il caso 1, e la funzione `c_driver_td()` in modo che gestisca il caso 2.

```
struct des_proc {
    ...
    /// indice del semaforo su cui il processo è in attesa con timeout (0xffffffff se nessu
    natl waiting_on;
};
void rimozione_lista_attesa(des_proc* p)
{
    richiesta** r;

    for (r = &sospesi; *r && (*r)->pp != p; r = &(*r)->p_rich)
        ;
    if (richiesta* t = *r) {
        if ( (*r = t->p_rich) )
            (*r)->d_attesa += t->d_attesa;
        delete t;
    }
```

```
}
des_proc* crea_processo(void f(natq), natq a, int prio, char liv)
{
    ...
    p->waiting_on = 0xFFFFFFFF;
}
```

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

```
extern "C" void c_sem_signal(natl sem)
{
    // una primitiva non deve mai fidarsi dei parametri
    if (!sem_valido(sem)) {
        flog(LOG_WARN, "sem_signal: semaforo errato: %u", sem);
        c_abort_p();
        return;
    }

    des_sem* s = &array_dess[sem];
    s->counter++;

    if (s->counter <= 0)
    {
        des_proc* lavoro = rimozione_lista(s->pointer);
        if (lavoro->waiting_on != 0xFFFFFFFF) {
            rimozione_lista_attesa(lavoro);
            lavoro->contesto[I_RAX] = true;
            lavoro->waiting_on = 0xFFFFFFFF;
        }
        inspronti();    // preemption
        inserimento_lista(pronti, lavoro);
        schedulatore(); // preemption
    }
}

extern "C" void c_driver_td(void)
{
    inspronti();
    if (sospesi != nullptr) {
        sospesi->d_attesa--;
    }

    while (sospesi != nullptr && sospesi->d_attesa == 0)
    {
        if (sospesi->pp->waiting_on != 0xFFFFFFFF) {
            des_sem *s = &array_dess[sospesi->pp->waiting_on];
            s->counter++;
```

```
            des_proc **p = &s->pointer;
            while (*p != sospesi->pp)
                p = &(*p)->puntatore;
            *p = (*p)->puntatore;
            sospesi->pp->contesto[I_RAX] = false;
            sospesi->pp->waiting_on = 0xFFFFFFFF;
        }
        inserimento_lista(pronti, sospesi->pp);
        richiesta* p = sospesi;
        sospesi = sospesi->p_rich;
        delete p;
    }

    schedulatore();
}

extern "C" void c_sem_wait_to(natl sem, natl to)
{
    // una primitiva non deve mai fidarsi dei parametri
    if (!sem_valido(sem)) {
        flog(LOG_WARN, "semaforo errato: %u", sem);
        c_abort_p();
        return;
    }

    des_sem *s = &array_dess[sem];
    if (to == 0 && s->counter <= 0) {
        esecuzione->contesto[I_RAX] = false;
        return;
    }
    s->counter--;
    if (s->counter < 0) {
        inserimento_lista(s->pointer, esecuzione);
        esecuzione->waiting_on = sem;
        c_delay(to);
    } else {
        esecuzione->contesto[I_RAX] = true;
    }
}
```