# 2024-07-17 - Notifiche da processi utente

Vogliamo permettere ai processi di livello sistema di essere notificati quando si verificano dei particolari eventi all'interno del modulo sistema. Ci limitiamo a considerare solo gli eventi corrispondenti alla terminazione (o abort) di un processo di livello utente. I processi di livello sistema interessati a questo tipo di eventi devono prima *registrarsi* tramite una nuova primitiva `evreg()`; da quel punto in poi verranno *notificati* ogni volta che un processo utente termina (o abortisce).

Per realizzare il meccanismo modifichiamo la primitiva `wfi()` in modo che possa attendere notifiche di eventi, oltre che richieste di interruzione. La primitiva modificata deve restituire un valore con il seguente significato:

- 1: è stata ricevuta una richiesta di interruzione (comportamento normale della `wfi()`);
- 2: è stata ricevuta una notifica di terminazione;
- 3: entrambe le cose (notifica di terminazione e richiesta di interruzione).

Se la `wfi()` restituisce 2 o 3, il processo deve poi *rispondere* alla notifica invocando la primitiva `evget()`, che restituisce l'id del processo terminato. La primitiva `evget()` può essere invocata più volte (anche senza aver prima invocato `wfi()`) e non è mai bloccante: se non ci sono notifiche pendenti si limita a restituire 0.

Il meccanismo, come descritto, impone anche di modificare gli handler, in quanto ora può accadere che un processo invochi `wfi()`, inviando l'EOI all'APIC e bloccandosi, e poi si risvegli a causa di una notifica. Una richiesta di interruzione può dunque arrivare mentre il processo non è bloccato dentro la `wfi()`; in quel caso l'handler non può mettere il processo forzatamente in esecuzione, ma deve limitarsi a settare un flag nel descrittore del processo. Il processo noterà questo flag e agirà di conseguenza la prossima volta che invoca `wfi()`.

Più processi possono registarsi per gli eventi, e ciascuno di essi deve ricevere tutte le notifiche generate dal momento in cui si è registrato in poi. Diciamo che la notifica di un evento è *in corso* se i processi registrati sono stati notificati, ma non hanno ancora risposto tutti. Quando tutti i processi registrati hanno risposto, diciamo che la notifica è *completata*. Un nuovo evento può essere notificato solo dopo che la notifica del precedente è stata completata. Infine, per evitare che gli id dei processi terminati vengano riusati prima che i processi registrati abbiano avuto il tempo di riceverli, i processi terminati vengono distrutti solo al completamento della notifica.

Per realizzare il meccanismo aggiungiumo i seguenti campi al descrittore di processo:

```
struct des_proc {
    ...
```

```
    /// il processo è registrato per la notifica degli eventi
    bool registrato;
    /// il processo ha ricevuto una notifica e non ha ancora chiamato evget()
    bool notificato;
    /// il processo è bloccato nella wfi()
    bool bloccato;
    /// è arrivata una richiesta di interruzione mentre il processo non era bloccato nella w
    bool ricevuto_intr;
};

extern "C" void c_terminate_p(bool logmsg)
{
    des_proc* p = esecuzione;
    if (esecuzione->registrato) {
        while (esecuzione->notificato)
            c_evget();
    }
    if (notify_event(esecuzione)) {
        schedulatore();
        return;
    }
    ...
}
```

Dove: `registrato` è true se il processo è registrato per la notifica degli eventi; `notificato` è true se il processo ha ricevuto una notifica a cui non ha ancora risposto; `bloccato` è true se il processo è bloccato nella `wfi()`; `ricevuto_intr` è true se è arrivata una richiesta di interruzione mentre il processo non era bloccato nella `wfi()`.

```
a_wfi:
    .cfi_startproc
    .cfi_def_cfa_offset 40
    .cfi_offset rip, -40
    .cfi_offset rsp, -16
    call salva_stato

    call maybe_event
    cmp $0, %al
    jne skip_wfi
    call apic_send_EOI
    call schedulatore
skip_wfi:

    call carica_stato
    iretq
```

```
    .cfi_endproc
```

Aggiungiamo inoltre le seguenti variabili globali:

```
des_proc *in_notifica;

/// numero di processi registrati che devono rispondere alla notifica in corso
natq risposte_mancanti;

/// coda dei processi terminati la cui notifica non è ancora stata avviata
des_proc *terminati;
```

Dove: `in_notifica` punta al descrittore del processo (terminato) la cui notifica è ancora in corso (nullptr se non ci sono notifiche in corso); `risposte_mancanti` conta quanti processi registrati devono ancora rispondere alla notifica in corso (0 se non ci sono notifiche in corso); `terminati` è una coda di processi terminati la cui notifica è stata rimandata perché ce n'era già un'altra in corso.

Modifichiamo gli handler e la `wfi()` come descritto e aggiungiamo le seguenti primitive (invocabili solo da livello sistema):

- `bool evreg()`: registra il processo per la ricezione delle notifiche; restituisce false in caso di errore (processo già registrato, notifica in corso);
- `natq evget()`: risponde ad eventuali notifiche; restituisce l'id di un processo terminato, o 0 se non ci sono notifiche o se il processo non era registrato.

```
extern "C" void c_evreg()
{
    esecuzione->contesto[I_RAX] = false;

    if (esecuzione->registrato) {
        flog(LOG_WARN, "evreg: gia' registrato");
        return;
    }
    if (risposte_mancanti) {
        flog(LOG_WARN, "evreg: non e' possibile registrarsi adesso");
        return;
    }

    esecuzione->registrato = true;
    esecuzione->notificato = false;
    esecuzione->bloccato = false;
    esecuzione->ricevuto_intr = false;
    esecuzione->contesto[I_RAX] = true;
}
```

Modificare il file `sistema.cpp` per completare le parti mancanti.

```
/**
 * @brief Chiamata da wfi() per controllare la presenza di notifiche e/o interrupt pendenti
 *
 * Se questa funzione restituisce true, wfi() non bloccherà il processo.
 *
 * @return  true se il processo ha ricevuto una notifica e/o una richiesta di interruzione,
 *      false altrimenti
 */
extern "C" bool maybe_event()
{
    if (!esecuzione->registrato)
        return false;
    esecuzione->contesto[I_RAX] = 0;
    if (esecuzione->notificato)
        esecuzione->contesto[I_RAX] = 2;
    if (esecuzione->ricevuto_intr) {
        esecuzione->ricevuto_intr = false;
        esecuzione->contesto[I_RAX]++;  // 1 o 3
    }
    if (esecuzione->contesto[I_RAX])
        return true;
    esecuzione->bloccato = true;
    return false;
}

bool notify_event(des_proc *src)
{
    // deve essere notificata solo la terminazione dei processi utente
    if (src->livello != LIV_UTENTE)
        return false;

    if (risposte_mancanti) {
        // è in corso un'altra notifica; questa viene rimandata
        inserimento_lista(terminati, src);
        // se è in corso una notifica, ci sono sicuramente processi registrati
        return true;
    }

    for (natq i = 0; i < MAX_PROC; i++) {
        if (!proc_table[i])
            continue;
        des_proc *dst = proc_table[i];
        if (!dst->registrato)
            continue;
        // abbiamo trovato un processo registrato
```

```
            risposte_mancanti++;
            dst->notificato = true;
            if (dst->bloccato) {
                // se è bloccato va risvegliato, in modo che possa
                // rispondere alla notifica invocando evget()
                dst->bloccato = false;
                dst->contesto[I_RAX] = 2;
                inserimento_lista(pronti, dst);
            }
        }
        if (!risposte_mancanti)
            return false;
        in_notifica = src;
        return true;
    }

    extern "C" void c_evget()
    {
        esecuzione->contesto[I_RAX] = 0;
        if (!esecuzione->registrato) {
            flog(LOG_WARN, "evget: processo non registrato");
            return;
        }
        if (!esecuzione->notificato)
            return;
        esecuzione->contesto[I_RAX] = in_notifica->id;
        esecuzione->notificato = false;
        risposte_mancanti--;
        if (!risposte_mancanti) {
            distruggi_processo(in_notifica);
            processi--;
            in_notifica = nullptr;
            if (terminati) {
                des_proc *p = rimozione_lista(terminati);
                inspronti();
                notify_event(p);
                schedulatore();
            }
        }
    }

    extern "C" void notify_intr(int irq)
    {
        des_proc *p = a_p[irq]; // processo esterno associato a irq

        if (!p->registrato || p->bloccato) {
            p->bloccato = false;
```

```
            p->contesto[I_RAX] = 1;
            inspronti();
            esecuzione = p;
        } else {
            p->ricevuto_intr = true;
        }
    }
```