2024-06-07 - Watch di variabili condivise

Aggiungiamo al sistema il meccanismo dei *watch*, tramite il quale un processo può monitorare tutti i cambiamenti del valore di una variabile causati dagli altri processi. Un processo installa un watch specificando l'indirizzo di una variable e la sua dimensione, diventando quindi *watcher* e proprietario del watch. Il watcher si può poi sospendere in attesa che un altro processo modifichi la variabile, e a quel punto si risveglia ricevendo il nuovo valore.

Per realizzare il meccanismo operiamo nel modo seguente:

1. il watch può essere installato solo su variabili che si trovano nello spazio utente condiviso;
2. quando il watch viene installato, il sistema disabilita le scritture sulla pagina che contiene la variabile da osservare;
3. questo comporta che tutti i processi che tentano di scrivere in quella pagina ricevono una eccezione di page fault;
4. il sistema intercetta questo page fault, riabilita le scritture sulla pagina, abilita il Single Step per il processo che stava tentando di scrivere e lo rimette in esecuzione: diciamo che il processo è sotto osservazione (*watched*);
5. quando il processo sotto osservazione riceve l'eccezione di debug (subito dopo aver eseguito l' operazione di scrittura nella pagina), il sistema confronta il nuovo valore della variabile con quello precedente e, se differiscono, lo passa al watcher; disabilita nuovamente le scritture nella pagina e disattiva il Single Step sul processo watched (che smette così di essere sotto osservazione).
6. il passaggio del nuovo valore al watcher è sincrono: il processo resta sospeso fino a quando il watcher non ha ricevuto il valore.

Per assicurarci che, nei punti 3 e 4, solo il processo sotto osservazione possa scrivere nella pagina, il sistema lo fa girare con le interruzioni esterne disabilitate (ovviamente solo per il tempo in cui il processo è sotto osservazione). Per semplicità prevediamo che nel sistema si possa installare un solo watch alla volta. Quando il watcher termina il watch corrente viene disattivato, in modo che un altro processo ne possa installare uno nuovo.

Attenzione: il watcher deve osservare i cambiamenti della variabile osservata nell'ordine in cui si sono verificate, indipendentemente dalla priorità dei processi che le hanno causate. Casi particolari: le scritture del watcher stesso vanno permesse, ma non vanno notificate; i processi che causano page fault o eccezioni di debug non correlate con il meccanismo del watch devono essere abortiti normalmente.

Aggiungiamo i seguenti campi al descrittore di processo:

```
struct des_proc {
    ...
    /// true se il processo sta per scrivere nella zona osservata
    bool being_watched;
    /// valore scritto dal processo sulla zona osservata
    natq new_watch_value;
};

des_proc* crea_processo(void f(natq), natq a, int prio, char liv)
{
    des_proc*   p;          // des_proc per il nuovo processo
    ...
    p->being_watched = false;
    p->new_watch_value = 0;
}
void distruggi_processo(des_proc* p)
{
    delwatch(p);
    ...
}
```

Il campo `being_watched` vale `true` se il processo è sotto osservazione; il campo `new_watch_value` contiene il nuovo valore della variabile scritto dal processo mentre era sotto osservazione.

Aggiungiamo la seguente struttura dati:

```
struct watch_des {
    /// indirizzo virtuale della variabile osservata (se 0, non ci sono watch installati)
    vaddr   v;
    /// dimensione (in byte) della variabile osservata
    natq    size;
    /// id del processo watcher
    natl    watcher_id;
    /// ultimo valore scritto nella variabile osservata
    natq    old_value;
    /// lista su cui si sospende il watcher in attesa dei processi osservati
    des_proc *watcher_waiting;
    /// lista su cui si sospendono i processi osservati in attesa di passare il valore al w
    des_proc *watched_waiting;
} watch_state;

void delwatch(des_proc *p)
{
    watch_des *w = &watch_state;

    if (!w->v || p->id != w->watcher_id)
```

```
        return;

    // riabilitiamo le scritture
    tab_iter it(p->cr3, w->v, w->size);
    while (it.down())
        ;
    it.get_e() |= BIT_RW;
    // resettiamo il watch_state
    w->v = 0;
    w->size = 0;
    w->watcher_id = 0;
    // risvegliamo eventuali processi watched
    while (des_proc *s = rimozione_lista(w->watched_waiting)) {
        s->being_watched = false;
        s->new_watch_value = 0;
        inserimento_lista(pronti, s);
    }

}
```

Il campo `v` contiene l'indirizzo virtuale della variabile osservata (se 0, non ci sono watch installati); il campo `size` contiene la dimensione (in byte) della variabile osservata; il campo `watcher_id` contiene l'id del processo watcher; il campo `old_value` contiene l'ultimo valore scritto nella variable osservata; il campo `watcher_waiting` punta alla testa della lista su cui si sospende il watcher in attesa dei processi osservati; il campo `watched_waiting` punta alla testa della lista su cui si sospendono i processi osservati in attesa di passare il valore al watcher (questa lista non è ordinata per priorità, ma in base all'ordine delle scritture operate dai processi).

```
extern "C" void gestore_eccezioni(int tipo, natq errore, vaddr rip)
{
    ...
    if (tipo == 14 && (errore & PF_WRITE) && handle_watch_pf(readCR2()))
        return;

    if (tipo == 1 && handle_watch_ss())
        return;
    ...
}
```

Infine aggiungiamo le seguenti primitive (abortiscono il processo in caso di errore):

- `bool setwatch(void *ptr, size_t size)` (già realizzata): installa un watch sulla variabile di indirizzo `ptr` e dimensione `size`; è un errore se la variabile non si trova nello spazio utente condiviso o non è accessibile in lettura, o se non ha una dimensione di 1, 2, 4, o 8, o non è allineata natu-

ralmente; restituisce `false` se c'è già un watch installato, `true` altrimenti;

- `natq watch()`: restituisce il prossimo valore della variabile osservata (eventualmente esteso senza segno a 8 byte); è un errore se non ci sono watch installati o se il processo non è il watcher corrente.

```
extern "C" void c_setwatch(void *ptr, natq size)
{
    if (size != 1 && size != 2 && size != 4 && size != 8) {
        flog(LOG_WARN, "setwatch: size %lu non valida", size);
        c_abort_p();
        return;
    }

    vaddr v = int_cast<vaddr>(ptr);
    if (v & (size - 1)) {
        flog(LOG_WARN, "setwatch: indirizzo %lx non allineato a %lu", v, size);
        c_abort_p();
        return;
    }

    if (!c_access(v, size, true, true)) {
        flog(LOG_WARN, "setwatch: intervallo [%lx, %lx) non valido", v, v + size);
        c_abort_p();
        return;
    }

    watch_des *w = &watch_state;
    if (w->v) {
        esecuzione->contesto[I_RAX] = false;
        return;
    }
    w->v = v;
    w->size = size;
    w->watcher_id = esecuzione->id;
    w->old_value = 0;
    memcpy(&w->old_value, voidptr_cast(w->v), w->size);
    tab_iter it(esecuzione->cr3, v, size);
    while (it.down())
        ;
    it.get_e() &= ~BIT_RW;
    esecuzione->contesto[I_RAX] = true;
    flog(LOG_DEBUG, "setwatch: v=%lx size=%lu", w->v, w->size);
}

bool handle_watch_pf(vaddr cr2)
{
```

```
    watch_des *w = &watch_state;

    if (!w->v) {
        // non ci sono watch attivi
        return false;
    }

    if (base(cr2, 0) != base(w->v, 0)) {
        // l'accesso non riguarda il watch corrente
        return false;
    }

    // permettiamo al processo di eseguire la scrittura e lo catturiamo subito dopo
    natq *pila =  ptr_cast<natq>(esecuzione->contesto[I_RSP]);
    pila[2] |= BIT_TF;
    pila[2] &= ~BIT_IF;
    tab_iter it(esecuzione->cr3, w->v, w->size);
    while (it.down())
        ;
    it.get_e() |= BIT_RW;
    // siccome stiamo attivando la scrittura che prima era disabilitata, il
    // TLB causerà sicuramente miss sull'operazione in scrittura e dunque
    // non c'è bisogno di invalidare l'entrata

    esecuzione->being_watched = true;

    return true;
}
```

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

```
/**
 * @brief   Gestisce eventuali eventi watch (single step)
 *
 * @return  true se l'evento è stato riconosciuto e gestito e il
 *      processo non deve essere abortito, false se la gestione
 *      dell'eccezione deve proseguire normalmente (abortendo
 *      il processo)
 */
bool handle_watch_ss()
{
    // se being_watched non è settato si tratta di un processo che ha
    // causato questa eccezione per altri motivi, e deve essere abortito
    if (!esecuzione->being_watched)
        return false;

    esecuzione->being_watched = false;
```

```
    // resettiamo il single step, riabilitiamo le interruzioni esterne e
    // disabilitiamo nuovamente le scritture sulla pagina che contiene
    // la zona da osservare
    natq *pila =  ptr_cast<natq>(esecuzione->contesto[I_RSP]);
    pila[2] &= ~BIT_TF;
    pila[2] |= BIT_IF;
    watch_des *w = &watch_state;
    tab_iter it(esecuzione->cr3, w->v, w->size);
    while (it.down())
        ;
    it.get_e() &= ~BIT_RW;
    // questa volta dobbiamo invalidare l'entrata, perché il TLB potrebbe
    // aver memorizzato il vecchio stato di RW
    invalida_entrata_TLB(w->v);

    // vediamo se il valore del watch è cambiato
    natq cur_value = 0;
    memcpy(&cur_value, voidptr_cast(w->v), w->size);
    if (cur_value == w->old_value) {
        // non è cambiato: non notifichiamo il watcher e lasciamo che
        // il processo prosegua indisturbato
        return true;
    }
    w->old_value = cur_value;
    // gli accessi da parte del watcher stesso vanno ignorati
    if (w->watcher_id == esecuzione->id)
        return true;
    // altrimenti notifichiamo il watcher (gestiamo i casi in cui il watcher
    // è già in attesa oppure no)
    if (w->watcher_waiting) {
        w->watcher_waiting->contesto[I_RAX] = cur_value;
        inspronti();
        inserimento_lista(pronti, w->watcher_waiting);
        w->watcher_waiting = nullptr;
    } else {
        esecuzione->new_watch_value = cur_value;
        inserimento_in_fondo(w->watched_waiting, esecuzione);
    }
    schedulatore();

    return true;
}

extern "C" void c_watch()
{
```

```
    watch_des *w = &watch_state;
    if (!w->v || w->watcher_id != esecuzione->id) {
        flog(LOG_WARN, "watch: chiamata non valida");
        c_abort_p();
        return;
    }
    if (w->watched_waiting) {
        des_proc *p = rimozione_lista(w->watched_waiting);
        esecuzione->contesto[I_RAX] = p->new_watch_value;
        inspronti();
        inserimento_lista(pronti, p);
    } else {
        inserimento_lista(w->watcher_waiting, esecuzione);
    }
    schedulatore();
}
```