Prova pratica di Calcolatori Elettronici

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

3 luglio 2019

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

```
struct st {
        char vv1[4];
        long vv2[4];
};
class cl {
        st s;
public:
        cl(char v[]);
        void elab1(int d, st& ss);
        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.

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

2. Vogliamo fornire ai processi la possibilità di bloccare l'esecuzione di tutti i processi che passano da una certa istruzione. Per far questo forniamo alcune primitive. Con la primitiva `bpadd(vaddr rip)` si

installa un breakpoint (istruzione `int3`, codice operativo `0xCC`) all'indirizzo `rip`. Da quel momento in poi, tutti i processi che passano da `rip` si bloccano e vengono accodati opportunamente. Nel frattempo, usando la primitiva `bpwait()`, un processo può sospendersi in attesa che un qualche altro processo passi dal breakpoint. La primitiva può essere invocata più volte, per attendere tutti i processi che si suppone debbano passare dal breakpoint. Infine, con la primitiva `bpremove()`, si rimuove il breakpoint e si risvegliano tutti i processi che vi si erano bloccati. I processi così risvegliati devono proseguire la loro esecuzione come se non fossero mai stati intercettati.

Prevediamo la seguente limitazione: ad ogni istante, nel sistema ci può essere al massimo un breakpoint installato tramite da `bpadd()`.

Si noti che se un processo esegue `int3` senza che ciò sia richiesto da una primitiva `bpadd()` attiva, il processo deve essere abortito.

Aggiungiamo al nucleo la seguente struttura dati:

```
struct b_info {
    proc_elem *waiting;
    proc_elem *intercepted;
    proc_elem *to_wakeup;
    vaddr rip;
    natb orig;
    bool busy;
} b_info;
```

dove: `waiting` è una coda di processi che hanno invocato `bpwait()` e sono in attesa che qualche processo passi dal breakpoint; `intercepted` è una coda di processi che sono bloccati sul breakpoint e il cui identificatore non è stato ancora restituisto da una `bpwait()`; `to_wakeup` è una coda di processi bloccati sul breakpoint e i cui indentificatori sono stati già restituiti tramite `bpwait()`; `rip` è l'indirizzo a cui è installato il breakpoint; `orig` è il byte originariamente contenuto all'indirizzo `rip`; `busy` vale `true` se c'è un breakpoint installato.

Si modifichino i file `sistema/sistema.s` e `sistema/sistema.cpp` per implementare le seguenti primitive (abortiscono il processo in caso di errore):

- `bool bpadd(vaddr rip)`: (tipo 0x59, già realizzata): se non c'è un altro breakpoint già installato, installa il breakpoint all'indirizzo `rip` e restituisce `true`, altrimenti restituisce `false`; è un errore se `rip` non appartiene all'intervallo [`ini_utn_c`, `fin_utn_c`) (zona utente/condivisa).
- `natl bpwait()`: (tipo 0x5a, già realizzata): attende che un qualche processo passi dal breakpoint e ne restituisce l'identificatore; può essere invocata più volte per ottenere gli identificatori di tutti i processi intercettati; è un errore invocare questa primitiva se non ci sono breakpoint installati;
- `void bpremove()` (tipo 0x5b, da realizzare): rimuove il breakpoint e risveglia tutti i processi che erano stati intercettati; è un errore invocare questa primitiva se non ci sono breakpoint installati.

Suggerimento: Il comando `process dump` del debugger è stato modificato in modo da mostrare il disassemblato del codice intorno al valore di `rip` salvato in pila.