PCAD
Programmazione Concorrente e Algoritmi Distribuiti

Laurea Triennale Informatica, a.a. 2021/22

Lock-free Programming

- La programmazione concorrente può richiedere la conoscenza dell'architettura ad esempio per gestire race condition legate al modello della memoria (es. relaxed memory model che non garantisce sequential consistency)
- La programmazione concorrente senza lock (lock-free) una tecnica usata per ridurre l'overhead dovuto all'uso di sincronizzazione tramite lock, semafori e monitor
- Definizione di programma lock-free: programma multithreaded che usa risorse condivise e che non presenta interleaving che bloccano i thread (deadlock, livelock)

Tecniche usate nel Lock-free Programming

- Nel lock-free programmi si usano istruzioni a basso livello come le memory fence
- Es. **mfence** su x86 garantisce che ogni load/store effettuata fino alla chiamata di mfence sia visibile al sistema prima di quelle seguenti
- Le memory fence (barriere di memoria) permettono di recuperare proprietà come *write atomicity*
- Vediamo quali altri meccanismi "hardware" possono essere usati per controllare le race condition

Soluzioni Hardware alle Race Condition

- Istruzioni per disabilitare interrupt
- Istruzioni speciali per rendere atomica l'esecuzione di un test e di un assegnamento

Disabilitazione degli interrupt

- Il processo può disabilitare TUTTI gli interrupt hw all'ingresso della sezione critica, e riabilitarli all'uscita
  - Soluzione semplice; garantisce la mutua esclusione
  - ma pericolosa: il processo può non riabilitare più gli interrupt, acquisendosi la macchina
  - può allungare di molto i tempi di latenza
  - non scala a macchine multiprocessore (a meno di non bloccare tutte le altre CPU)
- Inadatto come meccanismo di mutua esclusione tra processi utente
- Adatto per brevi(ssimi) segmenti di codice affidabile (es: in kernel, quando si accede a strutture condivise)

Istruzioni speciali: Test and Set (TS)

- Le istruzioni di Test and Set: testano e modificano atomicamente il contenuto di una variabile/cella di memoria

  TS(x,oldx) := atomico(oldx := x ; x := 1)

  TS(x,oldx) ritorna in oldx il valore precedente di x ed assegna 1 ad x item Questi due passi devono essere atomici Cioe' abbiamo bisogno di una ipotetica istruzione

  `TS RX,LOCK`

  che copia il contenuto della cella `LOCK` nel registro `RX`, e poi imposta la cella `LOCK` ad un valore $\neq 0$.
  Il tutto atomicamente (viene bloccato il bus di memoria).

TS per Mutua Esclusione

```
shared var lock=0;
process P:
  bool old;
  while (true) do
    repeat
      TS(lock, old);
    until (old==0);
    critical section
    lock:=0;
    non-critical section
  endwhile
```

- Assicura mutua esclusione e progresso
- Tuttavia e' basato su spinlock e quindi genera busy wait

Istruzioni speciali: Compare and Swap (CAS)

- Le istruzioni di Compare and Swap:

  CAS(x,y) := atomico(aux:=x; x:=y; y:=aux;)

- effettua uno swap del valore di due variabili (usando una variabile locale aux) in maniera atomica

CAS per Mutua Esclusione

```
shared var lock=0;
process P:
   bool old;
   while (true) do
     repeat
       old=1;
       CAS(lock, old);
     until (old==0);
     critical section
     lock:=0;
     non-critical section
   endwhile
```

Evitare busy wait

- Le soluzioni basate su spinlock portano a
  - busy wait: alto consumo di CPU
  - inversione di priorità: un processo a bassa priorità che blocca una risorsa viene ostacolato nella sua esecuzione da un processo ad alta priorità in busy wait sulla stessa risorsa.
- Idea migliore: quando un processo deve attendere un evento, che venga posto in *wait*; quando l'evento avviene, che venga posto in *ready*
- Servono specifiche syscall o funzioni di kernel. Esempio:
  - *sleep()*: il processo si autosospende (si mette in *wait*)
  - *wakeup(pid)*: il processo *pid* viene posto in *ready*, se era in *wait*.

  Ci sono molte varianti. Molto comune: con *evento* esplicito.

Riepilogo

- Abbiamo visto alcune soluzioni software/hardware senza uso esplicito di meccanismi di sincronizzazione come i lock/semafori (lock-free)
- Nelle prossime lezioni ci sposteremo verso l'uso di meccanismi ad alto livello