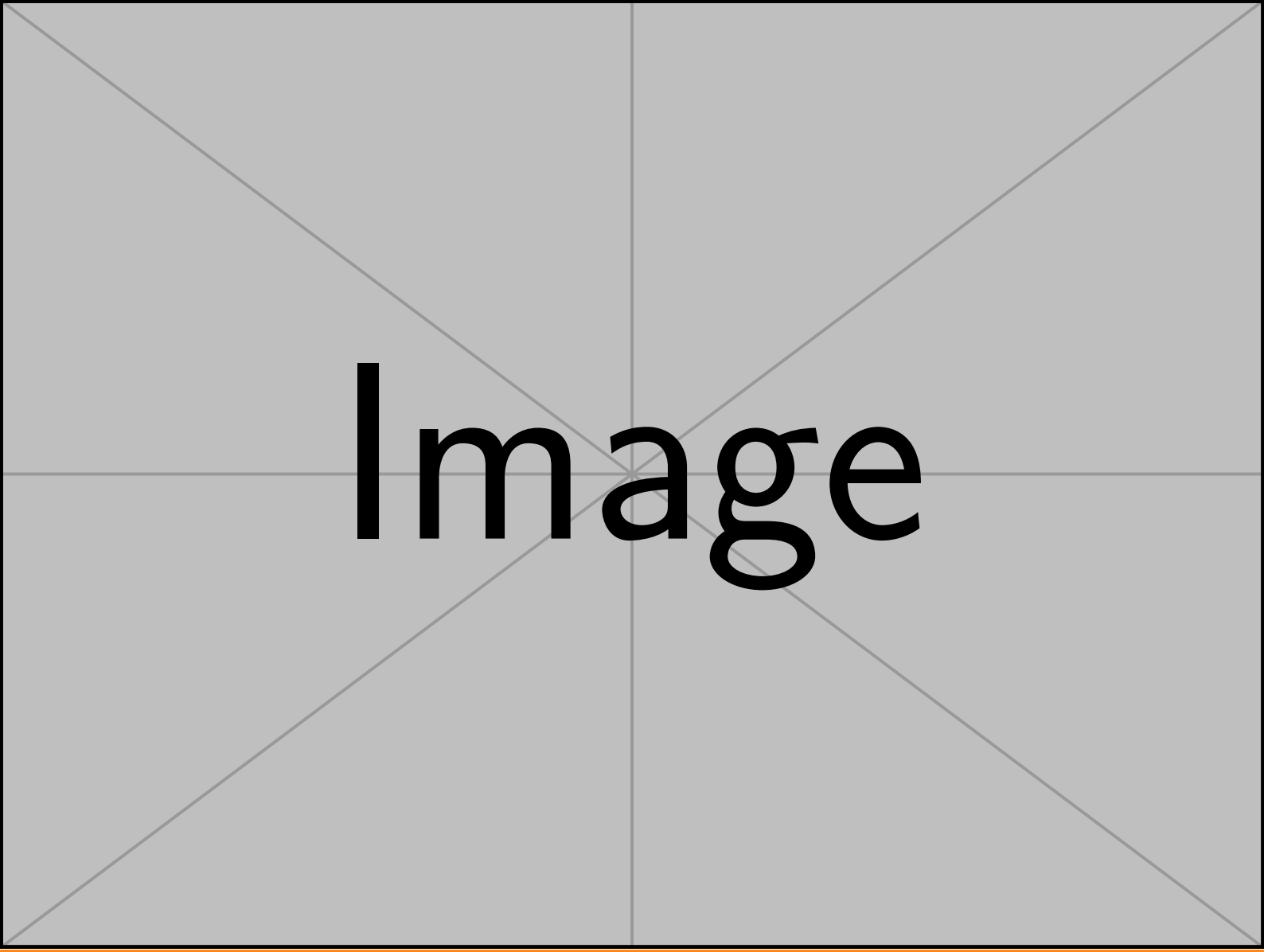

Sistemi Operativi Avanzati

E Sicurezza dei Sistemi

Autore: Andrea Efficace (https://github.com/Effibot)

Istituto: Università di Roma Tor Vergata

Data: 15 gennaio 2023

Indice

Capitolo 1 Hardware Insights

Il mondo dell'IT si è evoluto nel tempo verso linguaggi di programmazione più astratti ed espressivi, al fine di rendere più facile la vita al programmatore e di lasciare la gestione delle risorse hardware a moduli dedicati.

Di norma infatti un programmatore non si preoccupa di alcuni aspetti che sono alla base del funzionamento di un sistema come:

- Le decisioni prese dal compilatore per la gestione delle risorse.
- Le decisioni prese dal processore per un'esecuzione efficiente del **program flow**, come l'*hyper-threading* o l'esecuzione su più core.
- La disponibilità, o l'assenza, delle risorse hardware per l'esecuzione di un programma.

In questo modo si è persa tuttavia anche la capacità di configurare questo hardware in modo da ottimizzare le prestazioni del sistema, e di capire cosa succede quando si esegue un programma.

E' importante quindi per il sistemista riconsiderare la definizione di stato di un programma:

Definizione 1.1 (Program State)

Lo stato di un programma non è costituito soltanto da tutte le risorse software che lo compongono, ma anche da tutte le risorse hardware che lo eseguono accedendo alle risorse esposte dall'ISA (Instruction Set Architecture) del processore e non solo.

Un esempio importante di come l'hardware possa influenzare il comportamento di un programma rispetto al program flow espresso nel codice sorgente è dato dal **Bakery Algorithm** di Lamport, famoso per **risolvere** il problema dell'**accesso concorrente** a una risorsa condivisa:

Esempio 1.1 (Lamport's Bakery Algorithm)

```
# var: choosing = array[1,n] of boolean
# var: number = array[1,n] of integer
while true do {
    choosing[i] := true
    number[i] := 1 + max(number[1], ..., number[n])
    choosing[i] := false
    for j := 1 to n do{
        while choosing[j] do no-op
        while number[j] != 0 and (number[j], j) < (number[i], i) do no-op
    }
    # -- critical section -- #
    number[i] := 0
}
```

Osservazione 1.1

Nello schema precedente l'attesa è attiva: infatti avviene a livello utente tramite controlli su numeri e ID di processi senza mai chiamare il sistema operativo.

Infatti, quando un thread vuole entrare nella sezione critica viene alzato un **flag**, settand il valore di *choosing*[*i*] a *true*, e viene assegnato un numero di ordine aumentando di uno il valore di *number*[*i*] rispetto al valore massimo tra tutti i thread che stanno aspettando di entrare nella sezione critica. Si abbassa poi il flag e si cerca di accedere alla sezione critica tramite con due cicli consecutivi all'interno di uno di controllo:

1. Il primo ciclo controlla che nessun altro thread stia aspettando di entrare nella sezione critica, altrimenti il thread si blocca.
2. Il secondo ciclo controlla che possa effettivamente essere servito effettuando un check per controllare che tutti i thread precedenti siano stati serviti e che la coppia (*number*[*j*], *j*) sia minore di quella degli altri.

Supponendo che il programma sia eseguito su una macchina con supporto al multi-threading e che quindi il processore esegua in modo parallelo la stessa funzione espressa nell'algoritmo per più thread, è possibile che l'accesso alla sezione critica venga concesso a più thread contemporaneamente, violando il principio di esclusione mutua.

Implementando l'algoritmo, ad esempio in C, sfruttando un sistema di *logging* facendo consegnare ai thread il loro numero d'ordine al momento di essere serviti, si potrebbe vedere che ad un certo punto il numero depositato avrebbe avuto un incremento doppio rispetto al precedente o che qualche numero d'ordine salterebbe. Questo comportamento non è dovuto al codice sorgente né al processo di compilazione, ma è dovuto al fatto che **nessuna macchina** è **off-the-shelf sequentially consistent** (OSSC), ovvero che nessuna macchina esegue in modo sequenziale i thread che gli vengono consegnati ma a run-time il processore compie delle decisioni che possono influenzare il comportamento del programma e, pertanto, andrebbero considerate nel momento in cui si scrive un programma concorrente.

Bibliografia

[1] «ElegantBook». In: (2022). https://github.com/ElegantLaTeX/ElegantBook.

[2] «The Legrand Orange Book». In: (2008). http://www.latextemplates.com/template/legrand-orange-book.