



## Risorse protette



**Anno accademico 2018/19**  
**Sistemi Concorrenti e Distribuiti**

Tullio Vardanega, [tullio.vardanega@math.unipd.it](mailto:tullio.vardanega@math.unipd.it)

Laurea Magistrale in Informatica, Università di Padova
1/24



Risorse protette

## Exclusion synchronization – 1

- ❑ **Requisiti**
  1. Accesso in scrittura in mutua esclusione
  2. Accesso in sola lettura potenzialmente concorrente
  3. Distinzione tra operazioni R/O e operazioni R/W

```
protected type Shared_Integer (Initial_Value : Integer) is
function Read return Integer;
procedure Write (Value : Integer);
private
The_Integer : Integer := Initial_Value;
end Shared_Integer;
```

```
protected body Shared_Integer is
function Read return Integer is
begin
return The_Integer;
end Read;
procedure Write (Value : Integer) is
begin
The_Integer := Value;
end Write;
end Shared_Integer;
```

Lettura concorrenti

Scrittura in mutua esclusione

Laurea Magistrale in Informatica, Università di Padova
3/24



Risorse protette

## Limiti del modello base

- ❑ Per supportare protocolli di condivisione avanzati è desiderabile combinare *exclusion synchronization* e *avoidance synchronization* nella stessa astrazione
  - Senza dover impiegare *server* e comunicazione sincrona
  - Garantendo sufficiente potenza espressiva
- ❑ Modelli di riferimento sono il *monitor* di Hoare e le regioni critiche condizionali di Brinch Hansen
  - 1972, Per Brinch Hansen, "Structured Multiprogramming", CACM 15(7):574-578

Laurea Magistrale in Informatica, Università di Padova
2/24



Risorse protette

## Exclusion synchronization – 2

- ❑ **Vantaggi rispetto all'impiego di un *server***
  - **Risparmio di risorse a tempo d'esecuzione**
    - Entità passiva con un agente implicito di mutua esclusione vs entità attiva programmata come agente sincrone
  - **Minore complessità di terminazione**
    - L'entità passiva non termina, semplicemente viene rimossa non appena il suo ambiente (*scope*) cessa di esistere
    - L'onere di esecuzione è tutto e solo a carico dei processi cliente
- ❑ **Possibili controindicazioni**
  - **Insufficiente espressività del protocollo di accesso**

Laurea Magistrale in Informatica, Università di Padova
4/24



Risorse protette

## Avoidance synchronization – 1

- ❑ **Requisiti**
  - Sincronizzarsi con una "condizione" associata allo stato logico della risorsa condivisa
  - Sospendersi fino al verificarsi della condizione
- ❑ **Soluzione migliorativa: *eggshell model***
  - La RP offre un punto di accesso (*entry*) sul quale una guardia Booleana può agire come preconditione
  - L'accodamento su guardia chiusa avviene all'interno della risorsa protetta e non fuori di essa
    - L'attesa sulla preconditione non comporta rischio di *starvation*
    - *Fairness* garantita da politica FIFO sull'accodamento



Laurea Magistrale in Informatica, Università di Padova

5/24



Risorse protette

## Esempio: *bounded buffer* – 1

```

Buffer_Size : constant Positive := 5;
type Index is mod Buffer_Size; -- tipo modulare
subtype Count is Natural range 0 .. Buffer_Size;
type Buffer_T is array (Index) of Any_Type;

protected type Bounded_Buffer is
  entry Get (Item : out Any_Type);
  entry Put (Item : in Any_Type);
private
  First : Index := Index'First; -- 0
  Last : Index := Index'Last; -- 4
  In_Buffer : Count := 0;
  Buffer : Buffer_T;
end Bounded_Buffer;
    
```

Laurea Magistrale in Informatica, Università di Padova

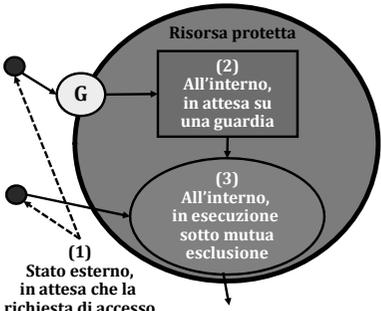
7/24



Risorse protette

## Avoidance synchronization – 2

I tre stati del modello a "guscio d'uovo"



- ❑ **Due stati "interni"**
  - Esecuzione in mutua esclusione (stato: 3)
  - Attesa su guardia Booleana di preconditione senza rischi di *starvation* (stato: 2)
- ❑ **Valutazione della guardia in mutua esclusione**
  - Nel passaggio da (stato: 1) verso (stato: 2) o (stato: 3)



Laurea Magistrale in Informatica, Università di Padova

6/24



Risorse protette

## Esempio: *bounded buffer* – 2

```

protected body Bounded_Buffer is
  entry Get (Item : out Any_Type)
    when In_Buffer > 0 is
  begin -- first read then move pointer
    Item := Buffer(First);
    First := First + 1; -- free from overflow
    In_Buffer := In_Buffer - 1;
  end Get;
  entry Put (Item : in Any_Type)
    when In_Buffer < Buffer_Size is
  begin -- first move pointer then write
    Last := Last + 1; -- free from overflow
    Buffer(Last) := Item;
    In_Buffer := In_Buffer + 1;
  end Put;
end Bounded_Buffer;
    
```



Laurea Magistrale in Informatica, Università di Padova

8/24



Risorse protette

## Avoidance synchronization – 3

- ❑ Un chiamante può limitare il tempo d'attesa su un punto d'accesso di RP usando il costrutto `select`
- ❑ Un punto d'accesso di RP può essere
  - Aperto, se la sua guardia, quando valutata, vale vero
  - Chiuso, se la sua guardia, quando valutata, vale falso
- ❑ Una guardia di RP va rivalutata
  - A ogni richiesta d'accesso la cui guardia abbia una componente che possa essere cambiata dall'ultima valutazione
  - A ogni completamento d'esecuzione R/W entro la RP, quando vi siano processi accodati su una guardia le cui componenti potrebbero essere cambiate dall'ultima valutazione



Laurea Magistrale in Informatica, Università di Padova

9/24



Risorse protette

## Avoidance synchronization – 5

- ❑ Un processo in possesso RW di una RP può invocare altri servizi della stessa RP senza dover "uscire dal guscio"
  - Essi verranno eseguiti immediatamente senza dover contendere per l'accesso esclusivo
- ❑ Ciò non accade per chiamate indirette effettuate all'esterno della RP in procedure invocate dall'interno di essa
  - "Uscire e rientrare": viola l'*encapsulation* della RP e comporta rischio di stallo

Laurea Magistrale in Informatica, Università di Padova

11/24



Risorse protette

## Avoidance synchronization – 4

- ❑ Una RP è "in uso R" (*read lock*) quando più processi eseguono una sua funzione
- ❑ La RP è "in uso RW" (*write lock*) quando un processo sta eseguendo una procedura o entry su di essa
- ❑ In entrambi i casi ogni altra chiamata viene trattenuta all'esterno della RP

Laurea Magistrale in Informatica, Università di Padova

10/24



Risorse protette

## Protocollo d'accesso – 1

1. Se la risorsa protetta (RP) è sotto "*read lock*" e la chiamata è a funzione, questa viene eseguita e si passa al punto 14
2. Se RP è sotto "*read lock*" e la chiamata è a procedura o *entry*, la chiamata viene differita fin quando vi siano chiamate attive all'interno di RP
3. Se RP è sotto "*write lock*", qualunque chiamata viene differita fin che vi siano chiamate attive all'interno di RP con requisiti di accesso potenzialmente in conflitto con essa
4. Se RP non è in uso e la chiamata è a funzione, RP assume un "*read lock*" e si passa al punto 5
5. La funzione di RP viene eseguita e si passa al punto 14
6. Se RP non è in uso e la chiamata è a procedura o *entry*, RP assume un "*write lock*" e si passa al punto 7
7. Se RP non è in uso e la chiamata è a procedura, questa viene eseguita e si passa al punto 10, altrimenti si passa al punto 8
8. Se la chiamata è a *entry*, la sua guardia è valutata e, se aperta, si esegue il corpo dell'*entry* e poi si passa al punto 10, altrimenti si passa al punto 9

Laurea Magistrale in Informatica, Università di Padova

12/24



Risorse protette

## Protocollo d'accesso – 2

9. Poiché la guardia è chiusa, la chiamata è posta nella coda associata alla guardia e da questo momento inizia la valutazione delle clausole di selezione del chiamante (*time out*); poi si passa al punto 10

10. Viene rivalutata ciascuna guardia che abbia chiamate in attesa, la cui espressione contenga variabili che possano essere cambiate dall'ultima valutazione; poi si passa al punto 11

11. Tra le guardie che risultassero aperte se ne seleziona una eseguendo il corpo della corrispondente *entry* e poi si torna al punto 10, altrimenti si passa al punto 12

12. Se nessuna guardia con chiamate in attesa è aperta si passa al punto 13

13. Tra le chiamate differite all'esterno di RP si selezionano o tutte quelle a funzione, che richiedono "*read lock*", oppure una tra quelle che richiedono "*write lock*" e si eseguono i passi 5 o 7 o 8; se non vi fossero chiamate il protocollo d'accesso completa

14. Quando non vi fossero più chiamate attive all'interno di RP si passa al punto 13

Laurea Magistrale in Informatica, Università di Padova

13/24



Risorse protette

## Controllo d'accodamento – 1

- ❑ **Ogni punto d'accesso a canale tipato (di *server*, di RP) possiede un attributo predefinito `\Count`**
  - Funzione che ritorna il numero di chiamate attualmente accodate sull'*entry*
- ❑ **Per questo motivo anche l'accodamento di chiamata richiede "*write lock*" sulla risorsa!**
- ❑ **L'uso dell'attributo `\Count` nell'espressione di una guardia ne causa la rivalutazione a ogni esito di valutazione di chiamata**

Laurea Magistrale in Informatica, Università di Padova

15/24



Risorse protette

## Protocollo d'accesso – 3

- ❑ **L'aspetto più delicato riguarda la valutazione delle guardie (punto 10.)**
  - Dopo l'esecuzione di procedure/ *entry* e al punto di chiamata (8.)
- ❑ **Per ogni guardia aperta si esegue il corpo dell'*entry* corrispondente**
  - Il protocollo non prescrive quale processo debba farsene carico
  - Il processo chiamante oppure quello che ha aperto la guardia (l'incapsulazione dello stato di RP consente di scegliere)
- ❑ **Le clausole temporali di selezione poste dal chiamante sono valutate solo a partire da quando la chiamata viene accodata**

Laurea Magistrale in Informatica, Università di Padova

14/24



Risorse protette

## Controllo d'accodamento – 2

Esempio: barriera con uscita a gruppi di N

```
protected Guardian is
entry Let_In;
private
Open : Boolean := False;
end Guardian;
protected body Guardian is
entry Let_In
when [Let_In'Count = N or Open] is
begin
if Let_In'Count = 0 then
Open := False;
else
Open := True;
end if;
end Let_In;
end Guardian;
```

- L'N-esimo chiamante troverà la guardia chiusa, ma il suo accodamento causerà la modifica di `'Count`
- Ciò comporta una nuova valutazione della guardia, ora divenuta aperta
- La I chiamata accodata modificherà `Open` così che la guardia resti aperta fino a che sia stata rilasciata la N-esima chiamata accodata
- Essa chiuderà di nuovo la guardia

Laurea Magistrale in Informatica, Università di Padova

16/24



Risorse protette

## Osservazione ed esercizio

- ❑ La RP dell'esempio precedente realizza un protocollo di accesso a uno stato logico *S*, di sistema, realizzato al di fuori di RP
- ❑ Estenderne la logica per garantire che non più di *N* chiamanti siano simultaneamente nello stato *S*

Laurea Magistrale in Informatica, Università di Padova17/24



Risorse protette

## Restrizioni d'uso – 2

- ❑ La rilevazione di situazioni erronee comporta il completamento forzato del programma con eccezione `Program_Error`
- ❑ In caso di mancata rilevazione il programma può entrare in stato di stallo
- ❑ Costrutti potenzialmente bloccanti
  - `select`, `accept`, `entry call`, `delay [until]`, `new`
  - Transitivamente, qualunque chiamata a sottoprogramma che sia al suo interno potenzialmente bloccante

Laurea Magistrale in Informatica, Università di Padova19/24



Risorse protette

## Restrizioni d'uso – 1

- ❑ L'esecuzione entro RP dovrebbe essere il più breve possibile
  - Così come la sincronizzazione tra *client* e *server*
- ❑ Per questo consideriamo erronee le azioni potenzialmente bloccanti effettuate entro azioni protette
  - Vogliamo proteggere la logica implicita del modello
  - Nel caso del *server* invece la prevenzione è in capo al programmatore (!)

Laurea Magistrale in Informatica, Università di Padova18/24



Risorse protette

## Elaborazione e finalizzazione

- ❑ L'elaborazione di una RP avviene quando si incontra la sua dichiarazione in una regione dichiarativa
- ❑ La sua finalizzazione però non può avvenire fin quando vi siano chiamate accodate sui suoi punti d'accesso
  - Ciò richiederebbe che i corrispondenti processi venissero terminati con eccezione `Program_Error`

Laurea Magistrale in Informatica, Università di Padova20/24



## Risorse protette

### Ordinamento preferenziale – 1

- ❑ Per erogare ordinamento preferenziale (OP) serve più della sola mutua esclusione
  - P.es.: per smaltire richieste di scrittura in preferenza a quelle di lettura
- ❑ La RP non può erogare servizi che comportino la sospensione del chiamante
  - Come è tipico di alcune risorse fisiche remote (disco)
- ❑ Ma la RP aiuta a realizzare efficaci protocolli d'accesso (anche con OP)

Laurea Magistrale in Informatica, Università di Padova

21/24



## Risorse protette

### Ordinamento preferenziale – 3

- ❑ Il protocollo d'accesso dell'esempio consente
  - Scritture in mutua esclusione (guardia G2)
    - Come se fossero incapsulate all'interno di una RP
  - Preferenza a scritture su letture (guardia G1)
- ❑ Quando la risorsa fisica utilizzata è esterna alla RP si corre rischio di stallo
  - La terminazione erronea di operazioni sulla risorsa fisica può impedire il rilascio della RP e bloccarne l'accesso



Laurea Magistrale in Informatica, Università di Padova

23/24



## Risorse protette

### Ordinamento preferenziale – 2

```
protected Access_Control is
entry Start_Read;
procedure Stop_Read;
entry Start_Write;
procedure Stop_Write;
private
Readers : Natural := 0;
Writers : Boolean := False;
end Access_Control;

procedure Read (I : out Item) is
begin
Access_Control.Start_Read;
... -- actual read (suspensive)
Access_Control.Stop_Read;
end Read;

procedure Write(I : in Item) is
begin
Access_Control.Start_Write;
... -- actual write (suspensive)
Access_Control.Stop_Write;
end Write;
```

```
protected body Access_Control is
entry Start_Read when not Writers and
[Start_WriteCount = 0] is
begin
Readers := Readers + 1;
end Start_Read;

procedure Stop_Read is
begin
Readers := Readers - 1;
end Stop_Read;

entry Start_Write when not Writers and
[Readers = 0] is
begin
Writers := True;
end Start_Write;

procedure Stop_Write is
begin
Writers := False;
end Stop_Write;
end Access_Control;
```

Laurea Magistrale in Informatica, Università di Padova

22/24



## Risorse protette

### Stati d'esecuzione di processo

Laurea Magistrale in Informatica, Università di Padova

24/24