

## MONITOR > INTRO

È un ADT creato per estrarre la gestione del semaforo.

Al suo interno contiene:

- dati: I dati comuni
- funzioni: per operare sui dati (indipendenti dal tipo di dato) sono in mutua esclusione.
- variabili e funzioni di gestione del monitor.

I dati e le funzioni del monitor possono operare solo con gli elementi interni al monitor.

### FUNZIONAMENTO

Il monitor consente ad 1 proc. alla volta di entrare ed eseguire le funzioni.

Se ci sono più processi, si usa una coda d'accesso.

### CONDITION VARIABLES

Il monitor base riesce a gestire l'accesso concorrente a risorse esistenti condivise ma fatica in altri casi (es: sync v<sup>ord.</sup>)

### SOLUZIONE BRUTALE

1 prima sol. sarebbe che P2 faccia dei controlli contenuti ma sarebbe terribile per le prestazioni.

Per gestire i casi si usano cond. variables. Ognuna mette a disposizione (per condition x):

- x.wait();

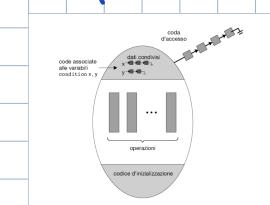
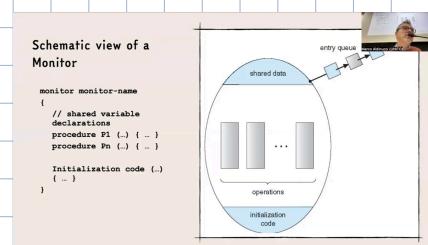
Aspetta fino a quando un altro processo invochi

- x.signal();

Sulla stessa variabile x.

Se non ci sono processi in attesa, x.signal non farà niente (segnale viene perso). Ovvero x resta invocato, cosa che è totalmente diversa dai semafori tradizionali.

Quindi, ogni c. variable è rappresentabile con la propria coda. (2 code tot: 1 fuori e 1 dentro)



## MONITOR > CONDITION VARIABLES CHOICIE

Siccome i processi in `x.wait()`; sono in attesa dentro il semaforo, saranno svegliati da un altro processo sempre dentro al semaforo. Questo viola il concetto di mutua esclusione del semaforo, quindi l'implementazione dovrà decidere tra:

- signal and wait

Il chiamante si mette in attesa appena sveglia il chiamato.

- signal and continue

Il chiamante prosegue e fa attendere il chiamato.

Il ling. Pascal prevede che il chiamante lasci il monitor.

## MONITOR > IMPLEMENTAZIONE

### CON SEMAFORI PER ACCESSO

Si usano 2 semafori:

- mutex: per garantire mutua esclusione (init 1)
- next : per definire la coda esterna di accesso (init 0)

Ogni funzione che deve essere eseguita dovrà essere "decorata" con

- wait(mutex) : Prima della funzione  $\rightarrow$  mutua excl.
- signal(next) / signal(next) : Dopo la funzione

Si usa un contatore degli elem. in coda

- coda > 0 : signal(next)  
inizia il prossimo
- coda = 0 : signal(mutex)

Chiude la mutua esclusione (necessario per funz.)

### AGGIUNTA COND. VARIABLE

Per ogni cond. var. viene aggiunto un semaforo e una variabile contatore per regolare la coda e il # di elem. rimanenti nella cond. variable. (entrambe init a 0).

Monitor Implementation Using Semaphores

```
< body of F >
Variables
    semaphore mutex; // init = 1
    semaphore next; // init = 0
    int next_count = 0;
...
if (next_count > 0)
    signal(next)
else
    signal(mutex);
...
Each procedure F will be replaced
by
wait(mutex);
...
Mutual exclusion within a monitor
is ensured
```

## x.Wait()

1.  $x\_count++$ : aumenta il contatore

2. if (next...) :

Se ci sono elementi in attesa nella coda  
esterna, "falli andare"

3. else {} : Altrimenti esci dal monitor

4. wait(x-sem) : Infine si sospende sul semaforo e  
 $x\_count--$   
se lo acquisisce decremente i posti.

```

x_count++;
if (next_count > 0)
    signal(next);
else
    signal(mutex);
wait(x_sem);
x_count--;

```

## x.signal()

1. if (...) : se ci sono elementi in coda

2. next\_count++ : Aumenta la coda esterna

3. signal() : libero i elem della coda.

4. wait(); : mi metto in attesa (signal and wait)  
next count : e appena finito decremento

```

if (x_count > 0) {
    next_count++;
    signal(x_sem);
    wait(next);
    next_count--;
}

```

## RIPRESA DEL PROCESSI ➤

Se ci sono N processi in attesa sulla cond. variable,  
di solito si usa un'attesa condizionale su priorità (x.wait(c))

## SINGLE RESOURCE ALLOCATION ➤

Modello di allocazione delle risorse in cui  
processi competitivi ne richiedono l'accesso.

Ogni processo deve specificare quanto tempo vuole tenere  
la risorsa. Il resource allocator darà priorità al tempo min.

Il monitor / semafori non riescono a garantire l'esecuzione  
con possibile soluzione che si includa l'accesso alla  
risorsa nel monitor.

### Single Resource allocation

```

• Allocate a single resource among competing processes using priority
numbers that specify the maximum time a process plans to use the resource

R.acquire(t);
...
access the resource;
...

R.release();
• Where R is an instance of type ResourceAllocator

```

## SYNC PROBLEMS ➤ [CAP 7] 7.1

### BOUNDED BUFFERS

Definisce uno schema generale del modello producer/cons.  
con bounded memory.

### STRUTTURE DATI

- n: La dimensione del buffer di comunicazione
- mutex: Garantisce la mutua esclusione (inizializzato a 1)
- empty: Conta il num. di pos. vuote (=n)
- full : // // // // // piene (=0)

```
int n;
semaphore mutex = 1;
semaphore empty = n;
semaphore full = 0;
```

## PRODUCER

0. Crea l'elemento da "mandare" al cons.
1. Aspetta che ci sia 1 posizione libera
2. Apri la mutua esclusione del buffer
3. Chiude la mutua esclusione
4. Segnala 1 posto occupato

```
do {
    ...
    /* produce an item in next_produced */
    ...
    wait(empty);
    wait(mutex);
    ...
    /* add next produced to the buffer */
    ...
    signal(mutex);
    signal(full);
} while (true);
```

## CONSUMER

1. Aspetta 1 elem. pieno nel buffer
2. Apri la mutua esclusione
3. Chiude //
4. Segnala 1 posto libero

```
Do {
    wait(full);
    wait(mutex);
    ...
    /* remove an item from buffer to next_consumed */
    ...
    signal(mutex);
    signal(empty);
    ...
    /* consume the item in next consumed */
    ...
} while (true);
```