

# Il problema dei lettori/scrittori

Christian Daraio

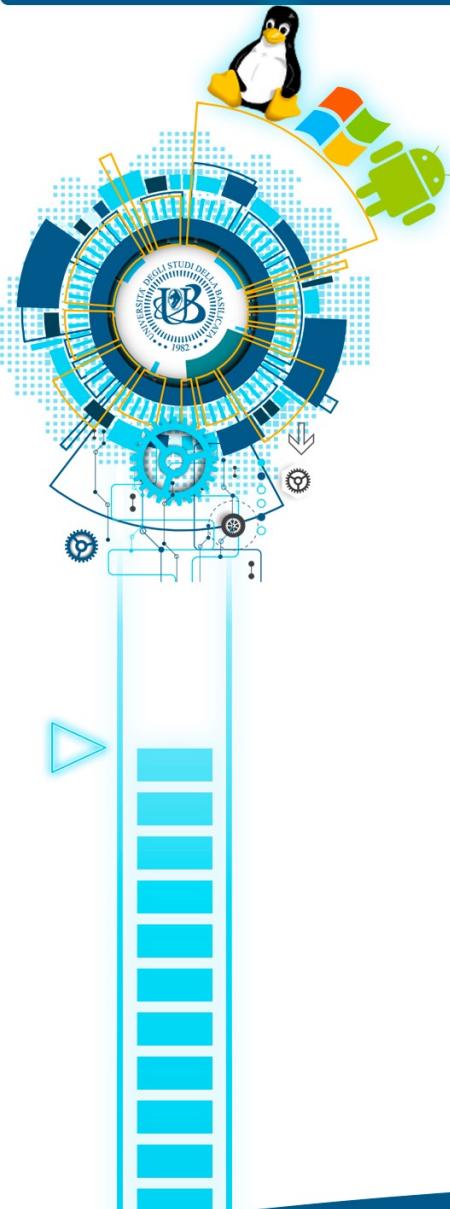
Università degli studi della Basilicata

A.A. 2021/2022

Corso: Sistemi Operativi

Prof. Domenico Bloisi

# Illustrazione del problema



Il problema dei lettori-scrittori è uno tra gli esempi di problemi connessi al **controllo della concorrenza**.

Esso rispecchia uno scenario che si può verificare in molti programmi applicativi.

Si supponga che una base di dati(es. file , database) sia da condividere tra numerosi processi concorrenti che vogliono accedervi in modalità di lettura o scrittura.



# Illustrazione del problema

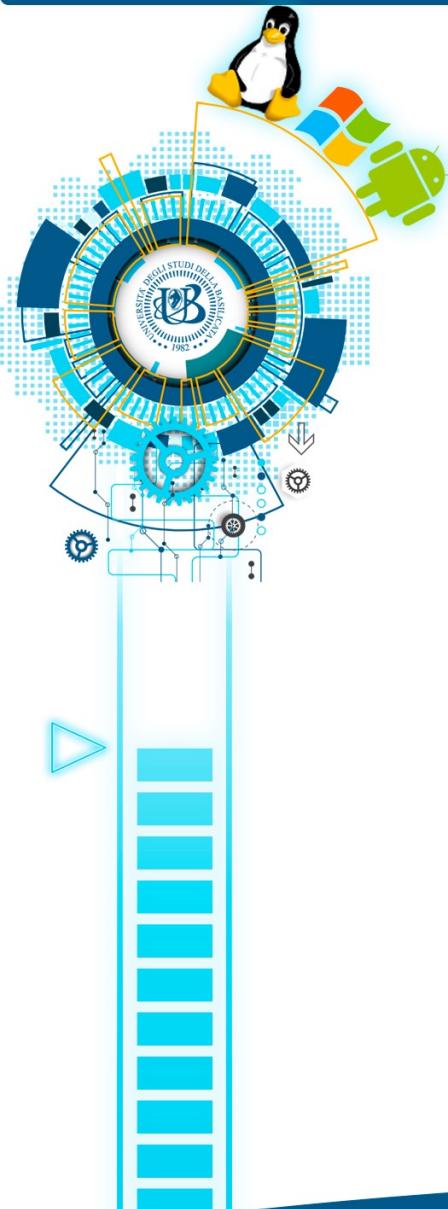


È possibile distinguere i processi in due tipi:

- Processi **lettori**, che possono richiedere solo la lettura del contenuto della base di dati.
- Processi **scrittori**, che ne possono effettuare un aggiornamento e quindi una scrittura.

Le strategie da adottare per far fronte al problema di sincronizzazione possono essere differenziate in base al tipo di processo che sta accedendo alla risorsa. In questo modo è possibile ottimizzare la soluzione.

# Il problema della sincronizzazione



Se due lettori accedono nello stesso momento all'insieme di dati condiviso, non si ha alcun effetto negativo;

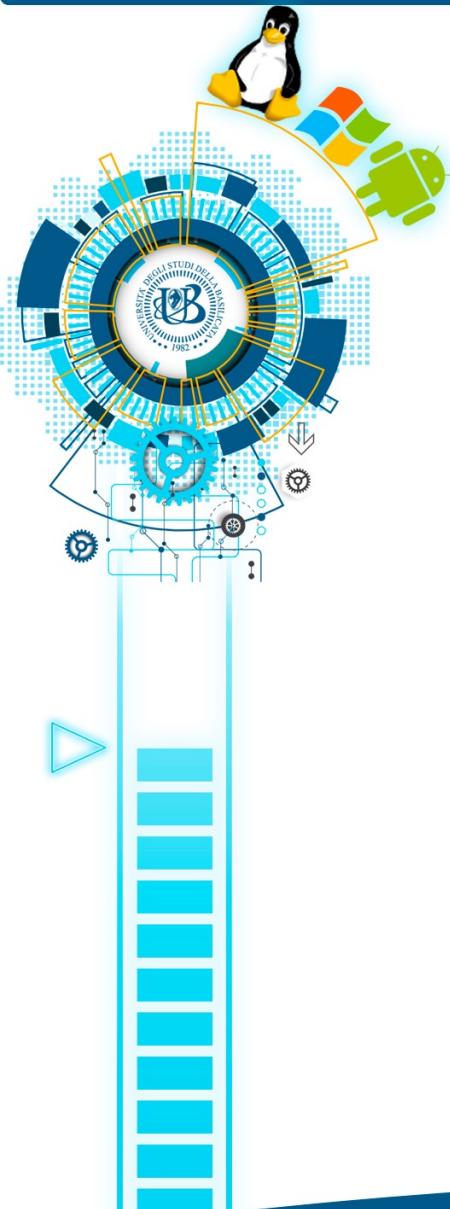
Il problema sorge nel momento in cui uno scrittore e un altro processo(lettore o scrittore) accedono contemporaneamente alla stessa base di dati.

Infatti se i processi lettori effettuassero una lettura contemporaneamente ad un processo scrittore, potrebbero leggere solo una parte del contenuto o addirittura non riuscire ad accedere alla risorsa.

Se invece due scrittori scrivessero in contemporanea, si potrebbero generare dei dati corrotti.

Per impedire l'insorgere di difficoltà di questo tipo è necessario che gli scrittori abbiano un accesso esclusivo in fase di scrittura alla base di dati condivisa.

# Riferimento ad esempio pratico



Nell'implementare la soluzione al problema, si fa riferimento ad un sistema di prenotazione di biglietti aerei.

La risorsa che viene condivisa è un volo aereo avente come destinazione una città qualsiasi(nell'esempio Londra).

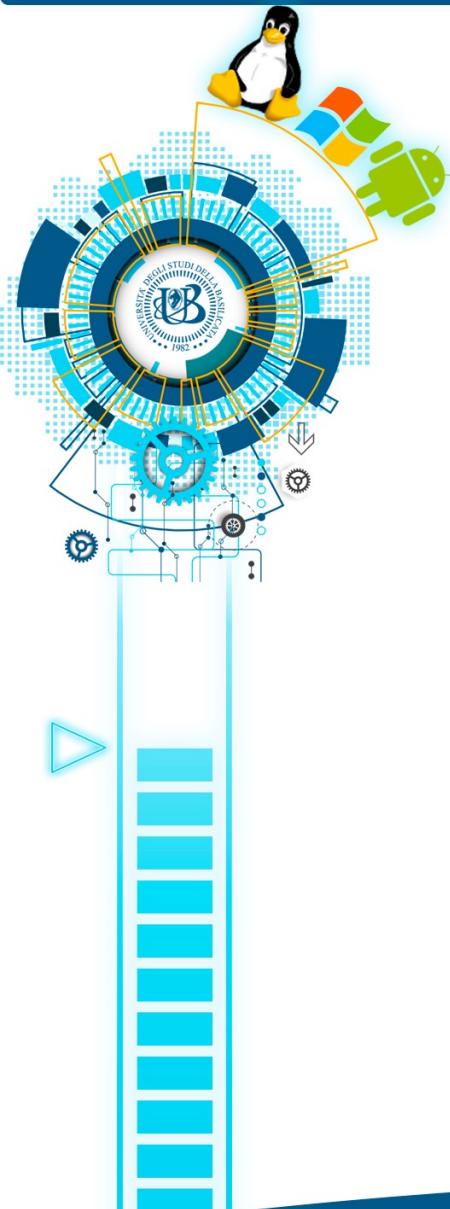
Gli addetti al settore marketing della compagnia aerea, aggiornano continuamente il prezzo del volo.

In particolare, sono stati creati:

- 10 threads lettori, che corrispondono a 10 utenti che vogliono leggere l'ultima tariffa aggiornata
- 5 threads scrittori, che corrispondono ai 5 dipendenti incaricati dell'aggiornamento tariffe del biglietto



# Il codice



Per la scrittura del codice viene utilizzata la libreria **pthread**.  
Viene di seguito descritta la sintassi e la semantica della libreria utilizzata.

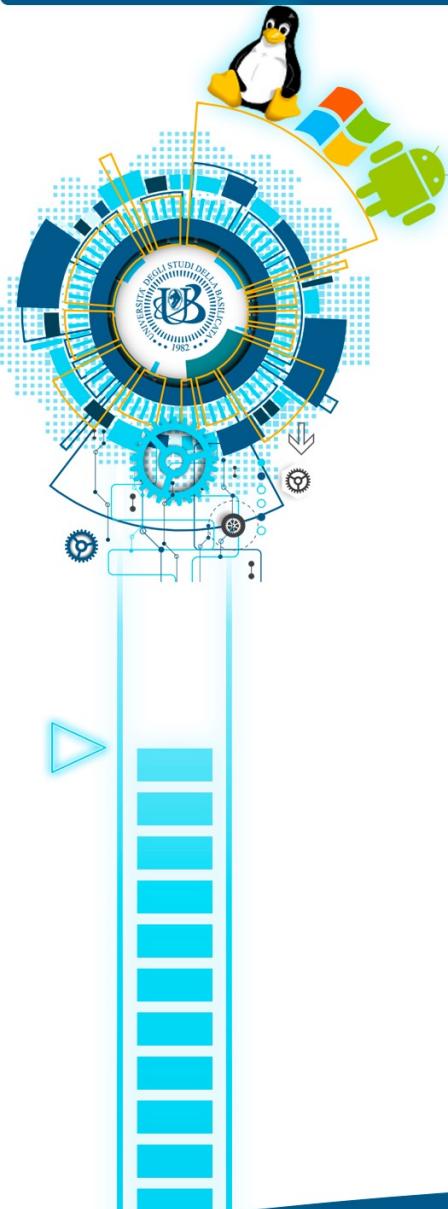
## ■ CREAZIONE DEL THREAD:

```
int pthread_create(pthread_t * thread, const pthread_attr_t * attr, void * (*start_routine)(void *), void *arg);
```

Questa funzione crea il thread accetta in input i seguenti parametri:

1. Il puntatore alla variabile che contiene il *thread\_id*
2. Un puntatore alla struttura con le caratteristiche del thread (*tipicamente NULL*)
3. La procedura che il thread deve eseguire
4. Gli eventuali parametri da passare alla funzione

# Il codice



Dopo aver creato i due array rispettivamente per gli scrittori e per i lettori con l'istruzione

```
pthread_t lettori[MAX_LETTORI], scrittori[MAX_SCRITTORI];
```

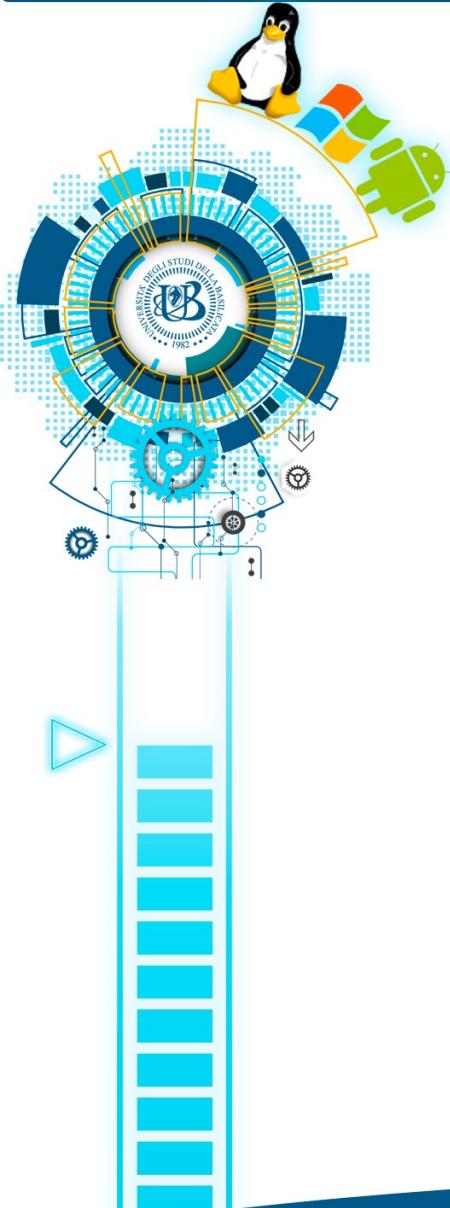
vengono creati i thread lettori e i thread scrittori attraverso le seguenti istruzioni immesse all'interno di due cicli for distinti:

```
// Creazione Threads (dipendenti)
for (unsigned long i = 0; i < MAX_SCRITTORI; i++) {
    pthread_create(&scrittori[i], NULL, scrittore, (void *)i);
}

// Creazione Threads (utenti)

for (unsigned long i = 0; i < MAX_LETTORI; i++) {
    pthread_create(&lettori[i], NULL, lettore, (void *)i);
}
```

# Soluzione proposta



Nella soluzione proposta, facendo riferimento all'esempio applicativo della gestione del costo del biglietto aereo, la base di dati condivisa viene rappresentata da una variabile avente visibilità globale, inizializzata con valore fissato 15.

Essa rappresenta il costo del biglietto e per semplicità viene utilizzato un valore di tipo intero.

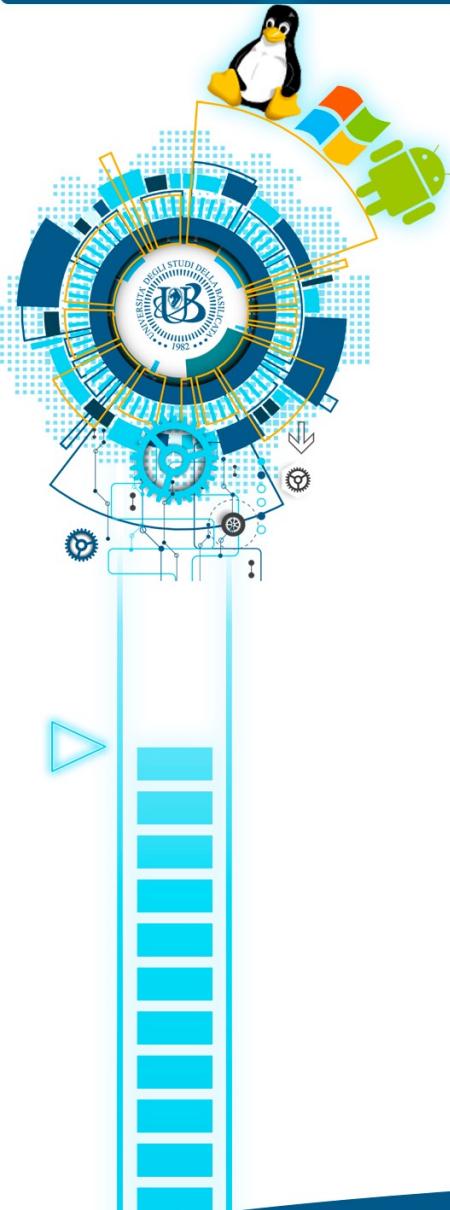
```
int costo_biglietto = 15; // risorsa condivisa ←  
int numero_lettori = 0; // numero lettori attivi  
pthread_mutex_t mutex; // protegge l'accesso alla variabile numero_lettori  
sem_t semaforo_scrittura;
```

I lettori (ovvero gli utenti che visionano la tariffa) non fanno altro che stampare il contenuto della variabile sullo standard output;

Gli scrittori (ovvero coloro che sono incaricati di aggiornare la tariffa) non fanno altro che sovrascrivere il valore (compreso tra 15 e 115)

con l'istruzione: `costo_biglietto = rand() %100 + 15;`

# Soluzione proposta



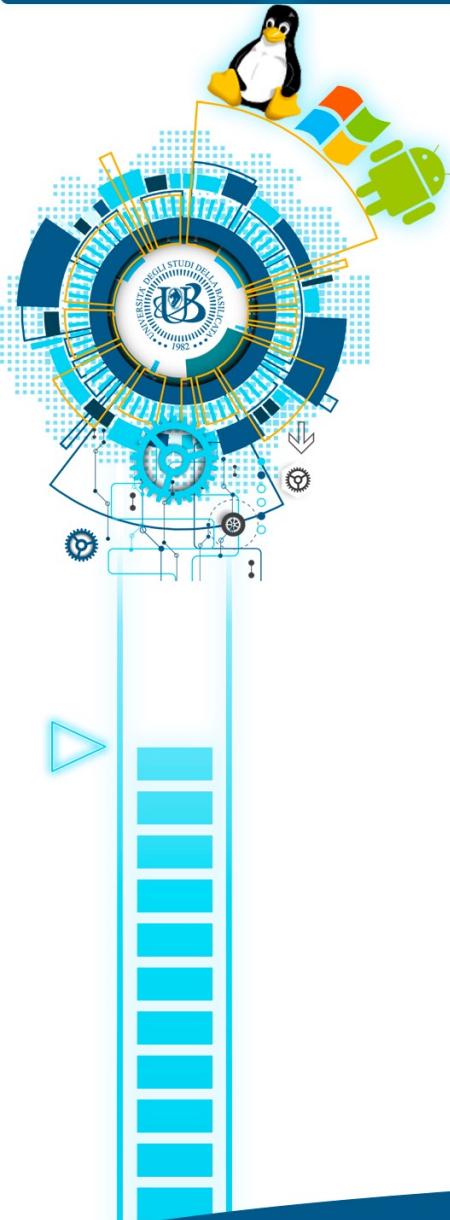
Dal momento che le operazioni di lettura-scrittura sono molto semplici e vengono effettuate su una sola variabile, le operazioni vengono effettuate istantaneamente. Per rallentare l'esecuzione e per simulare una casualità d'ordine di esecuzione è stata inserita una istruzione di *sleep* alla quale viene passato un valore compreso tra due valori limiti, generato da una funzione

```
float generaRandom(float min, float max);
```

Per risolvere il problema di sincronizzazione nella soluzione vengono privilegiati i lettori. Si richiede dunque che nessun lettore attenda, almeno che uno scrittore abbia già ottenuto l'accesso alla risorsa condivisa.

Viene utilizzata inoltre una variabile numero lettore che conta il numero di processi in lettura attualmente attivi.

# Semafori e lock mutex



La sincronizzazione viene gestita attraverso l'utilizzo di un lock (mutex) per la sezione critica relativa all'aggiornamento della variabile contatore dei processi lettori e per mezzo di un semaforo *semaforo\_scrittura* necessario per regolare l'accesso in scrittura ai processi scrittori.

Il lock mutex utilizza il tipo `pthread_mutex` di pthread.

Un thread è in grado di acquisire il lock su un lock mutex prima dell'ingresso in una zona critica e di rilasciarlo in seguito con le seguenti procedure:

- `pthread_mutex_lock(&mutex)`
- `phread_mutex_unlock(&mutex)`.

I semafori sono invece implementati nella libreria *semaphore.h*.

Vengono inizializzati con la procedura `sem_init(&semaforo, 0, 1)`, dove:

- il valore 0 indica che il semaforo viene condiviso soltanto nei threads del processo padre
- il parametro con valore 1 è il valore iniziale del semaforo.

Le istruzioni per acquisire e rilasciare il semaforo sono rispettivamente:

- `sem_wait(&semaforo)`
- `sem_post(&semaforo)`

# Il codice



```
void *lettore(void *id_thread) {
    sleep(generaRandom(1, 3)); // sleep per simulare un ritardo di lettura
    unsigned long tid = (unsigned long)id_thread;
    pthread_mutex_lock(&mutex); // inizio sezione critica
    numero_lettori++;
    if (numero_lettori == 1) {
        // verifica se si tratta del primo lettore. Esso attende che termini la fase di scrittura
        sem_wait(&semaforo_scrittura);
    }
    pthread_mutex_unlock(&mutex); // fine sezione critica
    printf("L'utente n.%lu sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto -> %d euro\n", tid, costo_biglietto);
    pthread_mutex_lock(&mutex);
    numero_lettori--;
    if (numero_lettori == 0) {
        // verifica se si tratta dell'ultimo lettore. In caso affermativo rilascia il semaforo per l'operazione di scrittura.
        sem_post(&semaforo_scrittura);
    }
    pthread_mutex_unlock(&mutex);
}
```

Il processo lettore, ottenuto il lock sul mutex, incrementa il valore della variabile che conta il numero di lettori attualmente attivi. Nel caso in cui si tratta del primo lettore, acquisisce il semaforo di scrittura assicurandosi che non ci siano processi in scrittura.

Successivamente rilascia il lock sul mutex per consentire ad altri lettori di poter leggere contemporaneamente il valore. Stampa poi sullo standard output il valore.

Dopo la lettura riacquisisce il lock sul mutex per decrementare la variabile *numero\_lettori*; nel caso in cui tutti i processi in lettura abbiano terminato rilascia il *semaforo\_scrittura* per consentire ad un processo scrittore di operare ed effettua il rilascio del lock sul mutex.

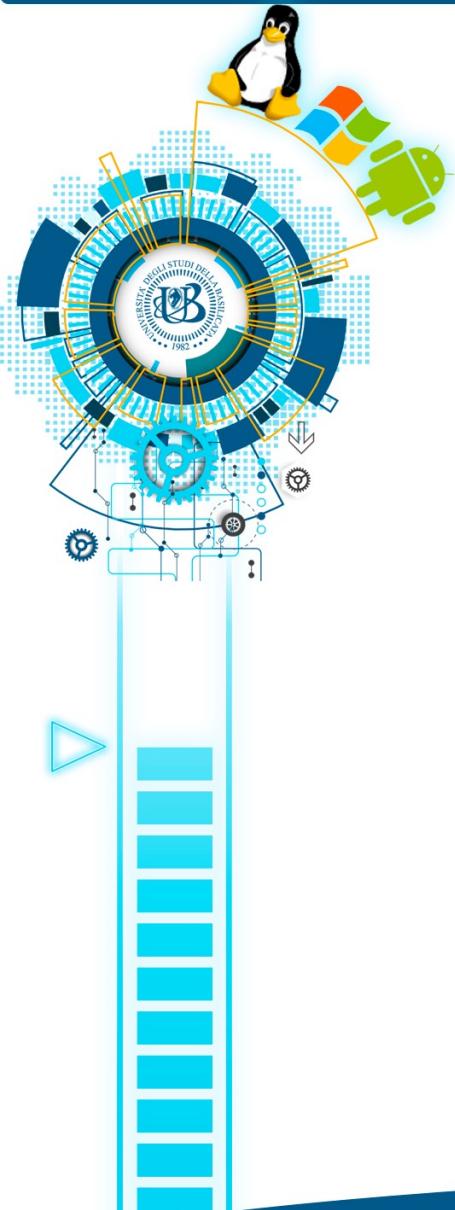
# Il codice



```
void *scrittore(void *id_thread) {
    sleep(generaRandom(0, 3)); // sleep per simulare un ritardo di scrittura
    unsigned long tid = (unsigned long)id_thread;
    sem_wait(&semaforo_scrittura); // semaforo di attesa per la fase di scrittura
    costo_biglietto = rand() %100 + 15; // generazione casuale di un nuovo importo tra 15 e 115
    printf("L'operatore n.%lu della compagnia aerea ha aggiornato il costo del biglietto per Londra\n", tid);
    sem_post(&semaforo_scrittura); // rilascio del semaforo
}
```

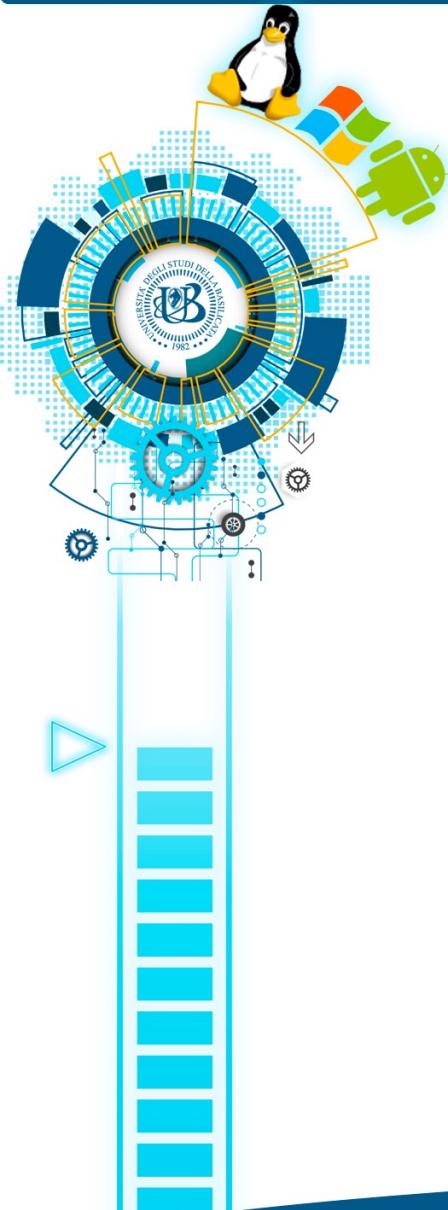
Il processo scrittore, effettua una chiamata *sem\_wait* sul semaforo di scrittura. Una volta acquisito aggiorna il valore della variabile *costo\_biglietto*, sovrascrivendola con un valore random intero compreso tra 15 e 115, ed effettua poi una stampa sullo schermo. Viene successivamente effettuata una chiamata *sem\_post* sul semaforo di scrittura per consentire ad altri processi lettore/scrittori di effettuare altre operazioni.

# Esecuzione



```
Last login: Mon Jul 18 14:55:28 on ttys000
[christiandaraio@MBPdichristian ~ % cd Desktop/progetto_SO
[christiandaraio@MBPdichristian progetto_SO % ./progetto_priorita
+-----+-----+-----+
| Questa applicazione simula un sistema di controllo delle tariffe di un biglietto aereo in continuo aggiornamento |
+-----+-----+
L'operatore n.1 della compagnia aerea ha aggiornato il costo del biglietto per Londra
L'utente n.6 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto -> 55 euro
L'utente n.3 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto -> 55 euro
L'operatore n.0 della compagnia aerea ha aggiornato il costo del biglietto per Londra
L'utente n.2 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto -> 91 euro
L'utente n.4 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto -> 91 euro
L'utente n.9 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto -> 91 euro
L'utente n.0 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto -> 91 euro
L'utente n.1 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto -> 91 euro
L'utente n.7 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto -> 91 euro
L'utente n.8 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto -> 91 euro
L'operatore n.3 della compagnia aerea ha aggiornato il costo del biglietto per Londra
L'operatore n.2 della compagnia aerea ha aggiornato il costo del biglietto per Londra
L'utente n.5 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto -> 104 euro
L'operatore n.4 della compagnia aerea ha aggiornato il costo del biglietto per Londra
christiandaraio@MBPdichristian progetto_SO % |
```

# Il problema

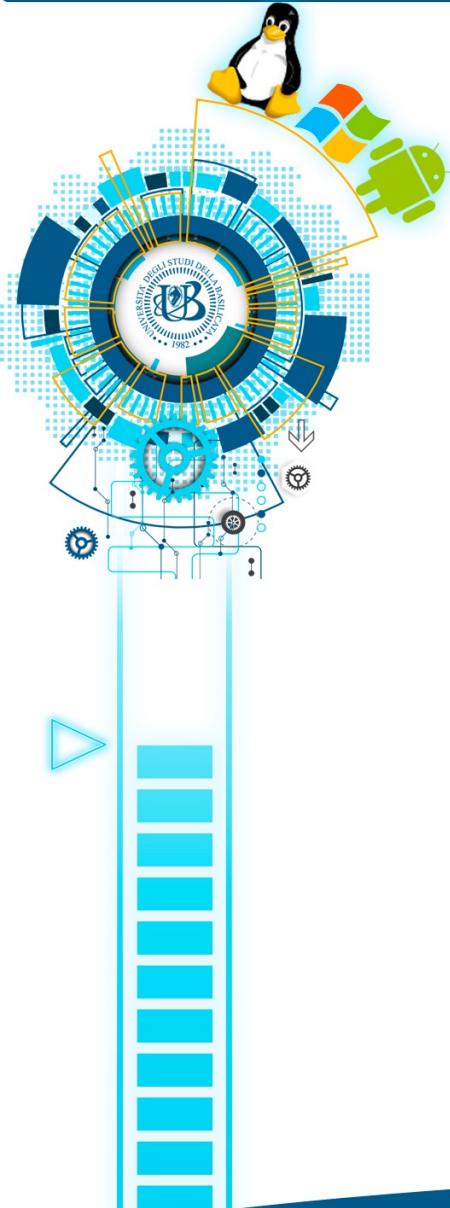


Supponendo di avere uno più processi lettori attivi e altri processi lettori in arrivo, il programma non manderà più in esecuzione i processi scrittori fino a quando tutti i processi lettori non avranno terminato.

Ciò potrebbe condurre a uno stato di attesa indefinita (**starvation**) degli scrittori.

Per porre rimedio a questo problema è stata implementata una soluzione in linguaggio C++ basata sullo stesso problema(e riferimento all'esempio applicativo) , con la particolarità di non avere più priorità per i lettori ma equità di accesso.

# Soluzione starvation-free



Nella seconda soluzione, sono stati creati:

- 20 threads di lettura
- 5 threads di scrittura

Essi scrivono/leggono una variabile intera ***risorsa condivisa***.

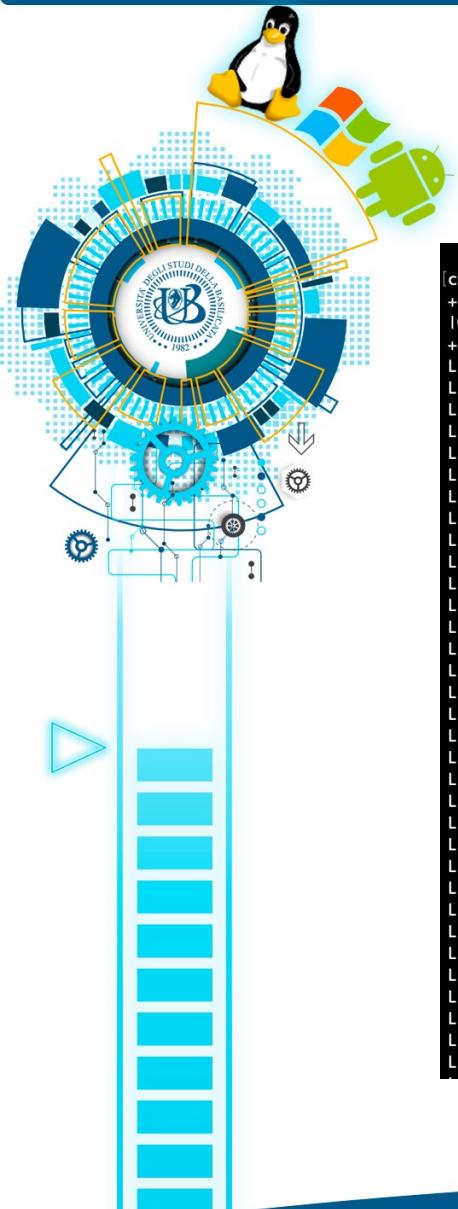
Una variabile intera ***conta lettori***(inizializzata a 0) viene utilizzata per contare il numero di lettori nella coda.

Viene utilizzata la libreria ***pthread.h*** per la creazione dei thread e tre semafori:

- ***sem\_t sem\_contatore***: per l'aggiornamento di ***conta lettori***
- ***sem\_t sem\_verifica***: per verificare l'ammissione alla sezione critica
- ***sem\_t sem\_starv***: per evitare la priorità ai lettori e prevenire la starvation.

La differenza tra questa soluzione e la soluzione con priorità dei lettori, è l'utilizzo del semaforo aggiuntivo ***sem\_starv***.

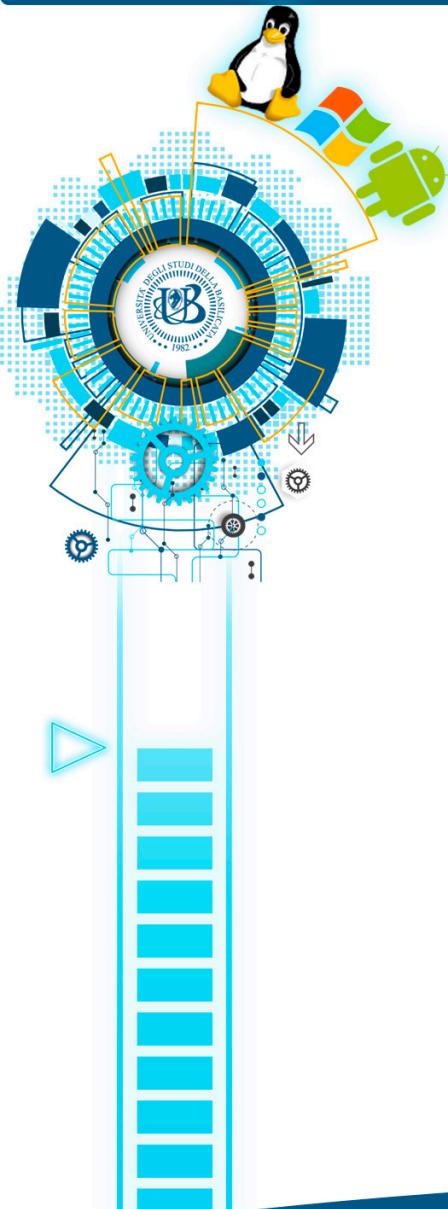
# Esecuzione codice starvation-free



```
[christiandaraio@MBPdichristian progetto_SO % ./progetto_starv_free
+-----+-----+-----+
| Questa applicazione simula un sistema di controllo delle tariffe di un biglietto aereo in continuo aggiornamento |
+-----+-----+-----+
L'utente n. 1 e' nella ZONA_INGRESSO
L'utente n. 1 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 15
L'utente n. 1 e' nella ZONA USCITA
L'utente n. 6 e' nella ZONA_INGRESSO
L'utente n. 6 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 15
L'utente n. 6 e' nella ZONA USCITA
L'utente n. 15 e' nella ZONA_INGRESSO
L'utente n. 15 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 15
L'utente n. 15 e' nella ZONA USCITA
L'utente n. 17 e' nella ZONA_INGRESSO
L'utente n. 17 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 15
L'utente n. 17 e' nella ZONA USCITA
L'utente n. 18 e' nella ZONA_INGRESSO
L'utente n. 18 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 15
L'utente n. 18 e' nella ZONA USCITA
L'utente n. 0 e' nella ZONA_INGRESSO
L'utente n. 2 e' nella ZONA_INGRESSO
L'utente n. 14 e' nella ZONA_INGRESSO
L'operatore n. 4 e' nella ZONA_INGRESSO
L'utente n. 3 e' nella ZONA_INGRESSO
L'utente n. 0 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 15
L'utente n. 0 e' nella ZONA USCITA
L'utente n. 2 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 15
L'utente n. 2 e' nella ZONA USCITA
L'utente n. 14 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 15
L'utente n. 14 e' nella ZONA USCITA
L'operatore n. 4 della compagnia aerea ha aggiornato il costo del biglietto per Londra. Valore = 75
L'operatore n. 4 e' nella ZONA USCITA
L'utente n. 3 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 75
L'utente n. 3 e' nella ZONA USCITA
L'utente n. 8 e' nella ZONA_INGRESSO
L'utente n. 8 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 75
L'utente n. 8 e' nella ZONA USCITA
```

```
L'utente n. 8 e' nella ZONA USCITA
L'utente n. 4 e' nella ZONA_INGRESSO
L'utente n. 4 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 75
L'utente n. 4 e' nella ZONA USCITA
L'utente n. 7 e' nella ZONA_INGRESSO
L'utente n. 7 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 75
L'utente n. 7 e' nella ZONA USCITA
L'utente n. 11 e' nella ZONA_INGRESSO
L'utente n. 11 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 75
L'utente n. 11 e' nella ZONA USCITA
L'operatore n. 3 e' nella ZONA_INGRESSO
L'operatore n. 3 della compagnia aerea ha aggiornato il costo del biglietto per Londra. Valore = 69
L'operatore n. 3 e' nella ZONA USCITA
L'operatore n. 0 e' nella ZONA_INGRESSO
L'operatore n. 0 della compagnia aerea ha aggiornato il costo del biglietto per Londra. Valore = 77
L'operatore n. 0 e' nella ZONA USCITA
L'utente n. 5 e' nella ZONA_INGRESSO
L'utente n. 5 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 77
L'utente n. 10 e' nella ZONA_INGRESSO
L'utente n. 10 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 77
L'utente n. 10 e' nella ZONA USCITA
L'operatore n. 2 e' nella ZONA_INGRESSO
L'operatore n. 2 della compagnia aerea ha aggiornato il costo del biglietto per Londra. Valore = 40
L'operatore n. 2 e' nella ZONA USCITA
L'utente n. 5 e' nella ZONA USCITA
L'operatore n. 1 e' nella ZONA_INGRESSO
L'operatore n. 1 della compagnia aerea ha aggiornato il costo del biglietto per Londra. Valore = 61
L'operatore n. 1 e' nella ZONA USCITA
L'utente n. 13 e' nella ZONA_INGRESSO
L'utente n. 16 e' nella ZONA_INGRESSO
L'utente n. 16 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 61
L'utente n. 16 e' nella ZONA USCITA
L'utente n. 13 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 61
L'utente n. 13 e' nella ZONA USCITA
L'utente n. 12 e' nella ZONA_INGRESSO
L'utente n. 12 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 61
L'utente n. 12 e' nella ZONA USCITA
L'utente n. 9 e' nella ZONA_INGRESSO
L'utente n. 9 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 61
L'utente n. 9 e' nella ZONA USCITA
L'utente n. 19 e' nella ZONA_INGRESSO
L'utente n. 19 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 61
L'utente n. 19 e' nella ZONA USCITA
christiandaraio@MBPdichristian progetto_SO %
```

# Esecuzione codice starvation-free



Dall'output seguente è chiaro che né il processo lettore né il processo scrittore hanno la priorità.

Gli scrittori sono in grado di scrivere anche se i lettori stanno aspettando. Inoltre anche se ci sono lettori in coda, gli scrittori entrano nella sezione di ingresso(vedi immagine). Questo non accadeva nella soluzione precedente con priorità dei lettori. Quindi lo scrittore è in grado di scrivere anche se un processo lettore è in coda. Di conseguenza la situazione di starvation è evitata: ogni thread accede alla risorsa.

```
L'utente n. 14 e' nella ZONA_INGRESSO
L'operatore n. 4 e' nella ZONA_INGRESSO
L'utente n. 3 e' nella ZONA_INGRESSO
L'utente n. 0 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 15
L'utente n. 0 e' nella ZONA_USCITA
L'utente n. 2 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 15
L'utente n. 2 e' nella ZONA_USCITA
L'utente n. 14 sta visualizzando la tariffa del biglietto con destinazione <Londra>. Costo biglietto = 15
L'utente n. 14 e' nella ZONA_USCITA
```



Il repository contenente le due soluzioni al problema dei lettori-scrittori con riferimento all'esempio applicativo è disponibile al seguente link:

<https://github.com/ChristianDaraio/Lettori-Scrittori-Sistemi-Operativi>