Esercitazione n.2
30 Ottobre 2017

Obiettivi:

- Programmazione **`pthreads`**:
  - Sincronizzazione thread posix:
    - mutua esclusione: pthread_mutex
    - semafori

Richiami esercizi esercitazioni precedenti

Esercizio 1.2 - Mutua esclusione

Una rete televisiva vuole realizzare un sondaggio di opinione su un campione di N persone riguardante il gradimento di K film.

Il sondaggio prevede che ogni persona interpellata risponda a K domande, ognuna relativa ad un diverso film.

In particolare, ad ogni domanda l'utente deve fornire una risposta (un valore intero appartenente al dominio [1,..10]) che esprime il voto assegnato dall'utente al film in questione.

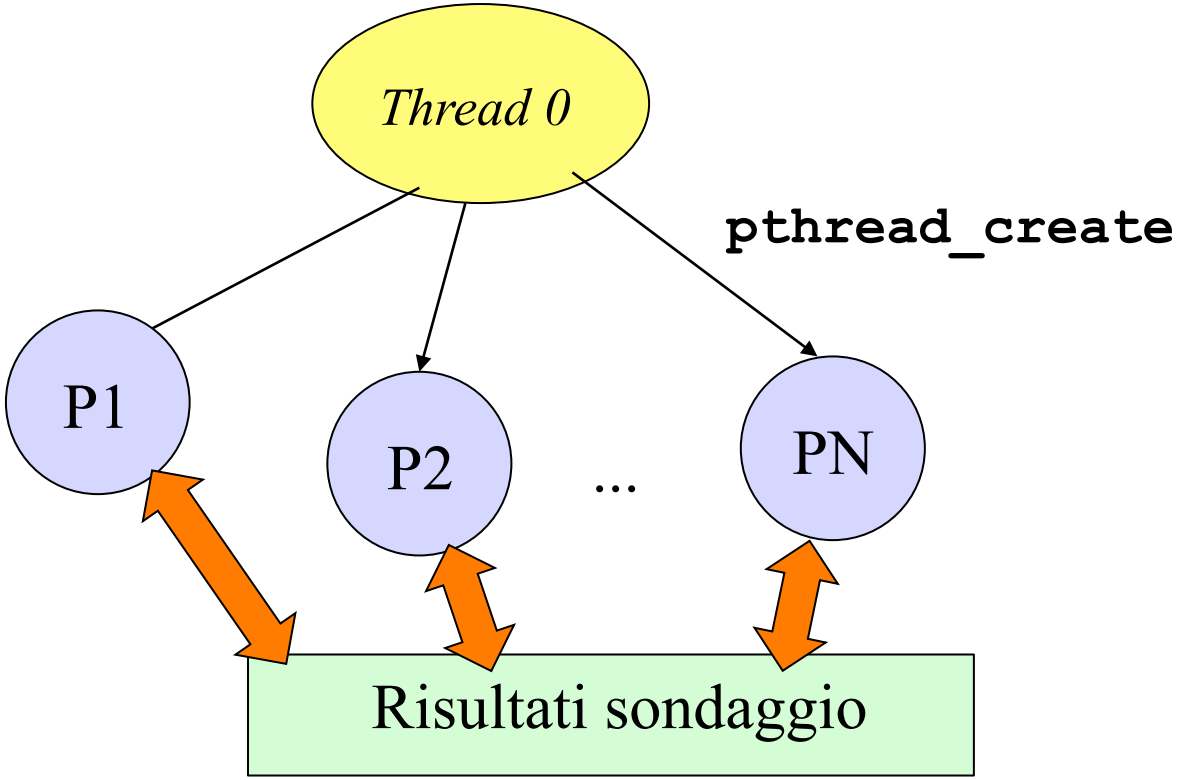
La raccolta delle risposte avviene in modo tale che, al termine della compilazione di ogni questionario, vengano presentati i risultati parziali del sondaggio, e cioè: per ognuna delle k domande, venga stampato il voto medio ottenuto dal film ad essa associato.

Al termine del sondaggio devono essere stampati i risultati definitivi, cioè il voto medio ottenuto da ciascun film ed il nome del film con il massimo punteggio.

Si realizzi un'applicazione concorrente che, facendo uso della libreria pthread e rappresentando ogni singola persona del campione come un thread concorrente, realizzi il sondaggio rispettando le specifiche date.

Spunti & suggerimenti (1)

- Persona del campione= thread
- Risultati del sondaggio: struttura dati **condivisa** composta da K elementi (1 per ogni domanda/film)



MUTUA ESCLUSIONE

- I thread spettatori dovranno accedere in modo mutuamente esclusivo alla variabile che rappresenta i risultati del sondaggio.
- Quale strumenti utilizzare?

  `pthread_mutex` o **`semaphore`**

Esercizio 1.3 – sincronizzazione a barriera

Si riconsideri il sondaggio di cui all'esercizio 2.

La rete televisiva vuole utilizzare i risultati del sondaggio per stabilire **quale dei K film interessati dalle domande del questionario mandare in onda**, secondo le seguenti modalità.

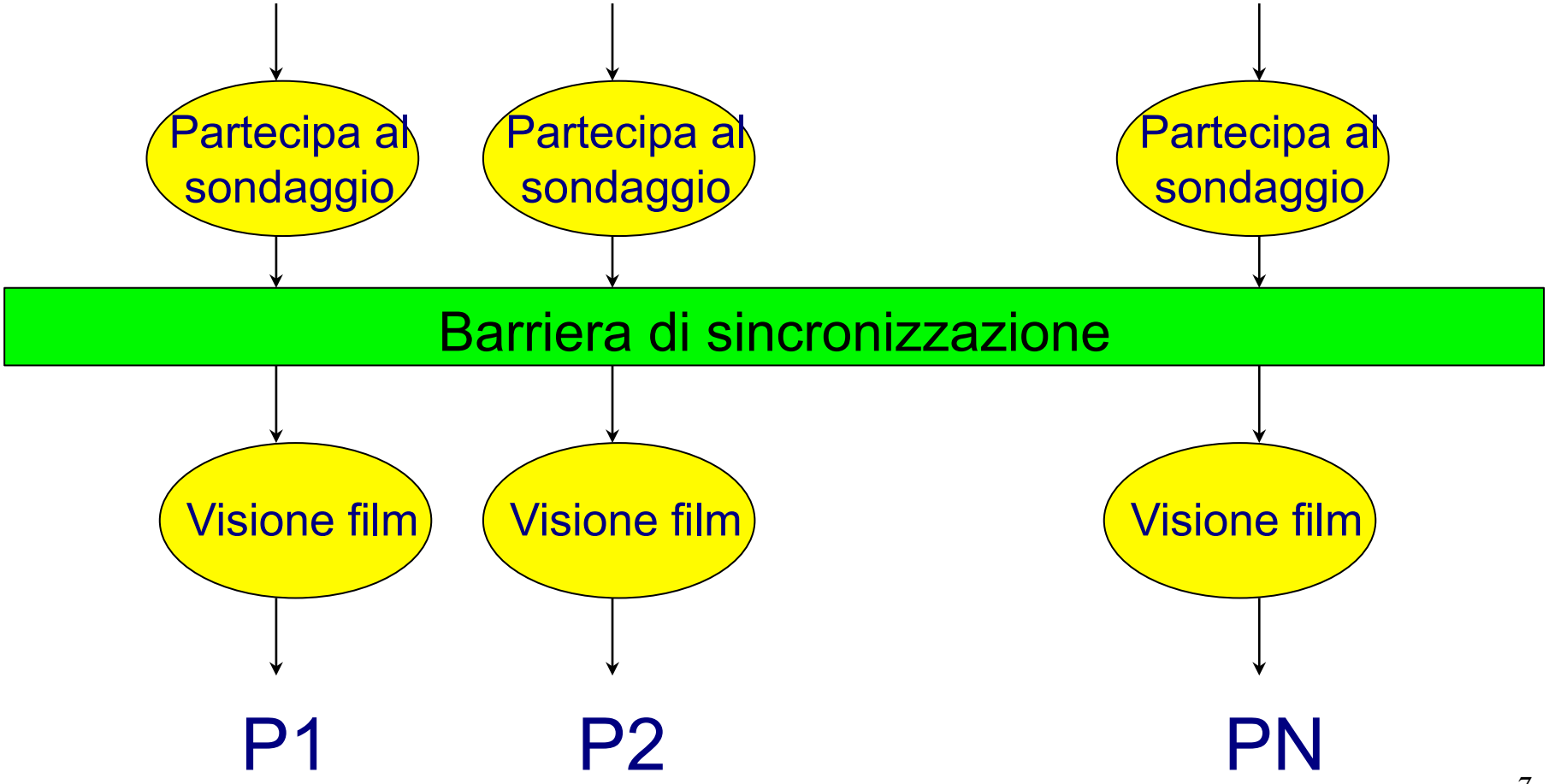
Ognuno degli N utenti ha un comportamento strutturato in **due fasi consecutive**:

1. Nella prima fase partecipa al **sondaggio**
2. Nella seconda fase **vede il film** risultato **vincitore** nel sondaggio (quello, cioè, con la valutazione massima).

Si realizzi un'applicazione concorrente nella quale ogni thread rappresenti un diverso utente, che tenga conto dei vincoli dati e, in particolare, che **ogni utente non possa eseguire la seconda fase** (visione del film vincitore) se prima non si è conclusa la fase precedente (compilazione del questionario) **per tutti gli utenti.**

Spunti & suggerimenti

Rispetto all'esercizio 1 è richiesta l'aggiunta di una **barriera di sincronizzazione** per tutti i thread concorrenti:



Barriera: possibile soluzione (pseudicodice)

- **Variabili condivise:**

```
semaphore mutex=1:
semaphore barriera=0;
int completati=0;
```

Struttura del thread i-simo Pi:

```
<operazione 1 di Pi>

p(mutex);
completati++;
if (completati==N)
   v(barriera);
v(mutex);
p(barriera);
v(barriera);

<operazione 2 di Pi>
```

Esercizio 2.1

Politiche di sincronizzazione basate su priorità:

- Estendere il problema 1.3 assumendo che la fase di visione del film vincitore richieda che ogni thread, prima di vedere il film, debba eseguire un'operazione di **download** del file.
- Si assuma che sia fissato un **numero massimo MAX di download** contemporanei consentiti, oltre al quale ogni ulteriore richiesta di download deve essere messa in attesa.
- Si assuma, infine, che il server, nell'autorizzare i download applichi una **politica che privilegi gli utenti in base alla media dei voti dati**: la priorità deve essere data all'utente la cui media dei voti (arrotondata all'intero superiore) è maggiore.

Suggerimenti

Fasi attraversate da ogni thread:

1. <partecipa a sondaggio>
2. <barriera di sincronizzazione>
3. **<richiedi download film vincitore>**
4. ***...download film...***
5. **<termina download>**
6. <visione film>

- Analogia con il pool di risorse equivalenti
- Politica con priorità: semafori privati.