Dining philosophers problem

STEFANO PRIOLO 63836
FRANCESCO PIO NARDIELLO 63914

A.A. 2022/2023

Introduzione al problema

Il problema dei cinque filosofi è un esempio di sincronizzazione fra processi paralleli. Ci sono dunque cinque filosofi, cinque piatti e cinque forchette e ogni filosofo deve avere una forchetta a destra e una a sinistra.

Ogni filosofo alterna periodi in cui mangia e periodi in cui pensa e per nutrirsi ha bisogno di due forchette, che vengono utilizzate una alla volta.

Dopo essere riuscito a prendere due forchette il filosofo mangia per un po', poi lascia le forchette e ricomincia a pensare.

Introduzione al problema

I problemi che possono verificarsi nella soluzione di questo algoritmo sono lo stallo e la starvation.
Lo **stallo o deadlock** è la situazione in cui due o più processi si bloccano a vicenda, aspettando che uno esegua una certa azione che serve all'altro o viceversa. Nel nostro caso questo problema si verifica quando ciascuno dei filosofi prende una forchetta senza mai riuscire a prendere l'altra.

La **starvation** si verifica quando un thread non ottiene le risorse hardware/software di cui necessita per essere eseguito. Questa situazione nel problema dei filosofi a cena si verifica quando uno dei filosofi non riesce mai a prendere entrambe le forchette.

I due problemi sono indipendenti l'uno dall'altro.

Stallo

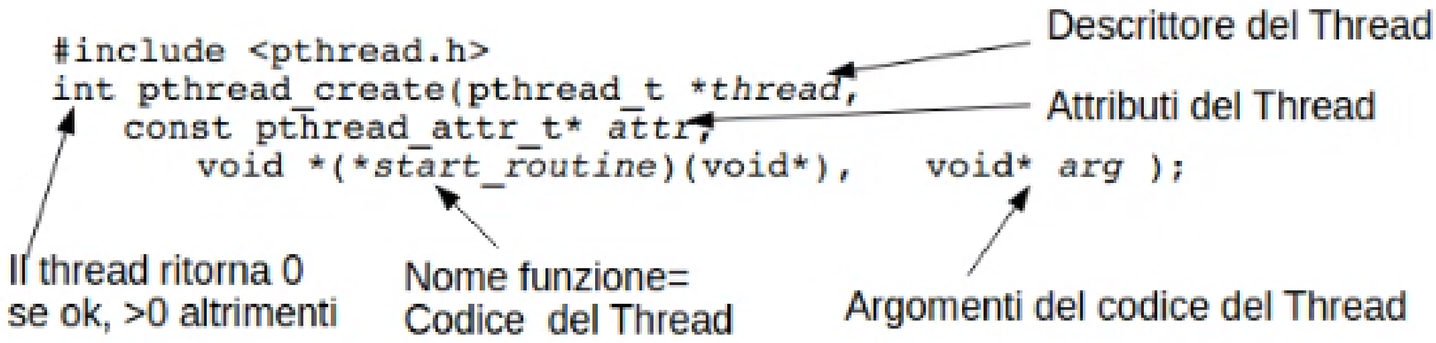
Si ha deadlock se si verificano simultaneamente le seguenti condizioni:

- **Mutua esclusione** : almeno una risorsa deve poter essere acceduta da un solo processo alla volta (gli altri vengono messi in attesa) ;
- **Possesso e attesa** : un processo possiede una risorsa ed è in attesa per un’altra risorsa;
- **Non-preemptive** : una risorsa posseduta da un processo non può essere rilasciata se non per spontanea volontà del processo stesso
- **Attesa circolare** : {P0, P1, ..., PN}, P0 attende una risorsa posseduta da P1, P1 attende una risorsa posseduta da P2, ..., PN attende una risorsa posseduta da P0

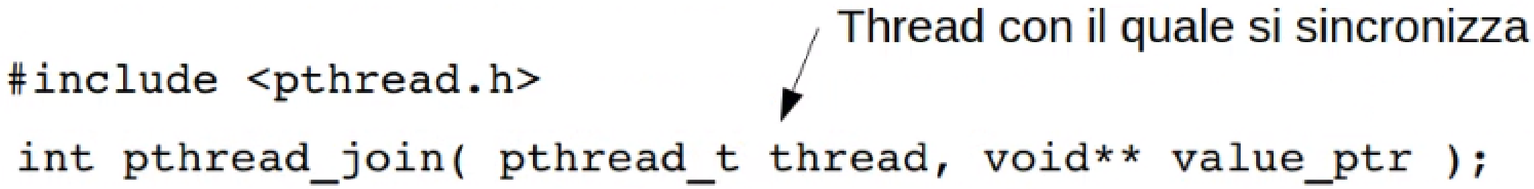
Libreria PTHREAD

E' una libreria che consente ad un programma di controllare più flussi di lavoro diversi che si sovrappongono nel tempo. I Thread Posix consentono di generare nuovi flussi di processi concorrenti.

- **Creazione del thread :**



- **Join del thread :**



La funzione **pthread_create** crea una thread e lo rende eseguibile, cioè lo mette a disposizione dello scheduler che prima o poi lo farà partire. **Pthread_t** é il tipo di dato utilizzato per identificare un thread.
Invece, la funzione **pthread_join()** sospende l'esecuzione del thread chiamante finché il thread di destinazione non termina, a meno che il thread di destinazione non sia già terminato.

Libreria SEMAPHORE

E' una libreria che permette di utilizzare i semafori. I semafori sono un meccanismo di sincronizzazione utilizzato per coordinare le attività di più processi in un sistema informatico. Sono utilizzati per imporre la mutua esclusione, per evitare race condition e per implementare la sincronizzazione tra i processi.

I principali metodi e struttre di dati di questa libreria sono:

- **sem_t sem_name:** dichiara una variabile di tipo semaforo;
- **int sem_init(sem_t *sem, int pshared, unsigned int value):** inizializza il semafore sem al valore value. La variabile pshared indica se il semaforo è condiviso tra thread (uguale a 0) o processi (diverso da 0).
- La funzione **sem_wait(sem_t *sem)** decrementa (blocca) il semaforo puntato da sem. Se il valore del semaforo è maggiore di zero, allora il decremento procede e la funzione ritorna. Se il semaforo attualmente ha il valore zero la chiamata si blocca finché non diventa possibile eseguire il decremento.
- La funzione **sem_post(sem_t *sem)** incrementa (sblocca) il semaforo puntato da sem.

Codice

```c
int main(){
    int i;
    pthread_t fil_id[NUM_FIL];
    sem_init(&mutex,0,1);
    for (i = 0; i < NUM_FIL; i++){
        sem_init(&F[i],0,0);
    }
    for (i = 0; i < NUM_FIL; i++) {
    pthread_create(&fil_id[i],NULL,filosofo, &fil[i]);
    printf("Filosofo %d PENSA\n",i+1);
    }
    for (i = 0; i < NUM_FIL; i++){
        pthread_join(fil_id[i],NULL);
    }
}
```

- Inizializzazione del semaforo condiviso dai thread al valore 1 e dell'array relativo ai thread;
- Creazione dei threads che rappresentano i filosofi;
- Nella sincronizzazione dei thread il join permette ad un thread di sospendere la propria esecuzione in attesa che venga completata quella di un altro thread

Codice

```
void* filosofo(void* num){
    while (true) {
        int* i = num;
        sleep(1);
        prendeForchetta(*i);
        sleep(1);
        lasciaForchetta(*i);
    }
}
```

La funzione sleep permette di sospendere il thread corrente per il numero specificato di millisecondi.

```
void prendeForchetta(int filosofi){
    sem_wait(&mutex);
    stato[filosofi] = AFFAMATO;
    printf("Filosofo %d e' affamato\n", filosofi+1);
    verifica(filosofi);
    sem_post(&mutex);
    sem_wait(&F[filosofi]);
    sleep(1);
}
```

Il metodo prendeForchetta consiste, dopo aver decrementato il semaforo e dopo aver inizializzato lo stato del filosofo ad affamato, nel chiamare il metodo verifica(>>) .
Successivamente il semaforo viene sbloccato ed infine viene bloccato il semaforo F.

Codice

```
void verifica(int filosofi){
    if (stato[filosofi] == AFFAMATO && stato[SINISTRA] != MANGIA && stato[DESTRA] != MANGIA) {
        stato[filosofi] = MANGIA;
        sleep(1);
        printf("Filosofo %d prende le forchette %d e %d\n", filosofi +1, SINISTRA+1, filosofi+1);
        printf("Filosofo %d MANGIA\n", filosofi+1);
        sem_post(&F[filosofi]);
    }
}
```

La procedura verifica prende in input l'id del filosofo. Essa consiste nel verificare se l'i-esimo filosofo può utilizzare le forchette di destra e sinistra, essendo affamato. In caso affermativo il filosofo prende le forchette e mangia.

```
void lasciaForchetta(int filosofi){
    sem_wait(&mutex);
    stato[filosofi] = PENSA;
    printf("Filosofo %d lascia le forchette %d e %d \n",filosofi+1, SINISTRA+1, filosofi+1);
    printf("Filosofo %d PENSA\n", filosofi+1);
    verifica(SINISTRA);
    verifica(DESTRA);
    sem_post(&mutex);
}
```

Dopo aver mangiato lo stato del filosofo viene impostato a pensa. Successivamente, attraverso la procedura verifica, si controlla se I filosofi di sinistra e di destra possono mangiare.

L'esecuzione

```
mint@Franc:~$ cd Desktop/
mint@Franc:~/Desktop$ gcc -o Finale prova.c
mint@Franc:~/Desktop$ ./Finale
+----------BENVENUTO-----------+
|   Dining philosophers problem  |
+--------------------------------+

Filosofo 1 PENSA
Filosofo 2 PENSA
Filosofo 3 PENSA
Filosofo 4 PENSA
Filosofo 5 PENSA
Filosofo 1 e' affamato
Filosofo 2 e' affamato
Filosofo 3 e' affamato
Filosofo 4 e' affamato
Filosofo 5 e' affamato
Filosofo 5 prende le forchette 4 e 5
Filosofo 5 MANGIA
Filosofo 5 lascia le forchette 4 e 5
Filosofo 5 PENSA
Filosofo 4 prende le forchette 3 e 4
Filosofo 4 MANGIA
Filosofo 1 prende le forchette 5 e 1
Filosofo 1 MANGIA
Filosofo 4 lascia le forchette 3 e 4
Filosofo 4 PENSA
Filosofo 3 prende le forchette 2 e 3
Filosofo 3 MANGIA
Filosofo 5 e' affamato
Filosofo 1 lascia le forchette 5 e 1
Filosofo 1 PENSA
Filosofo 5 prende le forchette 4 e 5
Filosofo 5 MANGIA
Filosofo 4 e' affamato
```

Da come si evince dall'output le situazioni critiche che si potevano verificare, deadlock e starvation, sono state evitate, dato che ogni thread accede alla risorsa: ogni filosofo usa sempre le due forchette per mangiare e nel frattempo gli altri attendono affamati. In esempio si può notare facilmente che un filosofo dopo aver preso due forchette, mangia e successivamente le lascia in modo tale che altri filosofi possano prenderle.

Come forzare il deadlock

```
+-----------BENVENUTO-------------+
|   Dining philosophers problem   |
+---------------------------------+

Filosofo 1 PENSA
Filosofo 2 PENSA
Filosofo 3 PENSA
Filosofo 4 PENSA
Filosofo 5 PENSA
Filosofo 1 e' affamato
Filosofo 2 e' affamato
Filosofo 4 e' affamato
Filosofo 5 e' affamato
Filosofo 5 prende le forchette 4 e 5
Filosofo 5 MANGIA
Filosofo 3 e' affamato
Filosofo 3 prende le forchette 2 e 3
Filosofo 3 MANGIA
Filosofo 1 e' affamato
Filosofo 4 e' affamato
```

Questa modifica al codice potrebbe potenzialmente portare a una situazione di stallo, poiché tutti i filosofi potrebbero prendere contemporaneamente la forchetta di sinistra e quindi attendere che la forchetta di destra diventi disponibile.

Questa situazione può causare un blocco completo del programma.

```
void* filosofo(void* num){
    while (true) {
    int* i = num;
    sleep(1);
    prendeForchetta(*i);
    prendeForchetta((*i+1)%NUM_FIL);
    sleep(1);
    lasciaForchetta(*i);
    lasciaForchetta((*i+1)%NUM_FIL);
    }
}
```

Dall'output si evince che il filosofo 1 è in attesa della forchetta posseduta dal filosofo 5, mentre il filosofo 4 è in attesa delle forchette 3 e 4.

Link a GITHUB

Di seguiito il link al progetto svolto:

https://github.com/francesco2706/Progetto-SO---Filosofi_Priolo-Nardiello

oppure eseguendo il seguente comando da terminale:

git clone https://github.com/francesco2706/Progetto-SO---Filosofi_Priolo-Nardiello