Elaborazione concorrente

Interazioni tra processi

Sincronizzazione di processi

Struttura di un programma per la stampa di un estratto conto bancario

```
while (esisteCliente) {
  leggiMovimenti();
  stampaRiepilogo();
}
```

Con questa elaborazione sequenziale si usano due risorse: il disco e la stampante; mentre si usa una risorsa l'altra è inutilizzata

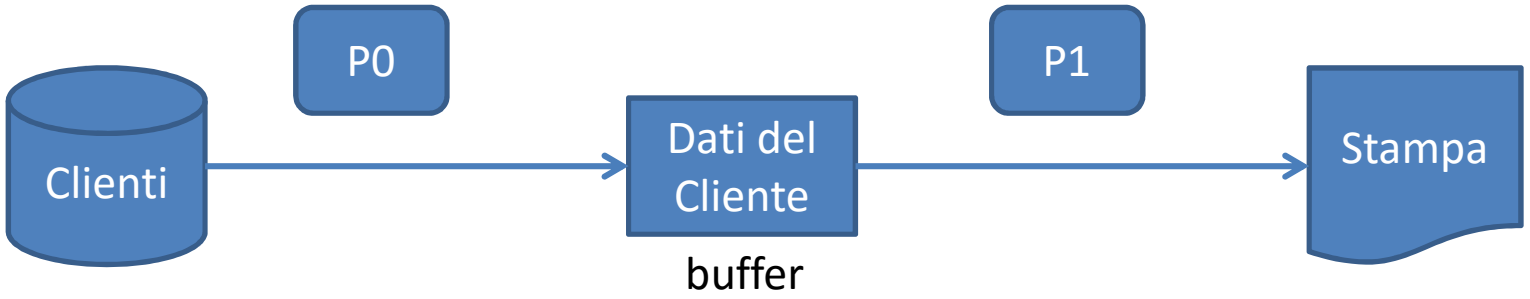
Sincronizzazione di processi

Processo P0

```
while (esisteCliente) {
  leggiMovimenti();
  scriviDatiNelBuffer();
}
```

Processo P1

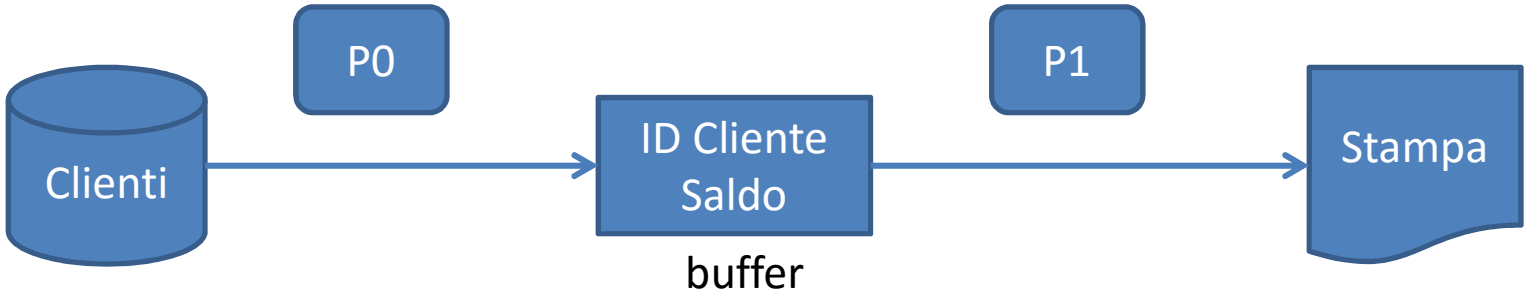
```
while (esisteCliente) {
  leggiDatiDalBuffer();
  StampaEstrattoConto();
}
```



Il buffer consente a P0 di elaborare altri dati.
Il processo di stampa non deve attendere che P0 produca nuovi dati

Corsa critica

Se P0 è più veloce di P1, P0 sovrascrive i dati



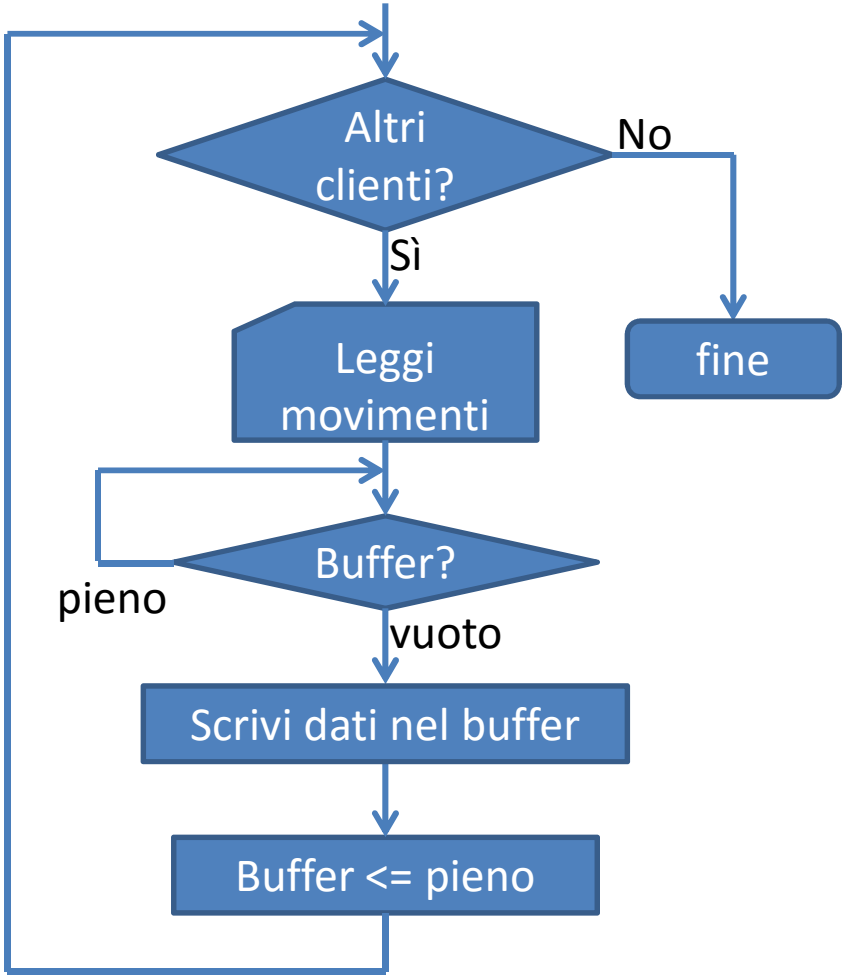
Se P1 è più veloce di P0, P1 usa gli stessi dati

Per effetto della concorrenza, inoltre, le operazioni di un processo potrebbero essere intercalate tra le operazioni dell'altro processo:

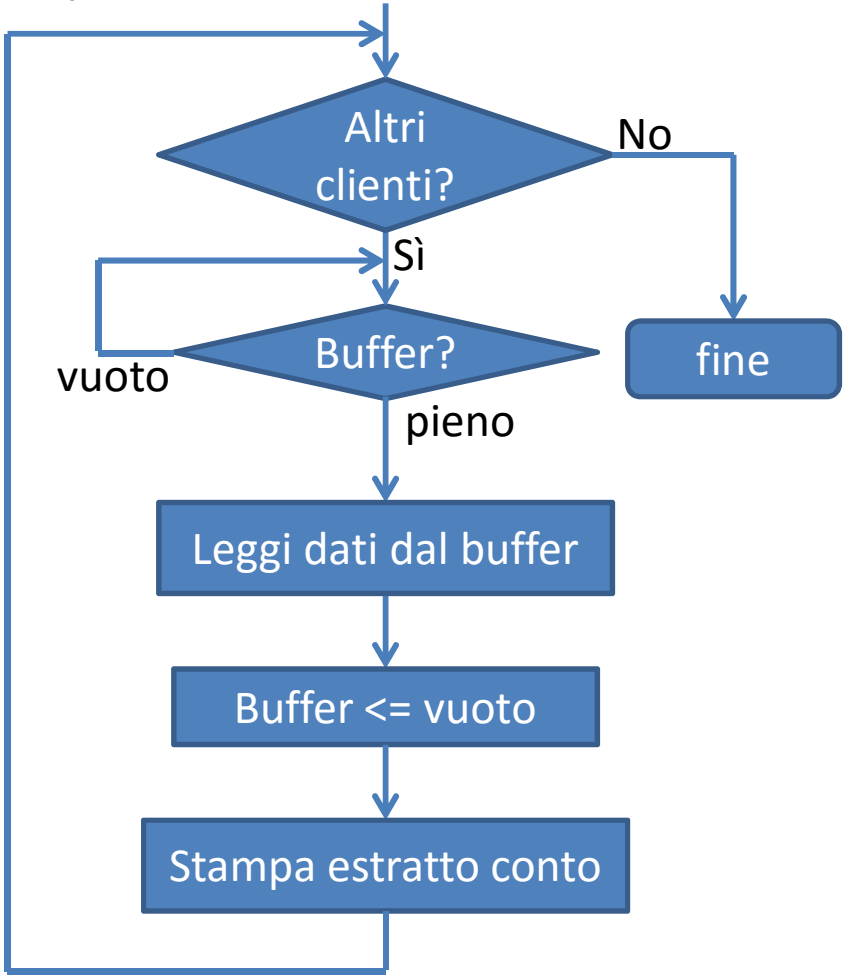
```
scriviDatiNelBuffer() (IDCliente)     (P0) (Timeout)
leggiDatiDalBuffer()                  (P1)
StampaEstrattoConto()                 (P1)
scriviDatiNelBuffer() (Saldo)         (P0) (Continua)
LeggiMovimenti()                      (P0)
```

Regole di sincronizzazione

- P0 può scrivere nel buffer se il buffer è stato svuotato



P1 può leggere dal buffer se è stato depositato un nuovo dato



```
Class estrattoConto {
  static boolean pieno = false;
  public static void main(String args[]) {
    P0();  // nel main vengono creati due processi e ognuno viene
    P1();  // eseguito su un processore. Essi vengono eseguiti in parallelo
  }
  void P0() {
    while (esisteCliente) {
      leggiMovimenti();
      while(pieno) {}
      scriviDatiNelBuffer();
      pieno = true;
    }
  }
  void P1() {
    while (esisteCliente) {
      while(!pieno) {}
      leggiDatiDalBuffer();
      pieno = false;
      stampaEstrattoConto();
    }
  }
}
```

Il problema della mutua esclusione

- La variabile *pieno* di tipo semaforo, realizza la sincronizzazione dei processi, costringendoli ad avanzare alla stessa velocità.
  - Se il buffer riesce a contenere dati di molti clienti, i due processi possono avanzare anche a velocità diverse
- Si verifica un'attesa attiva, in cui un processo esegue il test sul semaforo impegnando inutilmente la CPU.
  - Un processo che non può continuare deve passare nello stato di attesa.

Sezione critica

Prenotazione posti

Processo i-mo in esecuzione su un terminale:

```
while (true) {
   cerca posto libero
   segna posto occupato
}
```

Una volta iniziata la prima operazione, si deve svolgere anche la seconda, senza interruzioni.

Potrebbe accadere che dopo aver svolto la prima operazione, il processo viene sospeso. Se un altro processo iniziasse la stessa ricerca, potrebbe trovare lo stesso posto e entrambi lo riserverebbero erroneamente.

Prima di entrare in una sezione critica, bisogna disabilitare le interruzioni e dopo essere usciti bisogna riabilitare le interruzioni

Test and Set

- Una speciale istruzione macchina, che viene eseguita senza interruzione:
- Viene interrogato il valore di una variabile booleana e il suo valore viene impostato a 1
- Se la variabile booleana è già 1, il processo aspetta che diventi 0.

```
lock: tsl AL, bool   (copia bool in un registro)
cmp AL, 0
jnz lock             (se bool era 1, aspetta)
bool = 1
ret
```

Test And Set

- L’istruzione Test And Set su una variabile booleana autorizza un solo processo ad entrare in una sezione critica protetta dal semaforo booleano.
- Il primo processo che trova il semaforo nello stato “false” accede alla sezione critica
- Il processo che trova il semaforo “true” resta in attesa attiva.
- La soluzione è applicabile anche a sistemi multiprocessore, purché la variabile si trovi nella memoria condivisa e non nella cache del processore