Programmazione Concorrente e Algoritmi

Definizioni

Programmazione concorrente

Tecnica di programmazione in cui più attività (processi, thread, etc.) vengono eseguite simultaneamente.

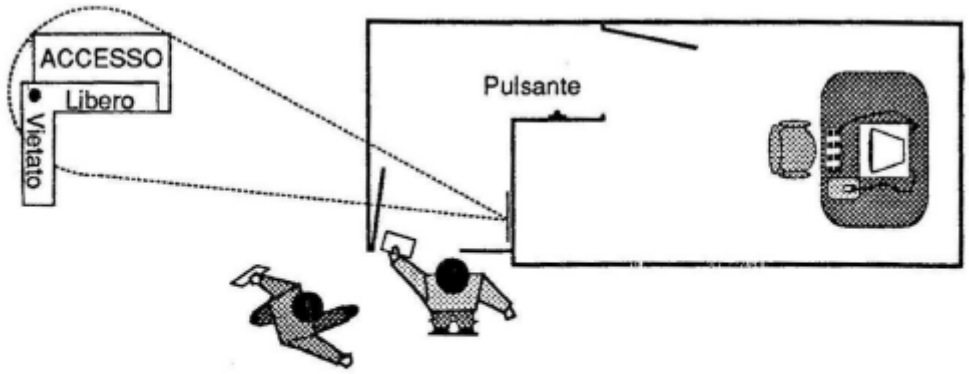
- L'obiettivo della programmazione concorrente è quello di migliorare l'efficienza e l'ottimizzazione delle risorse del sistema, sfruttando al massimo la capacità di elaborazione parallela delle moderne architetture hardware.

Sezione Critica

Parte del codice che deve essere eseguita in modo esclusivo, ovvero da un solo processo o thread alla volta, per evitare problemi come la race condition.

- Race condition = "gara" per vedere quale processo arrivi prima
- Durante l'esecuzione di una sezione critica, è necessario garantire che nessun altro processo o thread possa accedere alle risorse condivise.
- Gli algoritmi di Dekker e Peterson sono utilizzati per gestire le sezioni critiche in modo sicuro in ambienti concorrenti

Quando uno dei due è libero, l'altro può accedere:



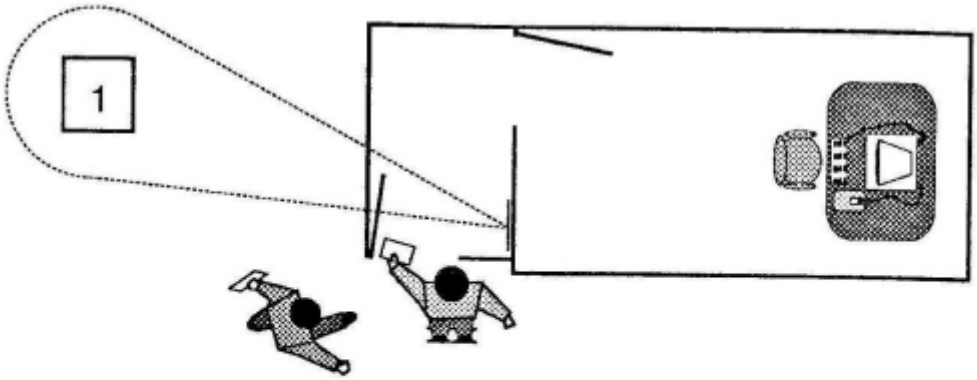
Algoritmo 1

```
boolean DISPONIBILE;
Procedura P1;
Procedura P2;
begin
   DISPONIBILE = true;
   cobegin
      P1; P2;
   coend
end
```

```
Procedura P1
   begin
      ripeti
         while (!DISPONIBILE) do NOP;
         DISPONIBILE = false
         Sezione critica 1;
         DISPONIBILE = true
         Istruzioni 1;
      until false
   end
```

```
Procedura P2
   ....
```

Uno dei due può prendere accesso:



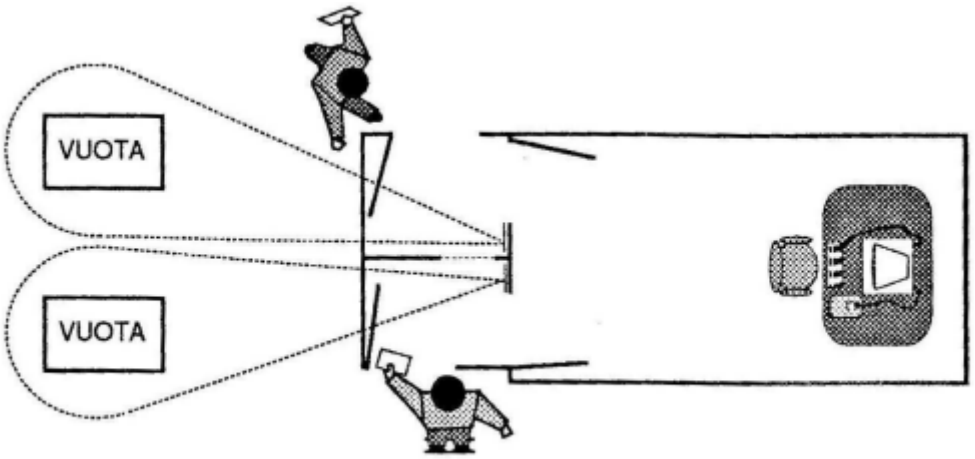
Algoritmo 2

```
integer TOCCA_A;
Procedura P1;
Procedura P2;
begin
   TOCCA_A = 1;
   cobegin
      P1; P2;
   coend
end
```

```
Procedura P1
   begin
      ripeti
         while (TOCCA_A = 2) do NOP;
         Sezione critica 1;
         TOCCA_A = 2;
         Istruzioni 1;
      until false
   end
```

```
Procedura P2
   ....
```

Completamente, diventa così:



Algoritmi 3

```
string KEY1, KEY2;
Procedura P1;
Procedura P2;
begin
   KEY1 = 'LIBERA'; KEY2 = 'LIBERA';
   cobegin
      P1; P2;
   coend
end
```

```
Procedura P1
   begin
      ripeti
         while (KEY2 = 'OCCUPATA') do NOP;
         KEY1  = 'OCCUPATA';
         Sezione critica 1;
         KEY1  = 'LIBERA';
         Istruzioni 1;
      until false
   end
```

```
Procedura P2
   ....
```

```
Procedura P1
   begin
      ripeti
         KEY1  = 'OCCUPATA';
         while (KEY2 = 'OCCUPATA') do NOP;
         Sezione critica 1;
         KEY1  = 'LIBERA';
         Istruzioni 1;
      until false
   end
```

```
Procedura P2
   ....
```

Algoritmi di gestione della sezione critica

Algoritmo di Dekker

Descrizione

L'algoritmo di Dekker è un algoritmo per il controllo della concorrenza, sviluppato da Edsger W. Dijkstra nel 1965. È stato uno dei primi algoritmi progettati per garantire la mutua esclusione tra processi o thread.

L'algoritmo di Dekker è noto per essere inefficiente a causa della possibilità di starvation (attesa permanente del processo) e deadlock (stallo in attesa di risorse).

Implementazione

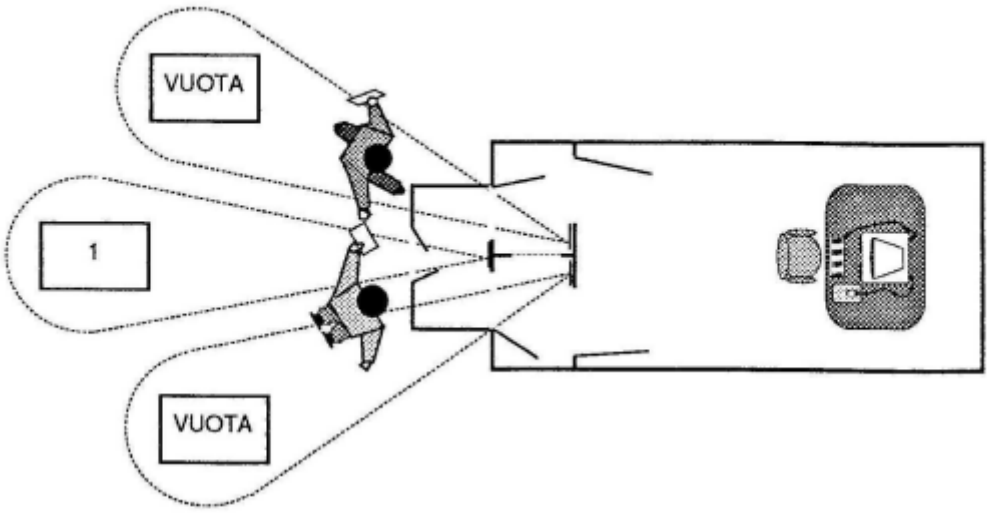
L'algoritmo di Dekker si basa su una variabile booleana condivisa per ogni processo che indica se il processo è interessato ad entrare nella sezione critica o meno. Inoltre, ci sono variabili di turno per coordinare l'accesso in modo sequenziale.

```
# Implementazione Python dell'Algoritmo di Dekker
interested = [False, False]  # Indica l'interesse dei due processi
turn = 0  # Variabile di turno

def process0():
    global interested, turn
    interested[0] = True
    while interested[1]:
        if turn == 1:
            interested[0] = False
            while turn == 1:
                pass
            interested[0] = True
    # Sezione critica
    # ...
    turn = 1
    interested[0] = False

def process1():
    global interested, turn
    interested[1] = True
    while interested[0]:
```

```
        if turn == 0:
            interested[1] = False
            while turn == 0:
                pass
            interested[1] = True
    # Sezione critica
    #  ...
    turn = 0
    interested[1] = False
```



Algoritmo DEKKER

```
string KEY1, KEY2;
integer TOCCA_A;
Procedura P1;
Procedura P2;
begin
   KEY1 = 'LIBERA'; KEY2 = 'LIBERA';
   TOCCA_A = 1;
   cobegin
      P1; P2;
   coend
end
```

```
Procedura P1
   begin
      ripeti
         KEY1  = 'OCCUPATA';
         while (KEY2 = 'OCCUPATA') do NOP;
         if (TOCCA_A == 2) {
            KEY1 = 'LIBERA';
            while (TOCCA_A = 2) do NOP;
               KEY1  = 'OCCUPATA';
         }
         Sezione critica 1;
         TOCCA_A = 2;
         KEY1  = 'LIBERA';
         Istruzioni 1
      until false
   end
```

```
Procedura P2
   ....
```

Algoritmo di Peterson

Descrizione

L'algoritmo di Peterson è un algoritmo per il controllo della concorrenza sviluppato da Gary Peterson nel 1981. È stato sviluppato come alternativa all'algoritmo di Dekker, per evitare i problemi di inefficienza e deadlock.

L'algoritmo di Peterson utilizza una variabile booleana condivisa e un vettore per memorizzare l'interesse dei processi.

Implementazione

```
# Implementazione Python dell'Algoritmo di Peterson
interested = [False, False]  # Indica l'interesse dei due processi
turn = 0  # Variabile di turno

def process0():
    global interested, turn
    interested[0] = True
    turn = 1
    while interested[1] and turn == 1:
        pass
    # Sezione critica
    # ...
    interested[0] = False

def process1():
    global interested, turn
    interested[1] = True
    turn = 0
    while interested[0] and turn == 0:
        pass
    # Sezione critica
    # ...
    interested[1] = False
```

Algoritmo PETERSON

```
string KEY1, KEY2;
integer TOCCA_A;
Procedura P1;
Procedura P2;
begin
   KEY1 = 'LIBERA'; KEY2 = 'LIBERA';
   TOCCA_A = 1;
   cobegin
      P1; P2;
   coend
end
```

```
Procedura P1
   begin
      ripeti
         KEY1  = 'OCCUPATA';
         TOCCA_A = 2;
         while (KEY2 = 'OCCUPATA' and TOCCA_A = 2) do NOP;
            Sezione critica 1;
         KEY1  = 'LIBERA';
         Istruzioni 1
      until false
   end
```

```
Procedura P2
   ....
```