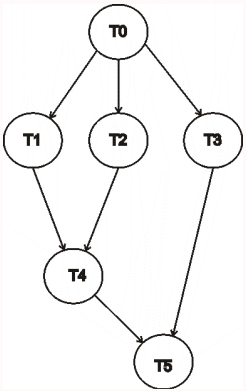
Riepilogo sulla Concorrenza

Passi per la costruzione di un programma concorrente

- Prima di tutto, definire se la concorrenza avviene mediante Processi concorrenti o Thread concorrenti
- Definizione delle precedenze tra processi/thread: descritti mediante il grafo delle precedenze
- Ulteriore elaborazione del grafo: semplificazione delle precedenze, determinazione del grado di parallelismo, definizione delle funzioni dei moduli
- Determinatezza del grafo (semplice se la concorrenza avviene tra processi)
- Definizione della modalitá di implementazione: scelta del linguaggio e degli strumenti linguistici da utilizzare
- Dalla definizione degli strumenti linguistici deriva la modalitá di sincronizzazione da utilizzare

Esempio

- Supponiamo di voler realizzare il seguente grafo:



- Le unitá computazionali da realizzare in concorrenza possono essere definite ad esempio come segue:
  - T0: inizializza le variabili a, b, c, d, e, f
  - T1: $t1 = a + b$
  - T2: $t2 = c + d$
  - T3: $t3 = e/f$
  - T4: $t4 = t1 * t2$
  - T5: $t5 = t4 - t3$
- Il grafo calcola la funzione $(a + b)(c + d) - e/f$

Implementazione

Pseudocodice con cobegin/coend

```
cobegin
   begin
      cobegin
         T1(); T2();
      coend
      T4();
   end
   T3();
coend
```

Implementazione

Pseudocodice con i thread di Java

```
t0.start(); t0.join();
t1.start(); t2.start(); t3.start();
t1.join(); t2.join();
t4.start(); t4.join();
t3.join();
t5.start(); t5.join();
```

Implementazione

Codice reale con i thread di Java: sincronizzazione tipo fork/join

```
import java.lang.*; import java.util.*;
public class prog {
    public static void main(String[] args) {

      data item= new data(1,2,3,4,5,6);//3*7-5/6=20.16666
      P1 p1=new P1(item); // istanze
      P2 p2=new P2(item);
      P3 p3=new P3(item);
      P4 p4=new P4(item);
      P5 p5=new P5(item);

      p1.start(); // calcolo concorrente
      p2.start();
      p3.start();
      try { p1.join(); } catch(InterruptedException e) {System.out.println("Errore P1");}
      try { p2.join(); } catch(InterruptedException e) {System.out.println("Errore P2");}
      p4.start();
      try { p3.join(); } catch(InterruptedException e) {System.out.println("Errore P3");}
      try { p4.join(); } catch(InterruptedException e) {System.out.println("Errore P4");}
      p5.start();
      try { p5.join(); } catch(InterruptedException e) {System.out.println("Errore P5");}
      p1.stop();
      p2.stop();
      p3.stop();
      p4.stop();
      p5.stop();
      System.out.println(" Fine programma ");
}
}
```

Implementazione

Gli strumenti base per la creazione dei processi in Linux

int fork();
Crea un nuovo processo. Al processo padre ritorna il PID del figlio (il padre puo avere piu figli) Al processo figlio ritorna 0 (un processo puo avere 1 solo padre (getppid())

void exit();
Chiude tutti i descrittori aperti, rilascia memoria.

pid_t wait(int *status);
blocca il chiamante finche un figlio termina

pid_t waitpid(pid_t pid, int *status, int options);
blocca il padre finche termina pid

int execlp(char *file, char *arg0, ..., char *argN);
invoca un altro programma, sovrascrivendo lo spazio di memoria di un processo con una copia di un programma eseguibile. NB: L'INDIRIZZO DI RITORNO E' PERSO!

Implementazione

Pseudocodice con la creazione/attesa dei processi in Unix

```
t1=fork();
if(t1==0)
   exec(T1);
else{
   t2=fork();
   if(t2==0)
       exec(T2);
   else {
       t3=fork();
       if(t3==0)
           exec(T3);
       else{
           waitpid(t1);
           waitpid(t2);
           t4=fork();
           if(t4==0)
                exec(T4);
           else{
                waitpid(t4);
                waitpid(t3);
                t5=fork();
                if(t5==0)
                    exec(T5);
                else{
                    waitpid(t5);
                }
           }
       }
   }
}
```

Implementazione: Codice reale con la creazione/attesa dei processi in Linux (RedHat)

```
#include <sys/wait.h> #include <stdlib.h> #include <unistd.h> #include <stdio.h>
int main(int argc, char *argv[])
{
    pid_t cpid, w, t1,t2,t3,t4,t5;
    int status;
    t1 = fork();
    if (t1 == 0) {            /* codice del figlio */
         printf("processo T1; PID %ld\n", (long) getpid()); sleep(1);  _exit(0);
    } else {                    /* codice del padre */
         t2 = fork();
         if (t2 == 0) {           /* figlio */
           printf("processo T2; PID %ld\n", (long)getpid()); sleep(1); _exit(0);
         } else {
             t3 = fork();
             if (t3 == 0) {           /* figlio */
                printf("processo T3; PID %ld\n", (long)getpid()); sleep(1);  _exit(0);
          } else {
            waitpid(t1, &status,0); printf("sono il padre, ho aspettato t1\n");
            waitpid(t2, &status,0); printf("sono il padre, ho aspettato t2\n");
               t4 = fork();
            if (t4 == 0) {           /* figlio */
             printf("processo T4; PID %ld\n", (long)getpid()); sleep(1); _exit(0);
              } else {
                  waitpid(t4, &status,0); printf("sono il padre, ho aspettato t4\n");
                  waitpid(t3, &status,0);  printf("sono il padre, ho aspettato t3\n");
                  t5 = fork();
                  if (t5 == 0) {           /* figlio */
                        printf("proc. T5; PID %ld\n",(long)getpid()); sleep(1); _exit(0);
                  } else {                   /* padre */
                        waitpid(t5, &status,0); printf("sono il padre, ho atteso t5\n");
                    }
             }
                }
          }
      }
}
```

Implementazione

Esempio di esecuzione del programma precedente

```
$ gcc -o oo multip1.c
$ ./oo
primo processo T1; PID 2688
secondo processo T2; PID 2689
terzo processo T3; PID 2690
sono il padre, ho aspettato t1
sono il padre, ho aspettato t2
quarto processo T4; PID 2691
sono il padre, ho aspettato t4
sono il padre, ho aspettato t3
quinto processo T5; PID 2692
sono il padre, ho eseguito e aspettato t5
$
```