Addendum alle chiamate di sistema per la gestione processi

E Mumolo

Implementazione di grafi delle precedenze

- Definiamo un processo proc.c che scrive l'argomento passato in linea:

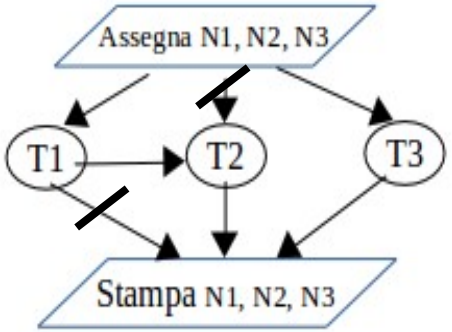
```
#include <sys/types.h> //pid_t
#include <unistd.h> //fork()
#include <sys/wait.h> //waitpid
#include <stdio.h> //printf...
#include <stdlib.h> //exit()

void main(int argc, char *argv[]){
    printf("Sono il processo %s\n", argv[1]);
}
```

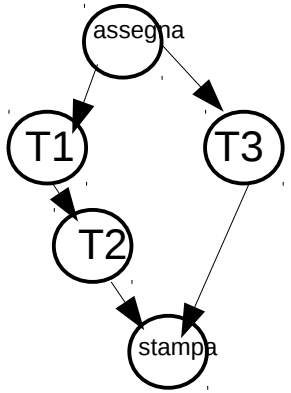
- Nei programmi d'esempio seguenti questo processo verrà eseguito così:

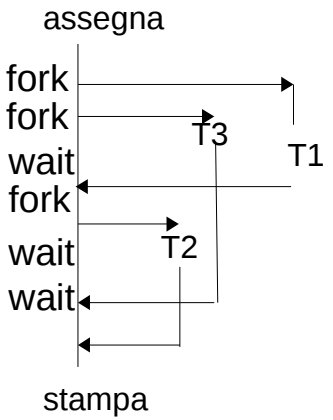
```
int execlp(const char *file, const char *arg, ...);
```

- Attenzione: exec richiede un processo generato con fork
- Altrimenti ricopre il processo
- Se non si usa exec, il codice deve essere descritto in una funzione

Implementazione di grafi delle precedenze



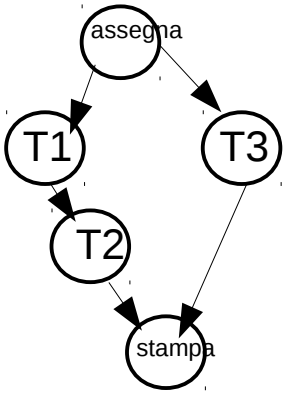


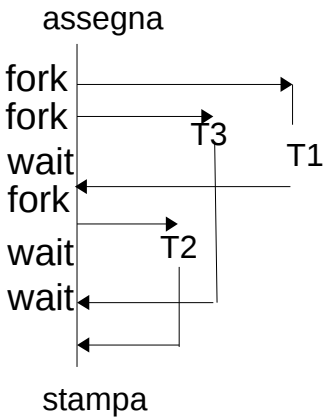



```c
#include <sys/types.h>     //pid_t
#include <unistd.h>        //fork()
#include <sys/wait.h>      //waitpid
#include <stdio.h>         //printf...
#include <stdlib.h>        //exit()

void main(){
       int pid1,pid2,pid3;

       pid1=fork();
       if ( pid1== 0)
              execlp("/home/mumolo/proc", "proc", "T1", NULL);
       else{
              pid3=fork();
              if (pid3 == 0)
                     execlp("/home/mumolo/proc", "proc", "T3", NULL);
              else{
                     waitpid(pid1, NULL, 0);

                     pid2=fork();
                     if( pid2 == 0)
                            execlp("/home/mumolo/proc","proc","T2",NULL);
                     else{
                            waitpid(pid3, NULL, 0);

                            waitpid(pid2, NULL, 0);
                     }
              }
       }
}
```

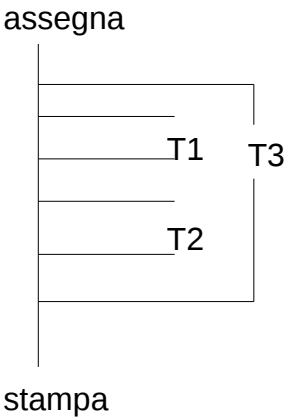
```c
#include <sys/types.h>  //pid_t
#include <unistd.h>     //fork()
#include <sys/wait.h>   //waitpid
#include <stdio.h>      //printf...
#include <stdlib.h>     //exit()
//gp-1.c
void main(){
       int pid1,pid2,pid3;

       pid1=fork();
       if(pid1==0) execlp("/home/mumolo/proc","proc","T1",NULL);

       pid3=fork();
       if(pid3==0) execlp("/home/mumolo/proc","proc","T3",NULL);

       waitpid(pid1, NULL, 0);

       pid2=fork();
       if(pid2==0) execlp("/home/mumolo/proc","proc","T2",NULL);

       waitpid(pid3, NULL, 0);

       waitpid(pid2, NULL, 0);
}
```

Implementazione di grafi delle precedenze





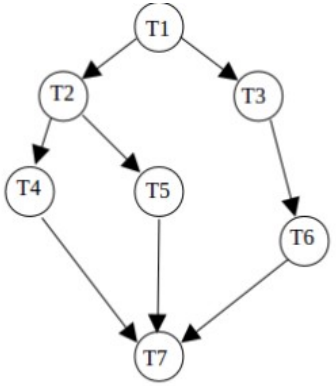
Altra sequenza
di esecuzione:

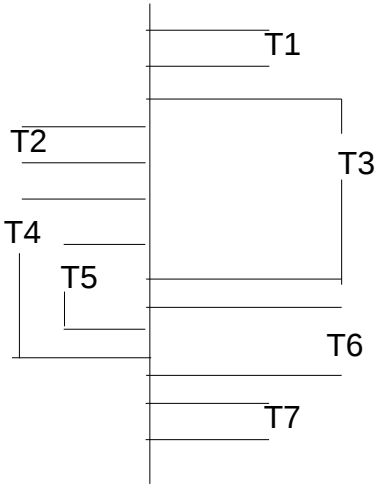



```
#include <sys/types.h>     //pid_t
#include <unistd.h>        //fork()
#include <sys/wait.h>      //waitpid
#include <stdio.h>  //printf...
#include <stdlib.h>        //exit()
//gp-2.c
void main(){
       int pid1,pid2,pid3;

       if (pid1=fork() == 0)
              execlp("/home/mumolo/proc","proc","T1",NULL);

       if (pid3=fork() == 0)
              execlp("/home/mumolo/proc","proc","T3",NULL);

       waitpid(pid1, NULL, 0);

       if( pid2=fork() == 0)
              execlp("/home/mumolo/proc","proc","T2",NULL);

       waitpid(pid3, NULL, 0);

       waitpid(pid2, NULL, 0);
}
```

```
#include <sys/types.h>     //pid_t
#include <unistd.h>        //fork()
#include <sys/wait.h>      //waitpid
#include <stdio.h>         //printf...
#include <stdlib.h>        //exit()
//gp-3.c
void main(){
       int pid1,pid2,pid3;

       if (pid3=fork() == 0)
              execlp("/home/mumolo/proc","proc","T3",NULL);

       if (pid1=fork() == 0)
              execlp("/home/mumolo/proc","proc","T1",NULL);

       waitpid(pid1, NULL, 0);

       if( pid2=fork() == 0)
              execlp("/home/mumolo/proc","proc","T2",NULL);

       waitpid(pid2, NULL, 0);

       waitpid(pid3, NULL, 0);
}
```

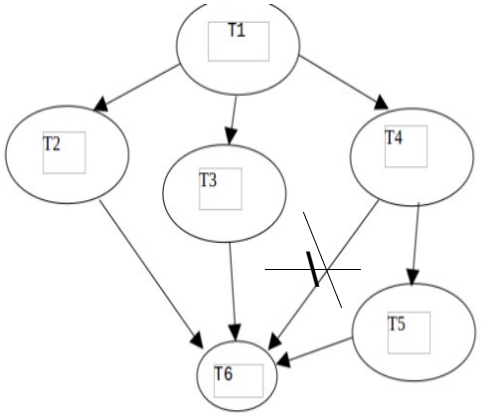
Implementazione di grafi delle precedenze

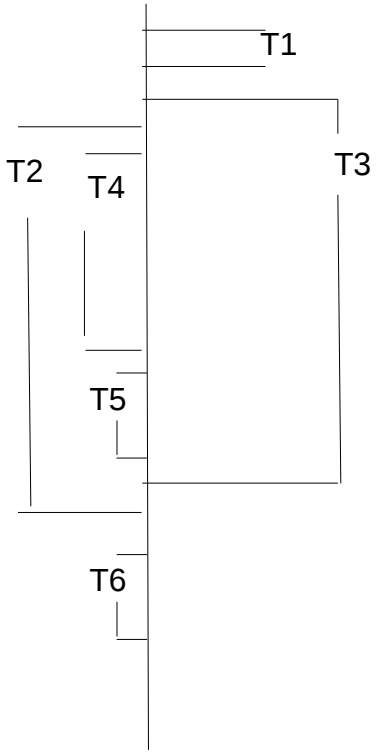





```
#include <sys/types.h> //pid_t
#include <unistd.h>    //fork()
#include <sys/wait.h>  //waitpid
#include <stdio.h>     //printf...
#include <stdlib.h>    //exit()
//gp-4.c
int main(int argc, char *argv[]){
     pid_t cpid, w, pid1,pid2,pid3,pid4,pid5,pid6,pid7;

     if (pid1=fork() == 0) execlp("/home/mumolo/proc","proc","T1",NULL);
     waitpid(pid1, NULL, 0);

     if (pid3=fork() == 0) execlp("/home/mumolo/proc","proc","T3",NULL);
     if (pid2=fork() == 0) execlp("/home/mumolo/proc","proc","T2",NULL);

     waitpid(pid2, NULL, 0);

     if (pid4=fork() == 0) execlp("/home/mumolo/proc","proc","T4",NULL);
     if (pid5=fork() == 0) execlp("/home/mumolo/proc","proc","T5",NULL);

     waitpid(pid3, NULL, 0);

     if (pid6=fork() == 0) execlp("/home/mumolo/proc","proc","T6",NULL);

     waitpid(pid5, NULL, 0);
     waitpid(pid4, NULL, 0);
     waitpid(pid6, NULL, 0);

     if (pid7=fork() == 0) execlp("/home/mumolo/proc","proc","T7",NULL);
     waitpid(pid7, NULL, 0);
}
```

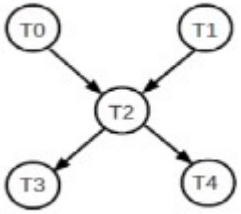
Implementazione di grafi delle precedenze

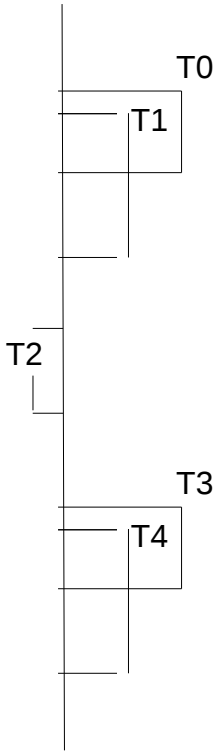





```
#include <sys/types.h> //pid_t
#include <unistd.h>    //fork()
#include <sys/wait.h>  //waitpid
#include <stdio.h>     //printf...
#include <stdlib.h>    //exit()
//gp-5.c
int main(int argc, char *argv[]){
     pid_t cpid, w, pid1,pid2,pid3,pid4,pid5,pid6,pid7;

     if (pid1=fork() == 0) execlp("/home/mumolo/proc","proc","T1",NULL);
     waitpid(pid1, NULL, 0);

     if (pid3=fork() == 0) execlp("/home/mumolo/proc","proc","T3",NULL);
     if (pid2=fork() == 0) execlp("/home/mumolo/proc","proc","T2",NULL);
     if (pid4=fork() == 0) execlp("/home/mumolo/proc","proc","T4",NULL);

     waitpid(pid4, NULL, 0);

     if (pid5=fork() == 0) execlp("/home/mumolo/proc","proc","T5",NULL);

     waitpid(pid5, NULL, 0);
     waitpid(pid3, NULL, 0);
     waitpid(pid2, NULL, 0);

     if (pid6=fork() == 0) execlp("/home/mumolo/proc","proc","T6",NULL);
     waitpid(pid6, NULL, 0);
}
```

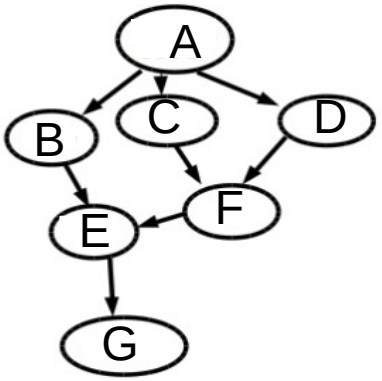
Implementazione di grafi delle precedenze

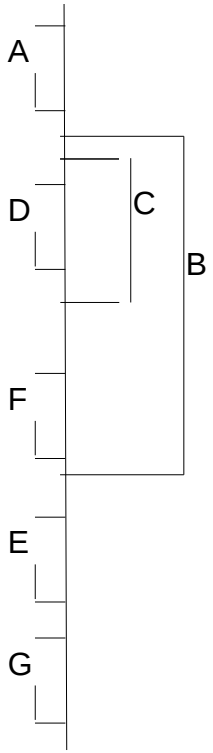





```
#include <sys/types.h> //pid_t
#include <unistd.h>    //fork()
#include <sys/wait.h>  //waitpid
#include <stdio.h>     //printf...
#include <stdlib.h>    //exit()
//gp-6.c
int main(int argc, char *argv[]){
     pid_t cpid, w, pid0, pid1,pid2,pid3,pid4,pid5,pid6,pid7;

     if (pid0=fork() == 0) execlp("/home/mumolo/proc","proc","T0",NULL);
     if (pid1=fork() == 0) execlp("/home/mumolo/proc","proc","T1",NULL);

     waitpid(pid0, NULL, 0);
     waitpid(pid1, NULL, 0);

     if (pid2=fork() == 0) execlp("/home/mumolo/proc","proc","T2",NULL);
     waitpid(pid2, NULL, 0);

     if (pid3=fork() == 0) execlp("/home/mumolo/proc","proc","T3",NULL);
     if (pid4=fork() == 0) execlp("/home/mumolo/proc","proc","T4",NULL);

     waitpid(pid3, NULL, 0);
     waitpid(pid4, NULL, 0);
}
```

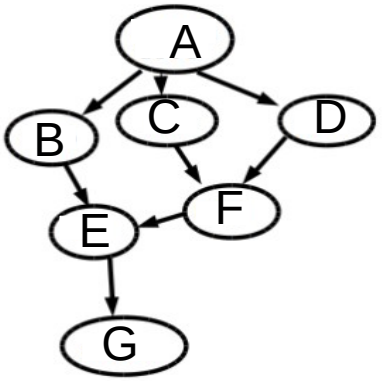
Implementazione di grafi delle precedenze

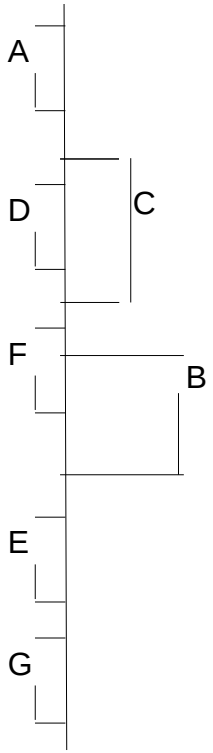





```
#include <sys/types.h> //pid_t
#include <unistd.h>    //fork()
#include <sys/wait.h>  //waitpid
#include <stdio.h>     //printf...
#include <stdlib.h>    //exit()
//gp-7.c
int main(int argc, char *argv[]){
     pid_t cpid, w, pid0, pid1,pid2,pid3,pid4,pid5,pid6,pid7;

     if (pid0=fork() == 0) execlp("/home/mumolo/proc","proc","A",NULL);
     waitpid(pid0, NULL, 0);

     if (pid1=fork() == 0) execlp("/home/mumolo/proc","proc","B",NULL);
     if (pid2=fork() == 0) execlp("/home/mumolo/proc","proc","C",NULL);
     if (pid3=fork() == 0) execlp("/home/mumolo/proc","proc","D",NULL);

     waitpid(pid2, NULL, 0);
     waitpid(pid3, NULL, 0);

     if (pid4=fork() == 0) execlp("/home/mumolo/proc","proc","F",NULL);

     waitpid(pid4, NULL, 0);
     waitpid(pid1, NULL, 0);

     if (pid5=fork() == 0) execlp("/home/mumolo/proc","proc","E",NULL);
     waitpid(pid5, NULL, 0);

     if (pid6=fork() == 0) execlp("/home/mumolo/proc","proc","G",NULL);
     waitpid(pid6, NULL, 0);

}
```

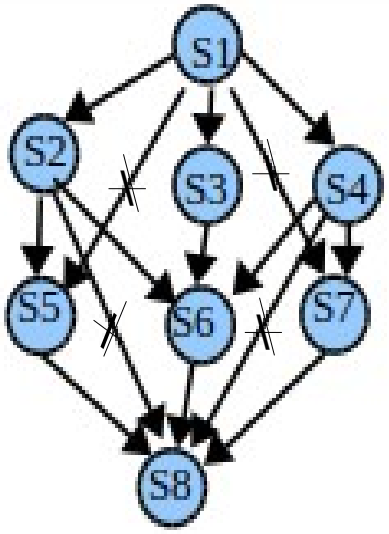
Implementazione di grafi delle precedenze

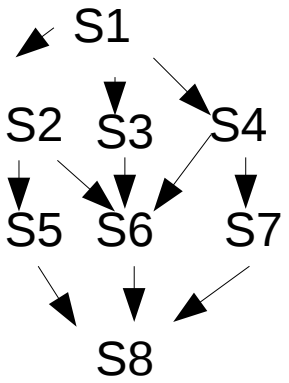




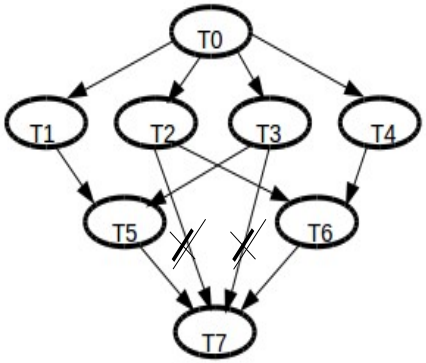

```
#include <sys/types.h> //pid_t
#include <unistd.h>    //fork()
#include <sys/wait.h>  //waitpid
#include <stdio.h>     //printf...
#include <stdlib.h>    //exit()
//gp-8.c
int main(int argc, char *argv[]){
     pid_t cpid, w, pid0, pid1,pid2,pid3,pid4,pid5,pid6,pid7;

     if (pid0=fork() == 0) execlp("/home/mumolo/proc","proc","A",NULL);
     waitpid(pid0, NULL, 0);

     if (pid2=fork() == 0) execlp("/home/mumolo/proc","proc","C",NULL);
     if (pid3=fork() == 0) execlp("/home/mumolo/proc","proc","D",NULL);

     waitpid(pid3, NULL, 0);
     waitpid(pid2, NULL, 0);

     if (pid4=fork() == 0) execlp("/home/mumolo/proc","proc","F",NULL);
     if (pid1=fork() == 0) execlp("/home/mumolo/proc","proc","B",NULL);

     waitpid(pid4, NULL, 0);
     waitpid(pid1, NULL, 0);

     if (pid5=fork() == 0) execlp("/home/mumolo/proc","proc","E",NULL);
     waitpid(pid5, NULL, 0);

     if (pid6=fork() == 0) execlp("/home/mumolo/proc","proc","G",NULL);
     waitpid(pid6, NULL, 0);

}
```

Implementazione di grafi delle precedenze





Implementazione di grafi delle precedenze



Facciamo il punto

- Abbiamo visto lo sviluppo di applicazioni in ambiente Linux (System call, funzioni, file, processi, IPC...)
- Programmazione concorrente
  - Processi concorrenti : abbastanza pesante
  - Thread concorrenti : molto più leggeri
- Perchè programmazione concorrente in Linux:
  - Una linea di comando Linux viene realizzata con processi concorrenti !
  - Applicazioni concorrenti e multithread
    - Gestione concorrente di più sensori
    - Ci sono Algoritmi intrinsicamente concorrenti, altri Sequenziali che possono essere resi più efficienti con la concorrenza
    - Hw concorrente (CPU multi core, GPU...)