



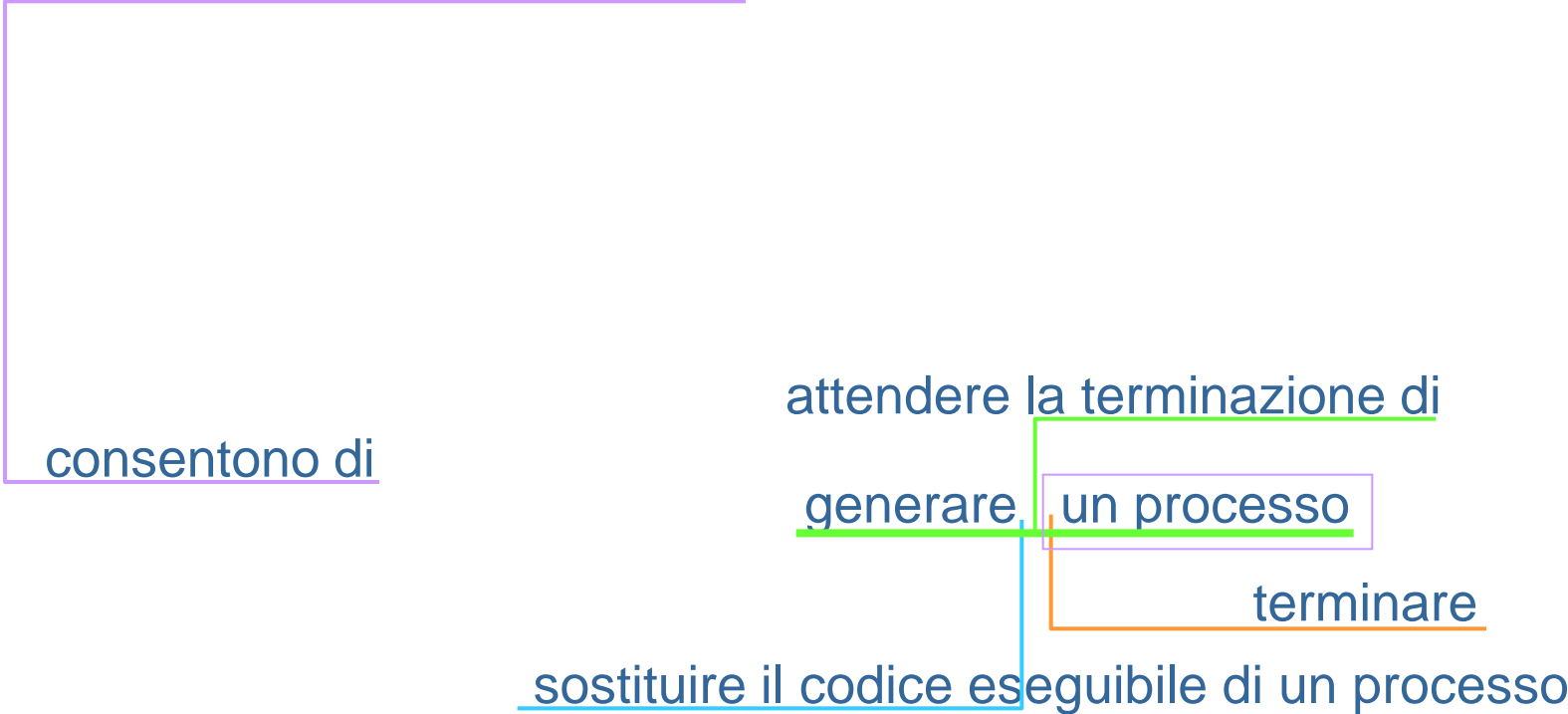
POLITECNICO
DI MILANO

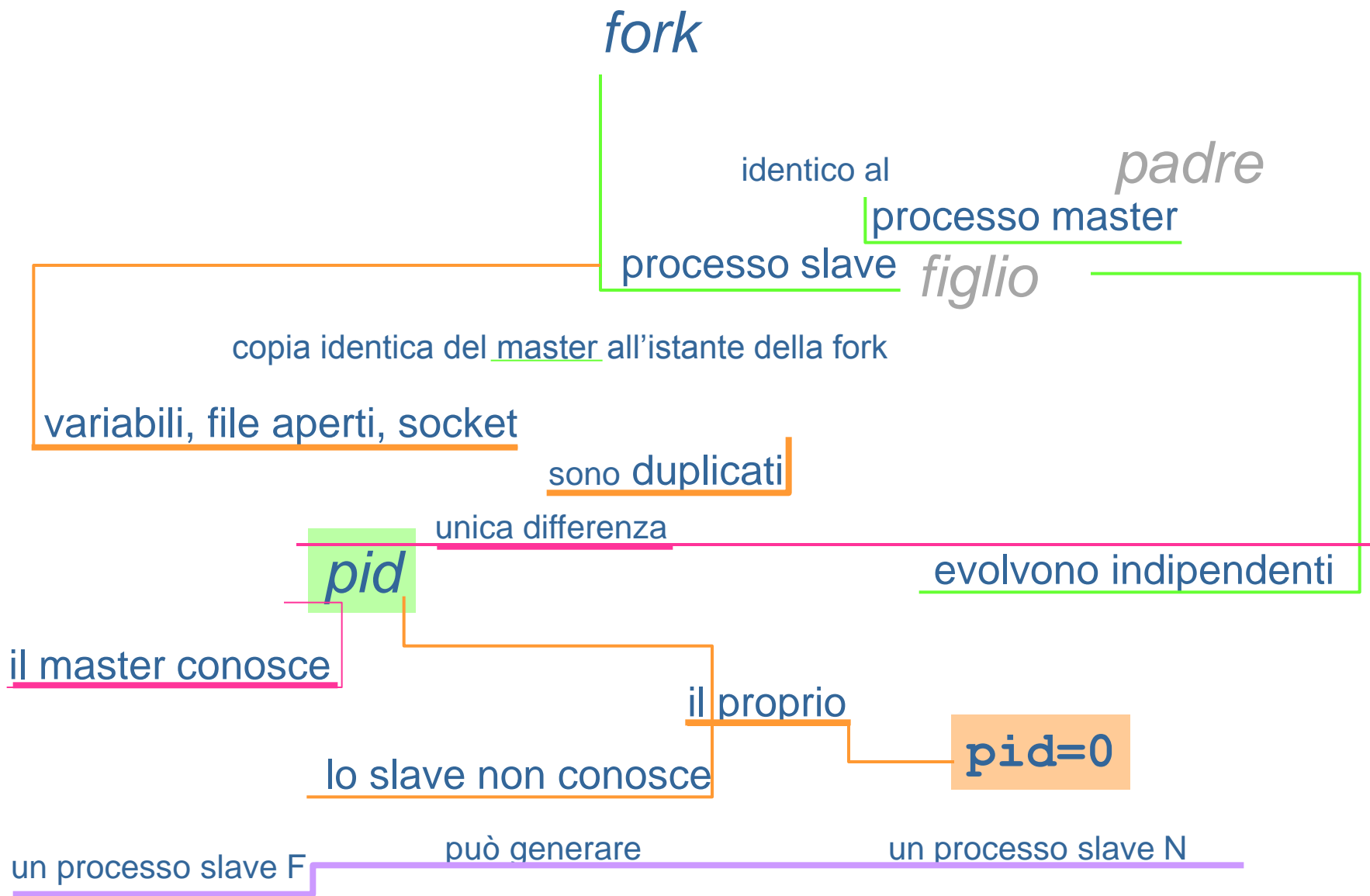
INFORMATICA

La gestione dei processi

Operazioni sui processi

Operazioni sui processi





Gerarchie di processi

Master P

Slave F1

Slave F2

Slave N11

Slave N12

Slave N2

F1 è master di N11 e N12

P è master di F1 e F2

F2 è master di N2

Sintassi della *fork*

qualsiasi altro valore nel master

indica il pid dello slave

```
pid = fork(); /* restituisce pid_t */
```

assume valore 0 nello slave

assume valore -1 nel master

se la fork non è stata eseguita

Utilizzo della *fork*

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char * argv[])
{  pid_t pid;
   int retstatus;

   pid = fork();
   if (pid==0) {
       /* sono nello slave */
       ...
       /* lo slave termina restituendo al padre lo */
       /* stato di terminazione */
       exit(retstatus);
   }
   else {
       /* pid != 0, sono nel master */
   }
}
```

master

processID=a

```
s_pid = fork();
if (s_pid == 0) {
    compito dello slave
}
else {w_pid = wait(&stato);}
seguito del master;
```

master

processID=a

s_pid=b

```
s_pid = fork();
if (s_pid == 0) {...}
else {w_pid = wait(&stato);
il master attende la terminazione
dello slave
};
```

dopo la terminazione del processo

```
s_pid = fork();
if (s_pid == 0) {...}
else {w_pid = wait(&stato);
dopo la terminazione dello slave
il master procede l'esecuzione
};
```

master

slave

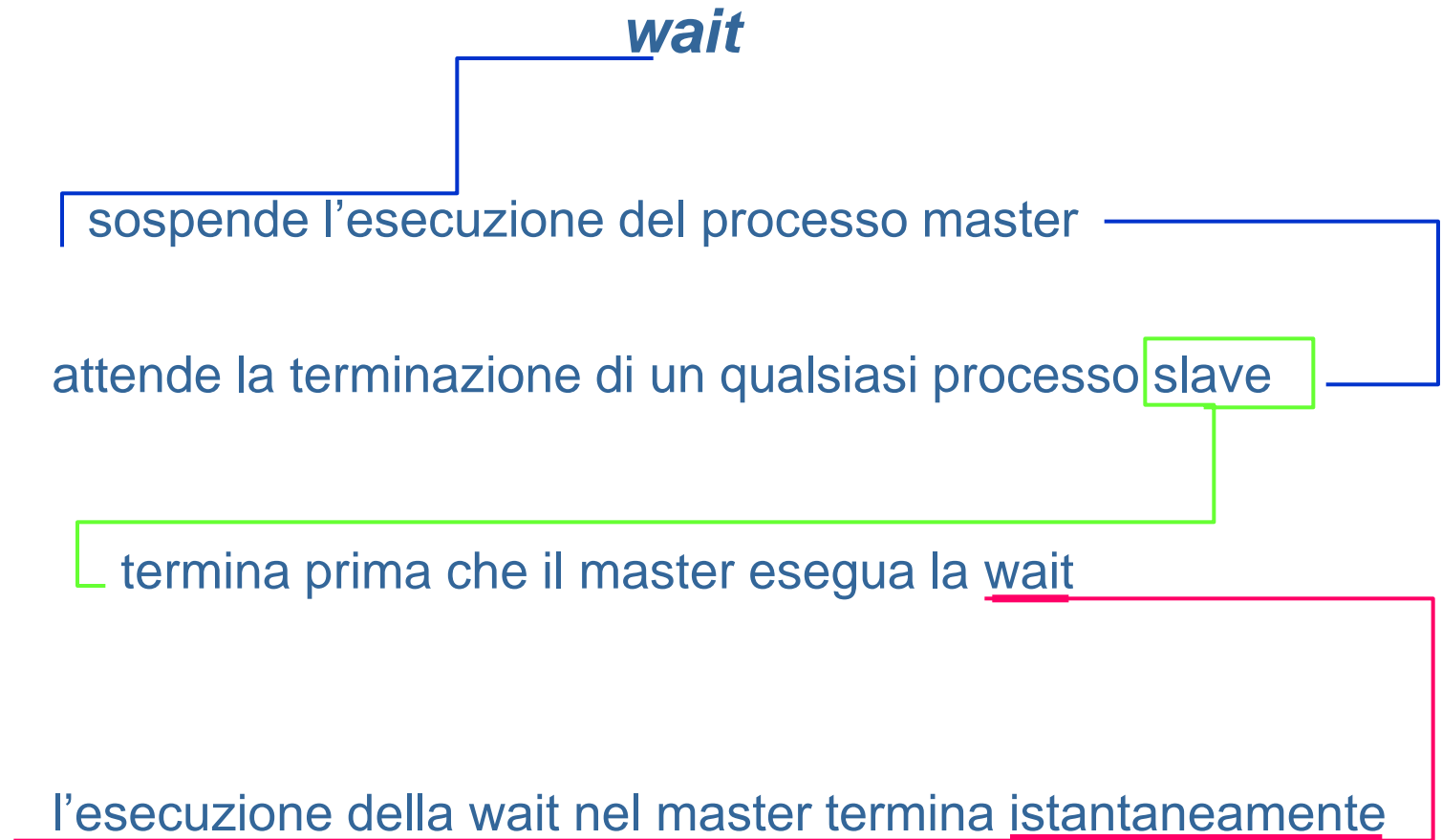
processID=b

s_pid=0

```
s_pid = fork();
if (s_pid == 0) {
    compito dello slave
    exit(0);
}
else {};
```

termine del processo

wait



Sintassi della *wait*

```
my_pid = wait(int *status);
```

```
/* restituisce un tipo pid_t */
```

nel master

assume il valore del pid dello slave terminato

assume il valore del codice di terminazione
del processo slave

Utilizzo della *wait*

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(int argc, char * argv[])
{
    pid_t pid, my_pid;
    int status;

    pid = fork();
    if (pid==0) {}
    else {
        printf("Ho generato il processo slave con pid %d\n",pid)
            /* pid != 0, sono nel master */
        ...
        my_pid = wait(&status);
        printf("E' terminato il processo %d con esito %d\n",my_pid, status);
    }
}
```

waitpid

evoluzione funzione wait

sospende l'esecuzione del processo master

attende la terminazione del processo slave

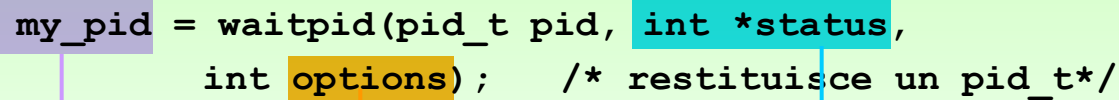
`pid (my_pid)`

termina prima che il master esegua la waitpid

l'esecuzione della waitpid nel master termina istantaneamente

Sintassi della *waitpid*

```
my_pid = waitpid(pid_t pid, int *status,  
                 int options); /* restituisce un pid_t*/
```



assume il valore del pid dello slave terminato

assume il valore del codice di terminazione del processo slave

specifica ulteriori opzioni (ipotizziamo >0)

Utilizzo della *waitpid*

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(int argc, char * argv[])
{   pid_t pid, my_pid;
    int status;

    pid = fork();
    if (pid==0) {}
    else {
        printf("Ho generato il processo slave con pid %d\n",pid)
            /* pid != 0, sono nel master */
        ...
        printf("Attendo la terminazione dello slave con pid %d\n",pid)
        my_pid_t = waitpid(pid, &status, 1);
        printf("E' terminato il processo %d con esito %d\n",my_pid, status);
    }
}
```

exec

sostituisce il codice eseguibile del **processo** corrente

con il codice eseguibile del programma specificato

- mantiene lo stesso **pid**

- passa i parametri al processo con il nuovo codice

attraverso la linea di comando (**argc** e **argv**)

Sintassi della exec

```
status = exec(char *nome_programma,  
               char **argv);  
/* restituisce int */
```

0 se l'operazione è stata eseguita correttamente

-1 se c'è stato un errore e l'operazione di sostituzione
del codice è fallita

Utilizzo della *exec*

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char * argv[])
{  pid_t pid;
   int retstatus;

   pid = fork();
   if (pid==0) { /* sono nello slave */
       exec("ls", {"ls", "-lag", "/home", (char *) NULL});
       retstatus = 0;
       exit(retstatus);
   }
   else {
       /* pid != 0, sono nel master */
   }
}
```

master

```
if (s_pid == 0) {  
    compito dello slave;  
    exec("my_prog",...)  
} else {w_pid = wait(&stato);}  
seguito del master;
```

master

processoID=a

```
if (s_pid == 0) {...}  
    else {w_pid = wait(&stato);  
    il master attende la terminazione  
    dello slave  
};
```

Dopo la terminazione di my_prog

master

```
if (s_pid == 0) {...}  
    else {w_pid = wait(&stato);  
    dopo la terminazione dello slave  
    il master procede l'esecuzione  
};
```

slave

```
if (s_pid == 0) {  
    compito dello slave  
    exec("my_prog",...);  
} else {};
```

slave

(con exec sostituisco il codice)

```
main(int argc, char **argv)  
{  
    exit(0);  
}
```

termine del processo slave

exit

pone termine al processo corrente (slave)

restituendo un codice al processo master

non ha un processo master

il codice della exit viene restituito
all'interprete comandi

Sintassi della *exit*

```
void exit(int status);  
    /* non restituisce nulla al */  
    /* processo che esegue la exit*/
```

Utilizzo della *exit*

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char * argv[])
{
    pid_t pid;
    int retstatus;

    pid = fork();
    if (pid==0) {
        /* lo slave restituisce al master lo stato di terminazione */
        retstatus = 1024;
        exit(retstatus);
    }
    else {
        /* pid != 0, sono nel master */
        pid = wait(&status);
        printf("E' terminato il processo %d con esito %d\n",pid,status/256);
        /* stampera' il numero del pid dello slave e il numero 1024 */
    }
}
```