Esercizio sulla gestione di processi in Unix

Esercizio

Si vuole realizzare un programma C che abbia la seguente interfaccia:

```
./launch_commands
```

Il programma deve richiedere all'utente:

- un numero **N** (pari al numero di comandi che vorrà lanciare)
- uno alla volta i comandi **<com1 … comN>**. Per semplicità consideriamo solo comandi senza argomenti e senza opzioni (es: **ls, date, whoami,** ecc…)

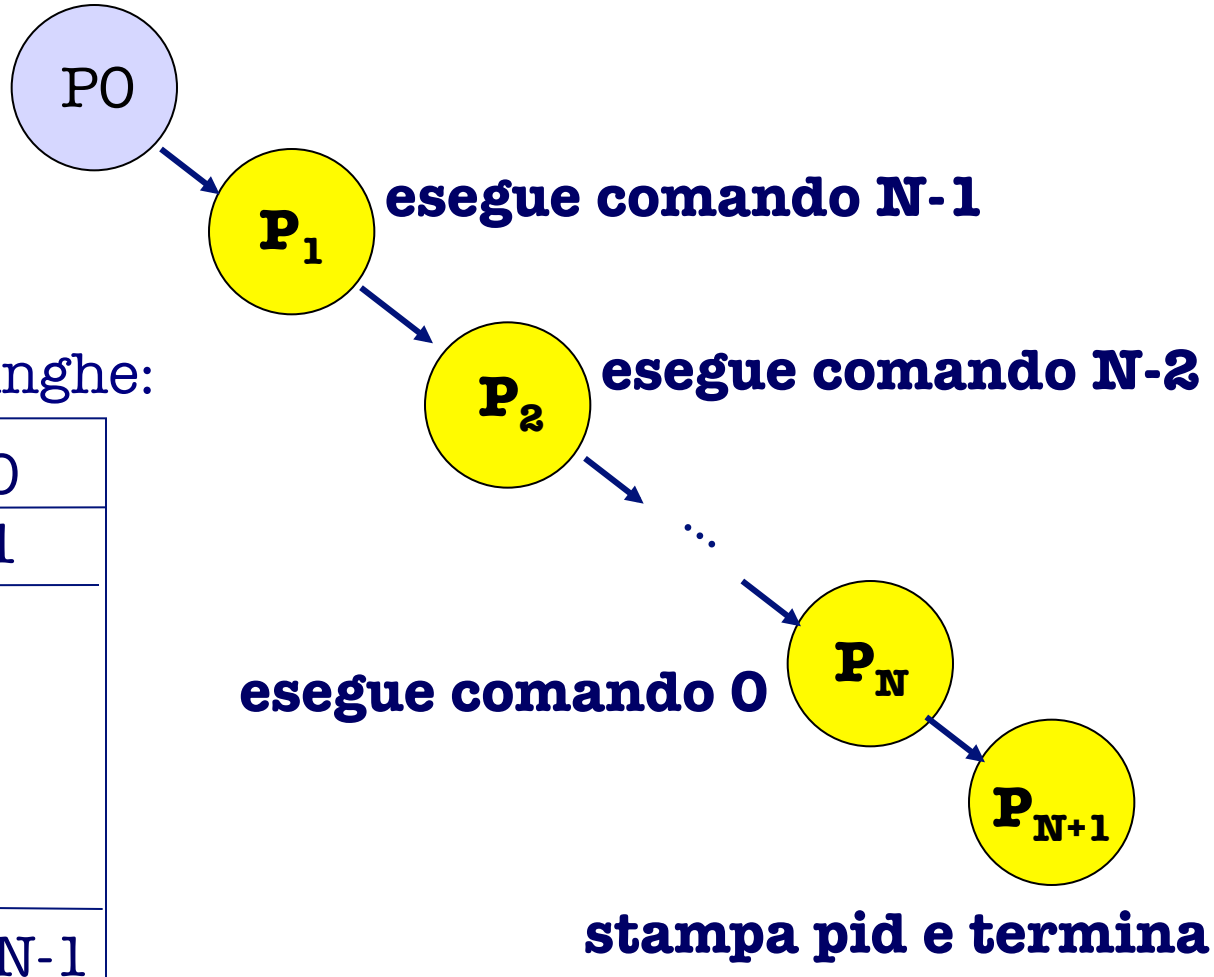
Esercizio

Il programma C deve rispettare le seguenti specifiche:

- il **processo iniziale** (**P_0**) deve acquisire da standard input N e la sequenza di N comandi.
- Una volta letti gli N comandi, il processo iniziale P_0 deve dare origine ad una gerarchia lineare di processi di **profondità N+1** (figlio **P_1** - nipote **P_2** - bisnipote **P_3** -... - **P_{N+1}**) : ogni processo della gerarchia dovrà eseguire un diverso comando della sequenza data;
- l'ultimo processo(**P_{N+1}**) della gerarchia, invece, dovrà semplicemente stampare il suo pid e terminare.

Creiamoci un vettore di stringhe:

0	Comando 0
1	Comando 1
N-1	Comando N-1



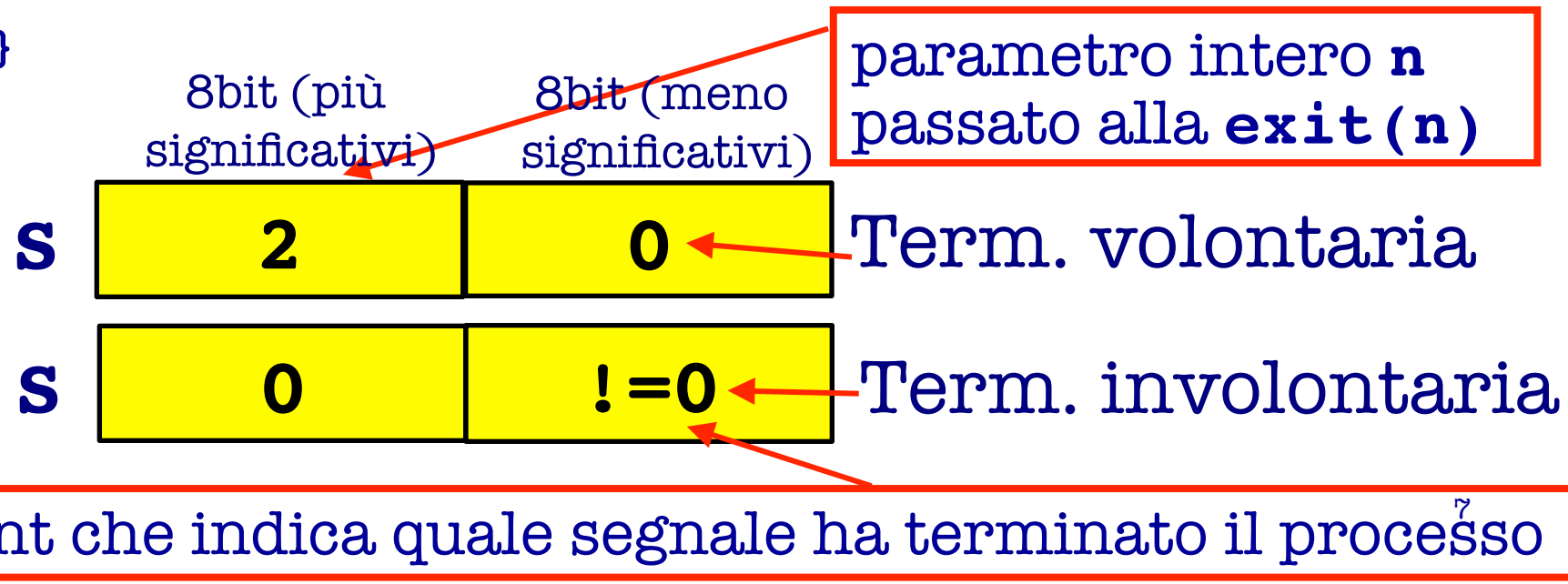
```
#include <stdio.h>
#define DIM 20
typedef char stringa[80];
typedef stringa strvett[DIM];
void gestoresequenza(int N, strvett vett);
int gest_stato(int S, int pid);

main()
{ int pid, ncom, stato, i;
  strvett vstr;
  printf("quanti comandi? ");
  scanf("%d", &ncom);
  for(i=0; i<ncom; i++)
  {     printf("\ndammi il prossimo comando(senza argomenti)");
        scanf("%s", vstr[i]);
  }
    gestoresequenza(ncom-1,vstr);
    pid=wait(&stato);
    gest_stato(stato,pid);
}
```

```
void gestoresequenza(int N, strvett vett)
{  int pid;
   pid=fork();
   if (pid==0) /* figlio*/
   {    if (N==-1) /* processo foglia */
        {    printf("\nfoglia %d: \n", getpid());
             exit(0);
        }
        else /*attivazione di un nuovo comando*/
        {   printf("\nProcesso %d per comando %s",getpid(),
                        vett[N]);
            gestoresequenza(N-1, vett);
            execlp(vett[N], vett[N], (char *)0);
            perror("\nexec fallita: ");
            exit(-1);
        }
   }
}
```

Potrei fare prima exec e poi gestoresequenza?

No, perché dalla exec non c'è ritorno!

```
int gest_stato(int S, int pid)
{
      printf("terminato processo figlio n.%d", pid);
      if ((char)S==0)
        printf("term. volontaria con stato %d", S>>8);
      else
        printf("terminazione involontaria  per segnale
                    %d\n", (char)S);
}
```



Simulazione di Esecuzione (1/7)

- Vediamo cosa succede durante l'esecuzione del programma

- Supponiamo di inserire:
  - **quanti comandi? 2**
  - **comando1 = ps** (lista i processi)
  - **comando2 = who** (lista utenti connessi)

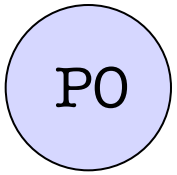
Processo P0

```
#include <stdio.h>
#define DIM 20
typedef char stringa[80];
typedef stringa strvett[DIM];
void gestoresequenza(int N, strvett vett);
int gest_stato(int S, int pid);

main()
{ int pid, ncom, stato, i;
  strvett vstr;
  printf("quanti comandi? ");
  scanf("%d", &ncom);
  for(i=0; i<ncom; i++)
  {     printf("\ndammi il prossimo comando(senza argomenti)");
        scanf("%s", vstr[i]);
  }
  gestoresequenza(ncom-1,vstr);
    pid=wait(&stato);
    gest_stato(stato,pid);
}
```

ncom=2

vstr[0]=ps
vstr[1]=who

N=1



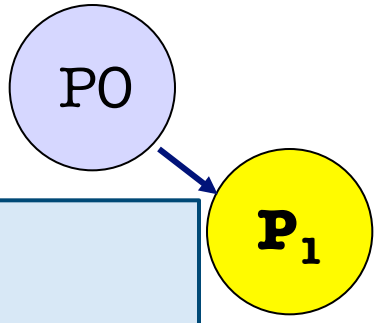

```
main:
        ncom=2
        vstr[0]=ps
        vstr[1]=who
gestoresequenza(1):
        N=1
        vett[0]=ps
        vett[1]=who
```

Processo P0

```
void gestoresequenza(int N, strvett vett)
{  int pid;
   pid=fork();
   if (pid==0) /* figlio*/
   {    if (N==-1) /* processo foglia */
        {    printf("\nfoglia %d: \n", getpid());
             exit(0);
        }
        else /*attivazione di un nuovo comando*/
        {   printf("\nProcesso %d per comando %s",getpid(),
                        vett[N]);
            gestoresequenza(N-1, vett);
            execlp(vett[N], vett[N], (char *)0);
            perror("\nexec fallita: ");
            exit(-1);
        }
   }
}
```

gestoresequenza(1):
N=1
vett[0]=ps
vett[1]=who

P0 genera un figlio P1




```
main:
        ncom=2
        vstr[0]=ps
        vstr[1]=who
gestoresequenza(1):
        N=1
        vett[0]=ps
        vett[1]=who
```

```
main:
        ncom=2
        vstr[0]=ps
        vstr[1]=who
gestoresequenza(1):
        N=1
        vett[0]=ps
        vett[1]=who
```

P1 riceve una **copia** del contesto di P0.

Continuiamo a concentrarci su P0

Processo P0

```
#include <stdio.h>
#define DIM 20
typedef char stringa[80];
typedef stringa strvett[DIM];
void gestoresequenza(int N, strvett vett);
int gest_stato(int S, int pid);

main()
{ int pid, ncom, stato, i;
  strvett vstr;
  printf("quanti comandi? ");
  scanf("%d", &ncom);
  for(i=0; i<ncom; i++)
  {     printf("\ndammi il prossimo comando(senza argomenti)");
        scanf("%s", vstr[i]);
  }
    gestoresequenza(ncom-1,vstr);
    pid=wait(&stato);
    gest_stato(stato,pid);
}
```

P0 si sospende nell'attesa che P1 termini
Quando ciò accade, esegue **gest_stato**

Processo P0

```
int gest_stato(int S, int pid)
{
      printf("terminato processo figlio n.%d", pid);
      if ((char)S==0)
        printf("term. volontaria con stato %d", S>>8);
      else
        printf("terminazione involontaria  per segnale
                    %d\n", (char)S);
}
```

Supponiamo che P1 termini volontariamente

=> P0 stampa il messaggio e termina.

Vediamo cosa fa **concorrentemente** P1...

Processo P1

```
void gestoresequenza(int N, strvett vett)
{  int pid;
   pid=fork();
   if (pid==0) /* figlio*/
   {    if (N==-1) /* processo foglia */
        {    printf("\nfoglia %d: \n", getpid());
             exit(0);
        }
        else /*attivazione di un nuovo comando*/
        {    printf("\nProcesso %d per comando %s",getpid(),
                        vett[N]);
             gestoresequenza(N-1, vett);
             execlp(vett[N], vett[N], (char *)0);
             perror("\nexec fallita: ");
             exit(-1);
        }
   }
}
```

gestoresequenza(1):
N=1
vett[0]=ps
vett[1]=who

who

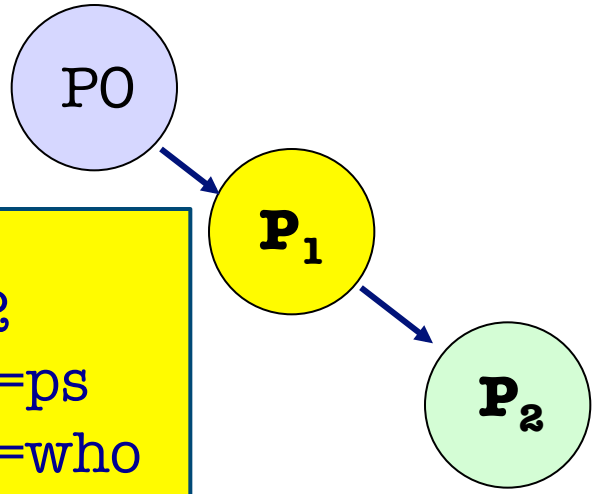
P1 chiama nuovamente la funzione **gestoresequenza(0,vett)**

Processo P1

```
void gestoresequenza(int N, strvett vett)
{  int pid;
   pid=fork();
   if (pid==0) /* figlio*/
   {    if (N==-1) /* processo foglia */
        {    printf("\nfoglia %d: \n", getpid());
             exit(0);
        }
        else /*attivazione di un nuovo comando*/
        {   printf("\nProcesso %d per comando %s",getpid(),
                        vett[N]);
            gestoresequenza(N-1, vett);
            execlp(vett[N], vett[N], (char *)0);
            perror("\nexec fallita: ");
            exit(-1);
        }
   }
}
```

gestoresequenza(2):
N=0
vett[0]=ps
vett[1]=who

P1 genera un figlio P2




```
main:
        ncom=2
        vstr[0]=ps
        vstr[1]=who
gestoresequenza:(1)
        N=1
        vett[0]=ps
        vett[1]=who
gestoresequenza:(2)
        N=0
        vett[0]=ps
        vett[1]=who
```

```
main:
        ncom=2
        vstr[0]=ps
        vstr[1]=who
gestoresequenza:(1)
        N=1
        vett[0]=ps
        vett[1]=who
gestoresequenza:(2)
        N=0
        vett[0]=ps
        vett[1]=who
```

P2 riceve una **copia** del contesto di P1. Continuiamo a concentrarci su P1…

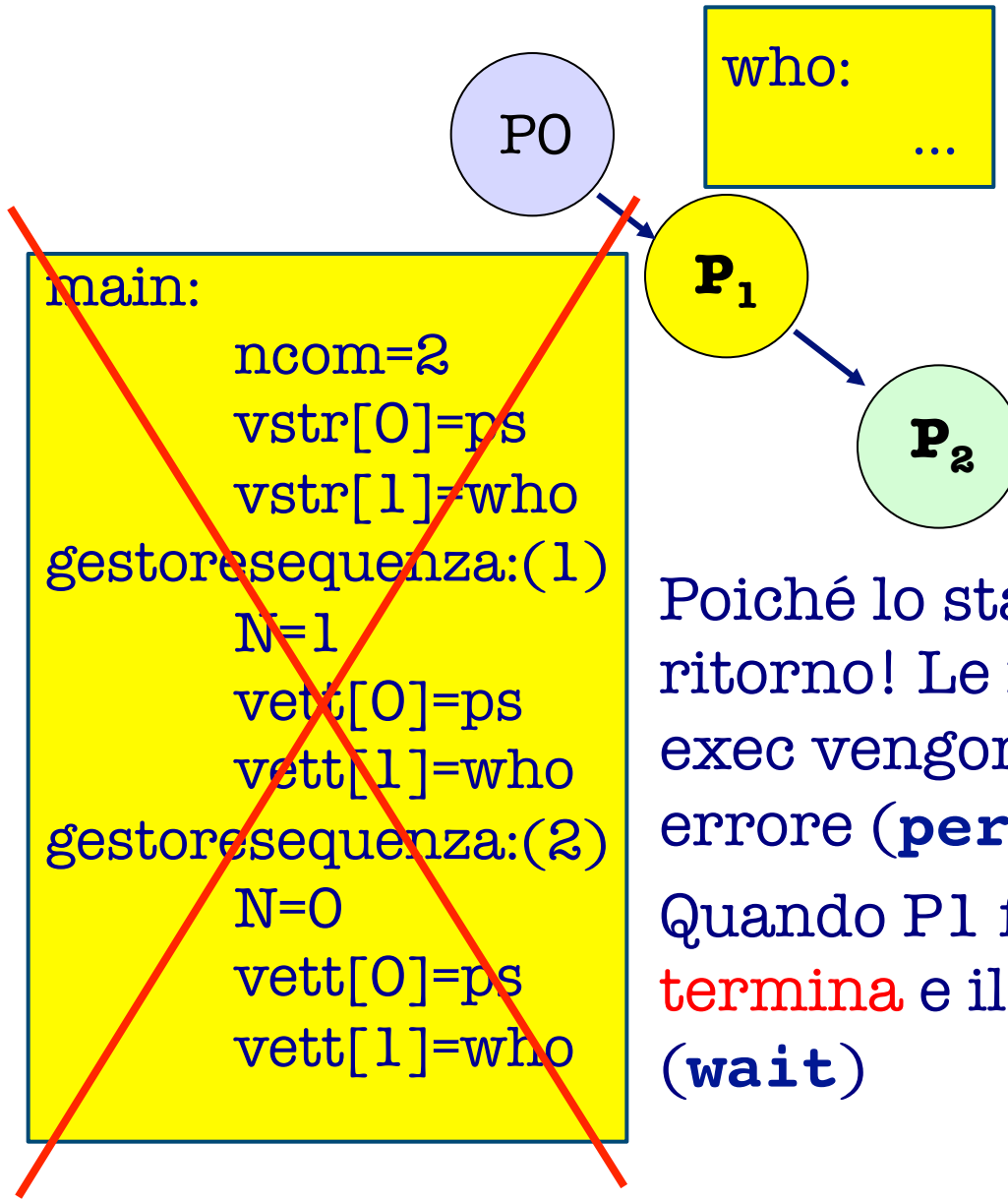
Processo P1

```
void gestoresequenza(int N, strvett vett)
{  int pid;
   pid=fork();
   if (pid==0) /* figlio*/
   {    if (N==-1) /* processo foglia */
        {    printf("\nfoglia %d: \n", getpid());
             exit(0);
        }
        else /*attivazione di un nuovo comando*/
        {   printf("\nProcesso %d per comando %s",getpid(),
                        vett[N]);
            gestoresequenza(N-1, vett);
            execlp(vett[N], vett[N], (char *)0);
            perror("\nexec fallita: ");
            exit(-1);
        }
   }
}
```

gestoresequenza(1):
N=1
vett[0]=ps
vett[1]=who

who

P1 ritorna alla 1° invocazione di **gestoresequenza**



la **exec sostituisce** il codice (scritto da noi) ed il contesto di P1 con il codice ed il contesto del comando **who**.

Poiché lo stack è sostituito, non c'è ritorno! Le istruzioni che seguono la exec vengono eseguite solo in caso si errore (**perror**)

Quando P1 finisce di eseguire **who**, termina e il suo stato è raccolto da P0 (**wait**)

Processo P2

gestoresequenza(2):
N=0
vett[0]=ps
vett[1]=who

```
void gestoresequenza(int N, strvett vett)
{  int pid;
   pid=fork();
   if (pid==0) /* figlio*/
   {    if (N==-1) /* processo foglia */
        {    printf("\nfoglia %d: \n", getpid());
             exit(0);
        }
        else /*attivazione di un nuovo comando*/
        {    printf("\nProcesso %d per comando %s",getpid(),
                        vett[N]);
             gestoresequenza(N-1, vett);
             execlp(vett[N], vett[N], (char *)0);
             perror("\nexec fallita: ");
             exit(-1);
        }
   }
}
```

ps

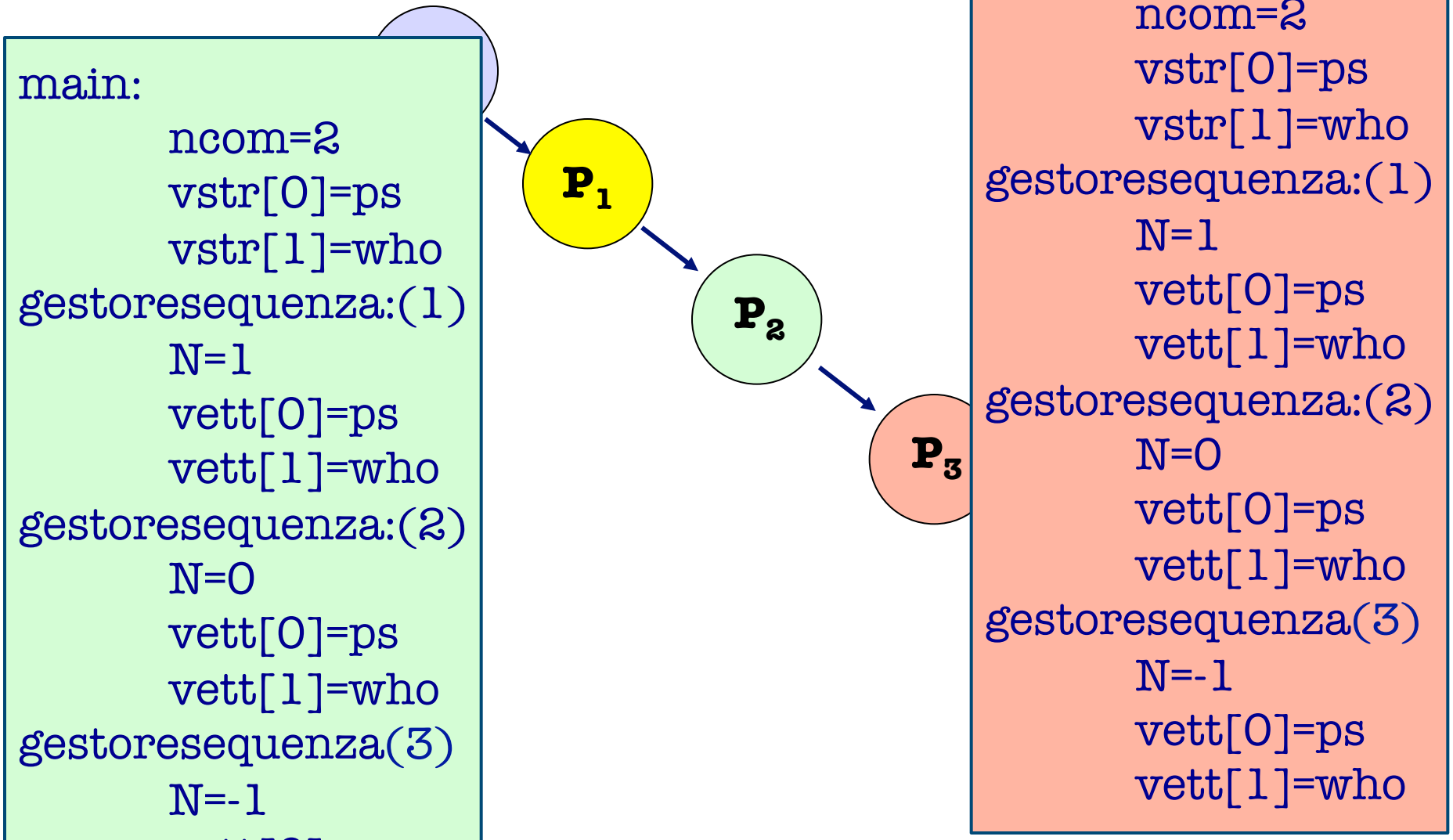
P2 chiama nuovamente (3°) la funzione **gestoresequenza(-1,vett)**

Processo P2

```
void gestoresequenza(int N, strvett vett)
{  int pid;
   pid=fork();
   if (pid==0) /* figlio*/
   {    if (N==-1) /* processo foglia */
        {    printf("\nfoglia %d: \n", getpid());
             exit(0);
        }
        else /*attivazione di un nuovo comando*/
        {   printf("\nProcesso %d per comando %s",getpid(),
                        vett[N]);
            gestoresequenza(N-1, vett);
            execlp(vett[N], vett[N], (char *)0);
            perror("\nexec fallita: ");
            exit(-1);
        }
   }
}
```

gestoresequenza(3):
N=-1
vett[0]=ps
vett[1]=who

P2 genera l'ultimo figlio P3 e ritorna.



P3 riceve una **copia** del contesto di P2. Continuiamo con P2...

Processo P2

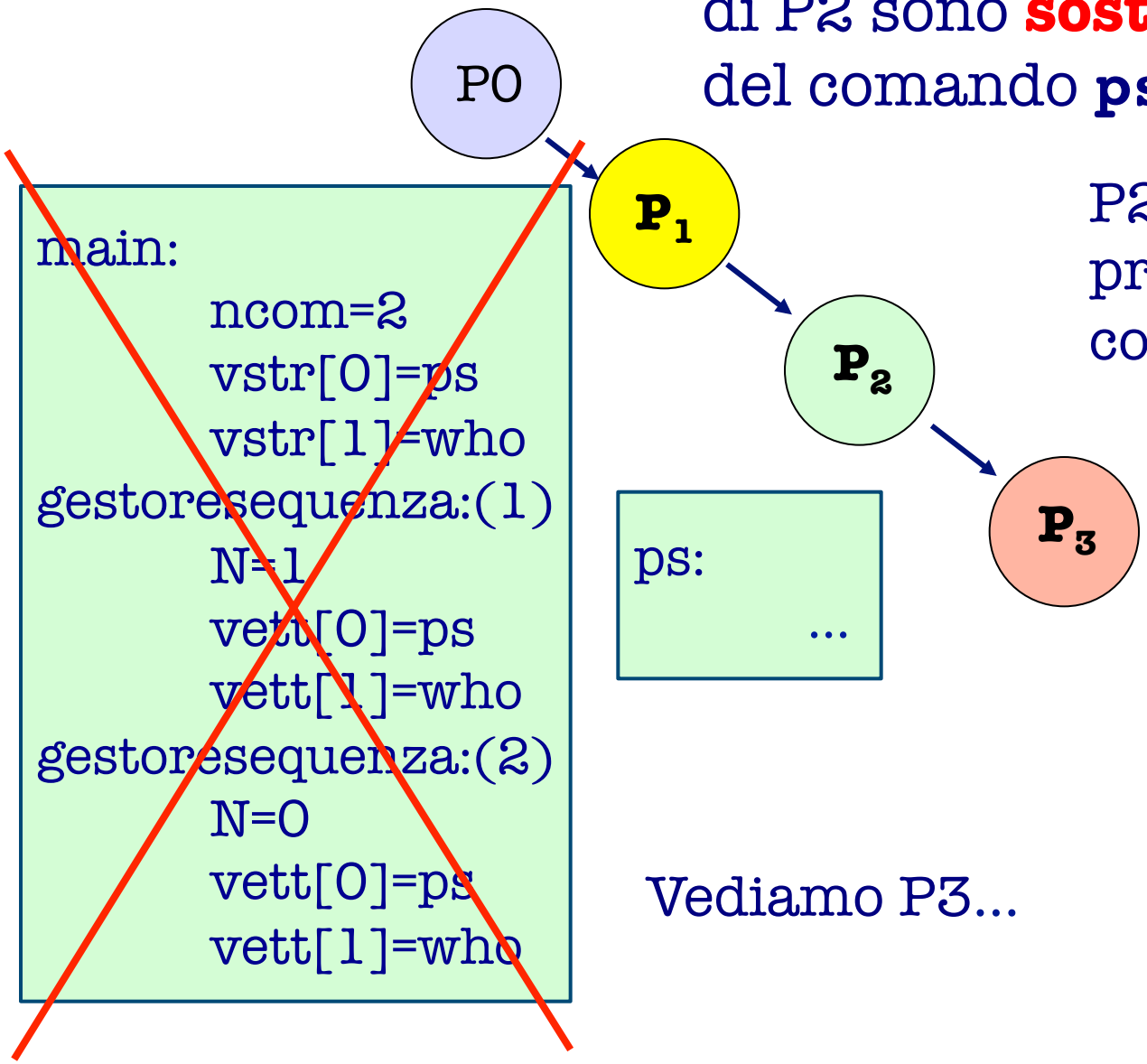
```
void gestoresequenza(int N, strvett vett)
{  int pid;
   pid=fork();
   if (pid==0) /* figlio*/
   {    if (N==-1) /* processo foglia */
        {    printf("\nfoglia %d: \n", getpid());
             exit(0);
        }
        else /*attivazione di un nuovo comando*/
        {   printf("\nProcesso %d per comando %s",getpid(),
                        vett[N]);
            gestoresequenza(N-1, vett);
            execlp(vett[N], vett[N], (char *)0);
            perror("\nexec fallita: ");
            exit(-1);
        }
   }
}
```

gestoresequenza(2):
N=0
vett[0]=ps
vett[1]=who

ps

P2 ritorna dalla 3° invocazione di gestoresequenza ed esegue il comando **ps**

grazie alla **exec** contesto e codice di P2 sono **sostituiti** con quelli del comando **ps.**



P2 termina come prescritto dal codice di ps

Vediamo P3...

Processo P3

```
void gestoresequenza(int N, strvett vett)
{  int pid;
   pid=fork();
if (pid==0) /* figlio*/
{    if (N==-1) /* processo foglia */
     {    printf("\nfoglia %d: \n", getpid());
          exit(0);
     }
     else /*attivazione di un nuovo comando*/
     {   printf("\nProcesso %d per comando %s",getpid(),
                    vett[N]);
         gestoresequenza(N-1, vett);
         execlp(vett[N], vett[N], (char *)0);
         perror("\nexec fallita: ");
         exit(-1);
     }
   }
}
```

gestoresequenza(3):
N=-1
vett[0]=ps
vett[1]=who

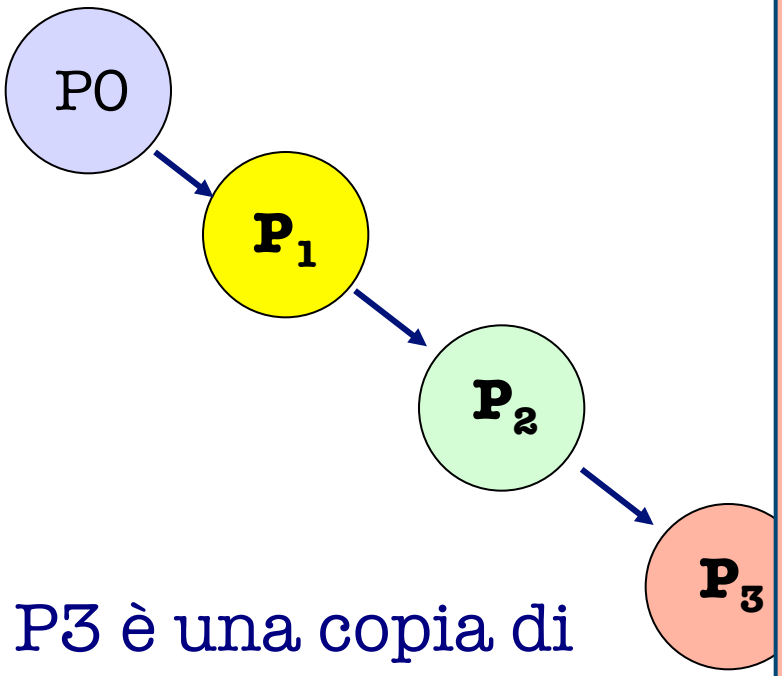
ora N=-1, quindi P3 stampa il messaggio ed esegue **`exit(0)`**! => P3 termina!

Processo P3: digressione

```
void gestoresequenza(int N, strvett vett)
{  int pid;
   pid=fork();
   if (pid==0) /* figlio*/
   {    if (N==-1) /* processo foglia */
        {    printf("\nfoglia %d: \n", getpid());
             exit(0);
        }
        else /*attivazione di un nuovo comando*/
        {   printf("\nProcesso %d per comando %s",getpid(),
                    vett[N]);
            gestoresequenza(N-1, vett);
            execlp(vett[N], vett[N], (char *)0);
            perror("\nexec fallita: ");
            exit(-1);
        }
   }
}
```

gestoresequenza(3):
N=-1
vett[0]=ps
vett[1]=who

se non ci fosse questa exit? cosa farebbe P3?




```
main:
        ncom=2
        vstr[0]=ps
        vstr[1]=who
gestoresequenza:(1)
        N=1
        vett[0]=ps
        vett[1]=who
gestoresequenza:(2)
        N=0
        vett[0]=ps
        vett[1]=who
gestoresequenza(3)
        N=-1
        vett[0]=ps
        vett[1]=who
```

Il contesto di P3 è una copia di quello di P2... quindi:

Se non ci fosse la exit(0), P3 uscirebbe dalla 3°chiamata a gestore sequenza e tornerebbe alla 2°...

Processo P3 se non ci fosse exit(0)

```
void gestoresequenza(int N, strvett vett)
{  int pid;
   pid=fork();
   if (pid==0) /* figlio*/
   {    if (N==-1) /* processo foglia */
        {    printf("\nfoglia %d: \n", getpid());
             exit(0);
        }
        else /*attivazione di un nuovo comando*/
        {   printf("\nProcesso %d per comando %s",getpid(),
                        vett[N]);
            gestoresequenza(N-1, vett);
            execlp(vett[N], vett[N], (char *)0);
            perror("\nexec fallita: ");
            exit(-1);
        }
   }
}
```

gestoresequenza(2):
N=0
vett[0]=ps
vett[1]=who

ps

P3 ritornerebbe alla 2° invocazione di gestoresequenza ed eseguirebbe ancora **ps**
Poi? ....ritornerebbe alla 1°???

Processo P3 se non ci fosse exit(0)

```
void gestoresequenza(int N, strvett vett)
{  int pid;
   pid=fork();
   if (pid==0) /* figlio*/
   {    if (N==-1) /* processo foglia */
        {    printf("\nfoglia %d: \n", getpid());
             exit(0);
        }
        else /*attivazione di un nuovo comando*/
        {   printf("\nProcesso %d per comando %s",getpid(),
                        vett[N]);
            gestoresequenza(N-1, vett);
            execlp(vett[N], vett[N], (char *)0);
            perror("\nexec fallita: ");
            exit(-1);
        }
   }
}
```

gestoresequenza(2):
N=0
vett[0]=ps
vett[1]=who

ps

NO!!! Non ritornerebbe alla 1° perché la exec sostituisce codice e contesto di P3 con ps... e l'exec non ha ritorno

Output del programma (originale)

```
bash-2.05$ vi ese_proc.c
bash-2.05$ gcc -o eseproc ese_proc.c
bash-2.05$ eseproc
quanti comandi? 2

dammi il prossimo comando(senza argomenti)ps

dammi il prossimo comando(senza argomenti)who

Processo 5591 per comando who
Processo 5592 per comando ps
foglia 5593:
root       pts/2        May  6 17:29    (deis125.deis.unibo.it)
root       pts/3        May  6 17:29    (deis125.deis.unibo.it)
anna       pts/5        May  9 10:46    (deis136.deis.unibo.it)
   PID TTY      TIME CMD
  5542 pts/5    0:00 bash
  5590 pts/5    0:00 eseproc
  5592 pts/5    0:00 ps
terminato processo figlio n.5591
term. volontaria con stato 0
```

Output del programma (originale)

```
bash-2.05$ vi ese_proc.c
bash-2.05$ gcc -o eseproc ese_proc.c
bash-2.05$ eseproc
quanti comandi? 2

dammi il prossimo comando(senza argomenti)ps

dammi il prossimo comando(senza argomenti)who

Processo 5591 per comando who
Processo 5592 per comando ps
foglia 5593:
root       pts/2        May  6 17:29    (deis125.deis.unibo.it)
root       pts/3        May  6 17:29    (deis125.deis.unibo.it)
anna       pts/5        May  9 10:46    (deis136.deis.unibo.it)
   PID TTY      TIME CMD
  5542 pts/5    0:00 bash
  5590 pts/5    0:00 eseproc
  5592 pts/5    0:00 ps
terminato processo figlio n.5591
term. volontaria con stato 0
```

P0: main
immissione
input

P1

P2

P3

P1: who

P2: ps

P0: gest_stato

Output del programma (originale)

```
bash-2.05$ vi ese_proc.c
bash-2.05$ gcc -o eseproc ese_proc.c
bash-2.05$ eseproc
quanti comandi? 2

dammi il prossimo comando(senza argomenti)ps

dammi il prossimo comando(senza argomenti)who

Processo 5591 per comando who
Processo 5592 per comando ps
foglia 5593:
root       pts/2        May  6 17:29    (deis125.deis.unibo.it)
root       pts/3        May  6 17:29    (deis125.deis.unibo.it)
anna       pts/5        May  9 10:46    (deis136.deis.unibo.it)
   PID TTY      TIME CMD
  5542 pts/5    0:00 bash
  5590 pts/5    0:00 eseproc
  5592 pts/5    0:00 ps
terminato processo figlio n. 5591
term. volontaria con stato 0
```

Output mescolato!

Rilevo solo lo stato di terminazione di P1 perché solo P0 fa `wait`

Spunti di discussione

- Output dei vari processi *mescolato: uso di file (o dispositivi) distinti*
- Come rilevare lo stato di terminazione di ogni processo???
- Pensare ad un'altra organizzazione per i processi:

  ad esempio: gerarchia a 1 solo livello con N figli
- Realizzare una variante in cui i comandi siano dati come argomenti dalla linea di comando