

Terza Esercitazione

Gestione di processi in Unix
Primitive Fork, Wait, Exec

Stefano Monti
stefano.monti6@unibo.it

System call fondamentali

<code>fork</code>	• Generazione di un processo figlio, che condivide il codice col padre e possiede dati replicati • Restituisce il pid del figlio per il padre, 0 per il figlio o un numero <0 in caso di errore
<code>exit</code>	• Terminazione di un processo • Si indica lo stato di terminazione; normalmente stato=0 se il processo termina correttamente, un valore diverso da 0 in caso di problemi
<code>wait</code>	• Raccolta dello stato di terminazione di un figlio (con eventuale attesa) • Restituisce il pid del figlio terminato e permette di capire il motivo della terminazione (Volontaria? Con quale stato? Involontaria? A causa di quale segnale?)
<code>exec</code>	• Sostituzione di codice e dati di un processo • NON crea processi figli

Esempio 1 - fork e exit

- Scrivere un programma in cui il processo padre procede alla istanziazione di un numero N di figli, tutti **fratelli**
`./generate <N> <term>` (con *term* [0,1])
- Attenzione all'indicazione di terminazione dei figli
 - Cosa succede se ci si dimentica di effettuare la `exit()` ?

3

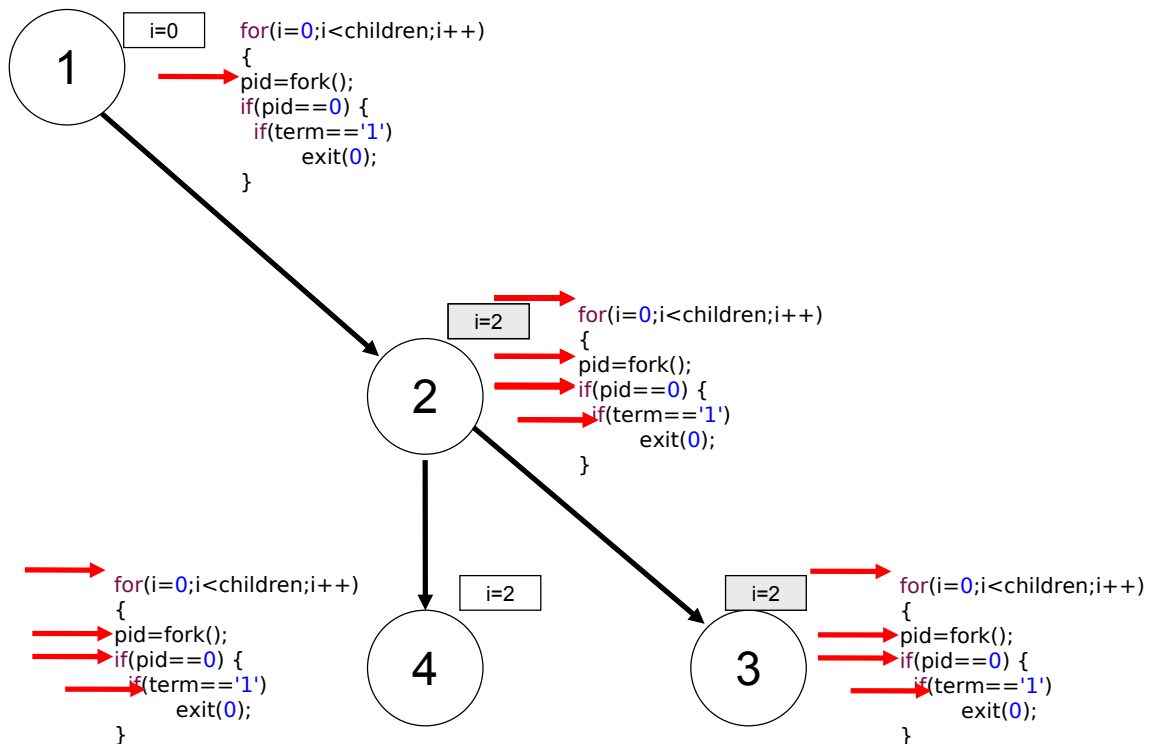
Esempio 1

```
int main (int argc, char * argv[]){
    int i,j,k,pid,status,children;
    char term;
    children=atoi(argv[1]);
    term=argv[2][0];
    for(i=0;i<children;i++) {
        pid=fork();
        if(pid==0) { //codice figlio
            if(term=='1')
                exit(0);
        }
        else if(pid>0) { //codice padre
            printf("%d)child created(pid=%d)\n", getpid(),pid);
        }
        else perror("fork error:");
    }
}
```



4

Assenza della `exit()`



5

Esercizio 1

Scrivere un programma C con la seguente interfaccia:

`./compilaEdEsegui <file1.c> <file2.c> .... <fileN.c>`

dove `file1.c, ..., fileN.c` sono file sorgenti C.

Il processo padre deve **generare $2*N$ processi** (figli e/o nipoti),

- 2 per ciascun sorgente; per ogni file,
- uno dei processi figli/nipoti si incaricherà di **compilare** il file,
- un altro processo figlio/nipote (DISTINTO dal precedente) di **metterne in esecuzione** l'eseguibile risultante.

Si generino i processi figli sequenzializzando il meno possibile le operazioni di compilazione ed esecuzione.

6

Vincoli di sincronizzazione

- I processi **compilatori** possono essere messi in esecuzione in maniera **concorrente**, ma...
- La **compilazione deve avvenire prima dell'esecuzione** --> il processo che esegue deve sincronizzarsi col processo che compila

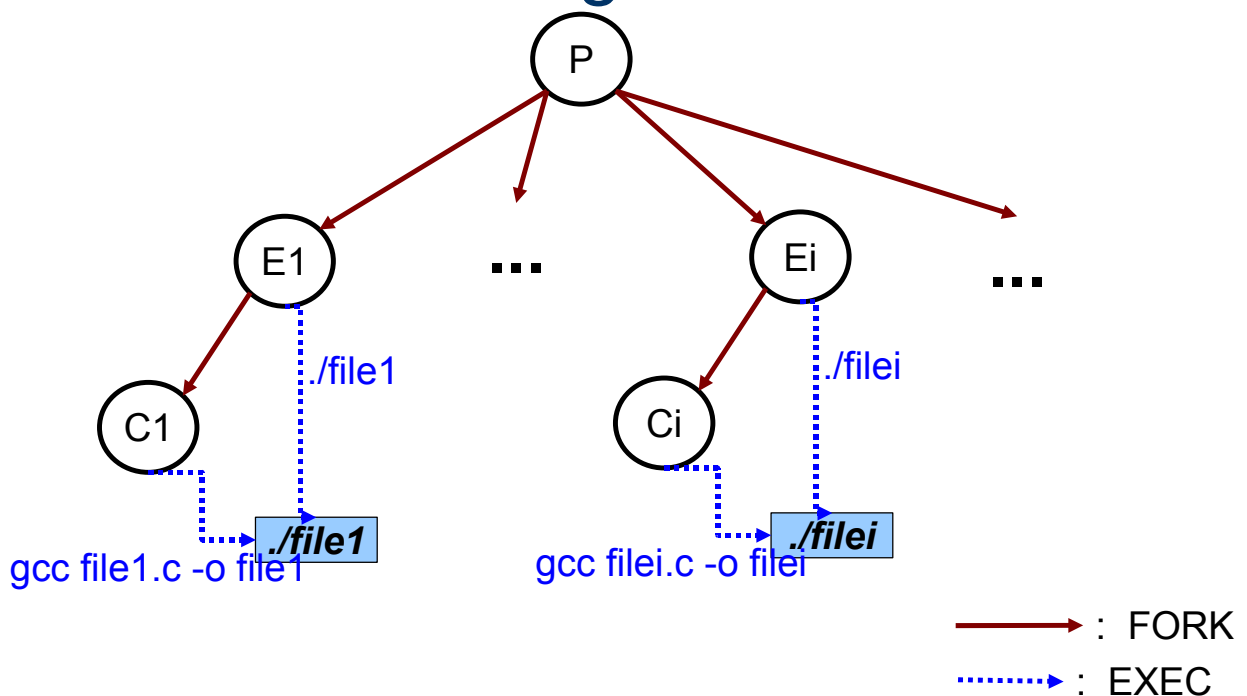


una possibile soluzione

- il processo esecutore **ATTENDE** il termine dell'esecuzione del processo compilatore --> **relazione di gerarchia**

7

Schema di generazione

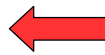


8

```

for ( i=1; i<argc;i++){
    pid = fork();
    if (pid == 0){ /* figlio i-esimo */
        pid = fork();
        if (pid == 0){ /* nipote i-esimo : compilazione */
            printf("Nipote: compilazione %s\n",argv[i]);
            execl("/usr/bin/gcc", "gcc", argv[i], "-o", executableName, (char *)0);
            perror("Errore in execl\n");
            exit(1); }
        else if (pid > 0){ /* figlio i-esimo : esecuzione */
            printf("Figlio: esecuzione %s\n",executableName);
            wait(&status); //attesa terminazione nipote
            execl(executableName, executableName, (char *)0);
            perror("Errore in execl\n");
            exit(1); }
        else{ perror("Errore in fork\n"); exit(1); }
        exit(0); }
    else if (pid > 0){ /* padre */
        printf("Padre ....\n");
    }
    else {perror("Errore in fork\n"); exit(1); }
}
for ( i=1; i<argc;i++){
    wait(&status);
}

```



- quali processi eseguono questa porzione di codice?
- a cosa serve?
- perché è al di fuori del ciclo **for** di generazione?

9

Esercizio 2

Si realizzi un programma, che, utilizzando le system call del sistema operativo UNIX, soddisfi le seguenti specifiche:

Sintassi di invocazione:

eseguiComandi K COM1 COM2 ... COMN

Significato degli argomenti:

- **eseguiComandi** è il nome del file eseguibile associato al programma.
- COM1, COM2, ... ,COMN sono N stringhe che rappresentano il nome di un file (per semplicità, si supponga che il direttorio di appartenenza del file COM sia nel PATH)
- K è un valore intero positivo (minore di N)

10

Specifiche

Il processo iniziale (P0) deve **mettere in esecuzione** gli N comandi passati come argomenti, secondo la seguente logica:

- i primi **K** comandi passati come argomenti dovranno essere eseguiti **in parallelo** da **altrettanti figli** di P0
- al **termine** dei primi K processi, i restanti **N-K** comandi dovranno essere eseguiti **in sequenza** da **altrettanti figli e/o nipoti** di P0

11

Esercizio per casa

Supponiamo che un processo padre P0 generi N figli

- due casi: generazione di
 - N processi fratelli (gerarchia a 2 livelli: padre e N fratelli)
 - N processi, ciascuno figlio del precedente (gerarchia a N + 1 livelli: padre e N figli, nipoti, pronipoti, ecc...)
- struttura dati **globale** per tenere traccia dei pid dei processi (padre + figli)

```
int pid [N+1];
```

- il risultato di ciascuna **fork** salvato nell'array, ad esempio:

```
int i = 0;
pid[0] = getpid(); //pid del padre
for (i= 0; i<N; i++){
    pid[i] = fork();
    if (pid[i] == 0){ ... }
    else{ ... }
}
```

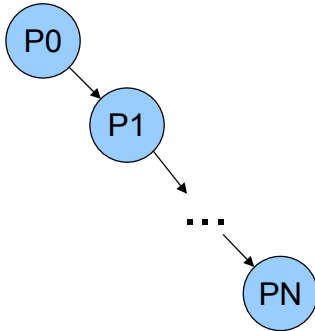
Quale tipo di gerarchia di processi viene generata da questo codice?

12

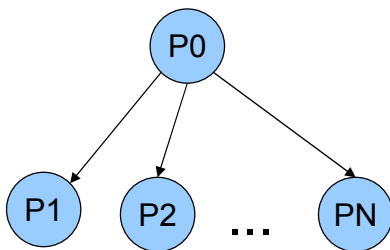
Esercizio per casa

`int pid [N]`

- con quali valori viene riempita?
- è uguale per tutti i processi?



	pid[0]	pid[1]	pid[2]	...	pid[N]
P0					
P1					
P2					
...					
PN					



	pid[0]	pid[1]	pid[2]	...	pid[N]
P0					
P1					
P2					
...					
PN					