Terza Esercitazione

Gestione di processi in Unix

Gestione di segnali in Unix

Stefano Monti
stefano.monti6@unibo.it

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.

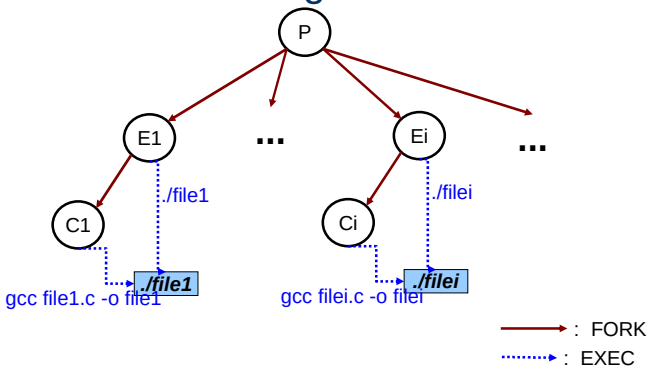
Vincoli di sincronizzazione

- 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**

Schema di generazione



```
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?

Esercizio 2

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

Sintassi di invocazione:

eseguiInSequenza ***N COM1 COM2***

Significato degli argomenti:

- eseguiInSequenza è il nome del file eseguibile associato al programma.
- N e` un valore intero positivo.
- COM1 e COM2 sono stringhe che rappresentano il nome di un file (per semplicita`, si supponga che il direttorio di appartenenza del file COM sia nel PATH)

Specifiche

Il processo iniziale (P0) deve creare 2 processi figli (fratelli) P1 e P2.

I processi P1 e P2 devono eseguire rispettivamente i comandi COM1 e COM2, con il vincolo esplicito che **COM1 venga eseguito soltanto al termine della esecuzione di COM2**.

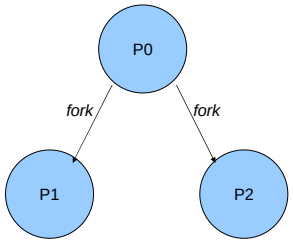
Considerando che i file COM1 e/o COM2 potrebbero non essere eseguibili, in caso di fallimento dell'esecuzione il processo figlio che ne è responsabile dovrà **notificare tale evento al padre P0** e successivamente terminare.

In caso di successo nell'esecuzione del comando di pertinenza, dopo N secondi dalla sua creazione, il processo figlio **P1** dovrà comunque **terminare la propria esecuzione**.

Note alla soluzione

- P0 crea P1 e P2: fork e **attende**
- P1 deve terminare dopo N secondi (qualunque sia il programma che esegue):
  - impostazione di timeout
  - gestione dei segnali & exec
- in caso di fallimento nell'esecuzione di COM(1,2) P(1,2) deve **notificare** l'evento a P0: invio di segnali e relativa gestione
- gestione della esecuzione "sequenzializzata" (COM2 prima di COM1):
  - P1 e P2 sono fratelli: quali strumenti a disposizione per sincronizzare P1 e P2?
  - può P2 eseguire COM2 e, in caso di successo, inviare un segnale a P1?

Schema di generazione (iniziale) dei processi



9

Variante

- Il padre P0, mentre il figlio esegue, continua la propria attivita` (cioe` non si pone in attesa di P1 e P2), per stampare il suo PID ripetutamente;
- tuttavia, al termine di P1 e P2, P0 dovra` tempestivamente raccogliere e stampare lo stato di terminazione dei figli (v. gestione del sigchld).

10