Sesta Esercitazione

Esercitazione conclusiva sulla programmazione di sistema Unix

Andrea Reale
andrea.reale@unibo.it

Agenda

- **Esercizio d'esame su programmazione Unix**
  - Esercitazione di sintesi su tutte le problematiche di programmazione C/Unix affrontate fin ora
  - Tratto (in versione semplificata) dall'appello d'esame del 16 Settembre 2008

Traccia (1/5)

Si scriva un programma C che, utilizzando le system call UNIX, abbia un'interfaccia del tipo:

contaMaiuscole fileIn fileOut

- **fileIn** path di un file esistente nel filesystem
- **fileOut** path del file che deve essere creato come risultato dell'esecuzione del programma

Dopo aver effettuato gli opportuni controlli sui parametri di invocazione, il processo iniziale **P0** deve generare un processo figlio **P1** che a sua volta generara` un processo nipote **P2**.

Traccia (2/5) - Comportamento

- **fileIn** contiene una sequenza di caratteri organizzate in righe:
  - Ogni riga e` lunga al piu` 80 caratteri
  - Il numero di righe totale non e` noto a priori
  - Ciascuna riga termina con il carattere '+' oppure con il carattere '-' (non si sa a priori)

- I processi figlio e nipote devono leggere **concorrentemente** l'intero **fileIn**
  - **P1** si occupera` delle sole righe che terminano con '+'
  - P2 si occupera` delle sole righe che terminano con '-'

- Dopo aver letto ciascuna riga di pertinenza, **P1** e **P2** dovranno inviare un messaggio a **P0** contenente i soli caratteri maisucoli trovati nella riga. Ogni messaggio deve essere terminato dal carattere terminatore **'\0'**.

Traccia (3/5) - Comportamento

- I processi figlio devono comunicare con il padre facendo uso ciascuno di una propria pipe condivisa con P0
  - Due processi scrittori (**P1** e **P2**) e uno lettore (**P0**)

- **P0** deve leggere i messaggi dalla pipe
  - Tenere traccia del numero totale di caratteri maiuscoli trovati da ciascun processo
  - Scrivere volta per volta il contenuto dei messaggi su fileOut

Traccia (4/5) - Comportamento

- Una volta terminata la lettura del file e la ricezione di tutti i messaggi da parte di **P0**
  - **P0** deve comunicare l'esito del conteggio sia a **P1** che a **P2**
  - **A questo scopo, dovranno essere riutilizzate le stesse pipe condivisa, questa volta in verso opposto**

Il processo "vincente" dovra` scrivere su **fileOut** "Sono il processo **pid** e ho realizzato il conteggio **x** piu` alto" (**pid** e **x** sostituiti opportunamente)

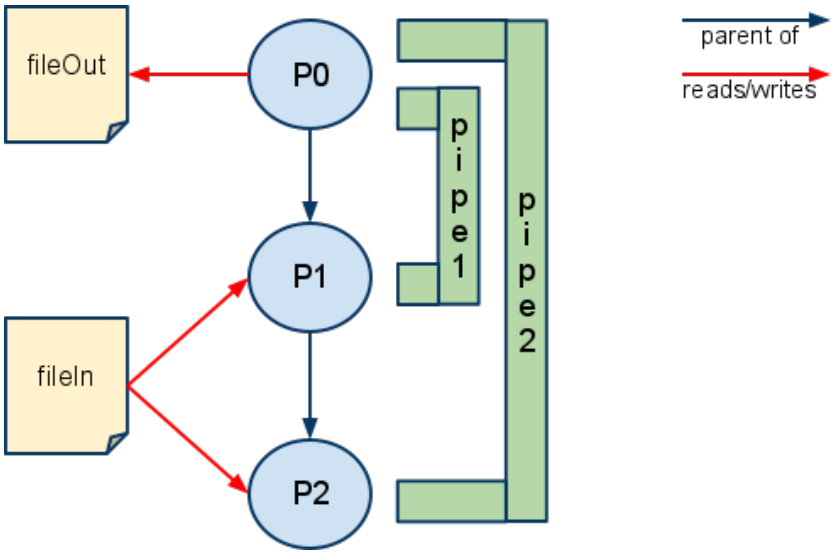
Traccia (5/5) - Comportamento

Inoltre, **in qualunque istante**, deve essere possibile per l'utente forzare la modifica del comportamento del programma concorrente premendo la combinazione di tasti <CTRL-C>.

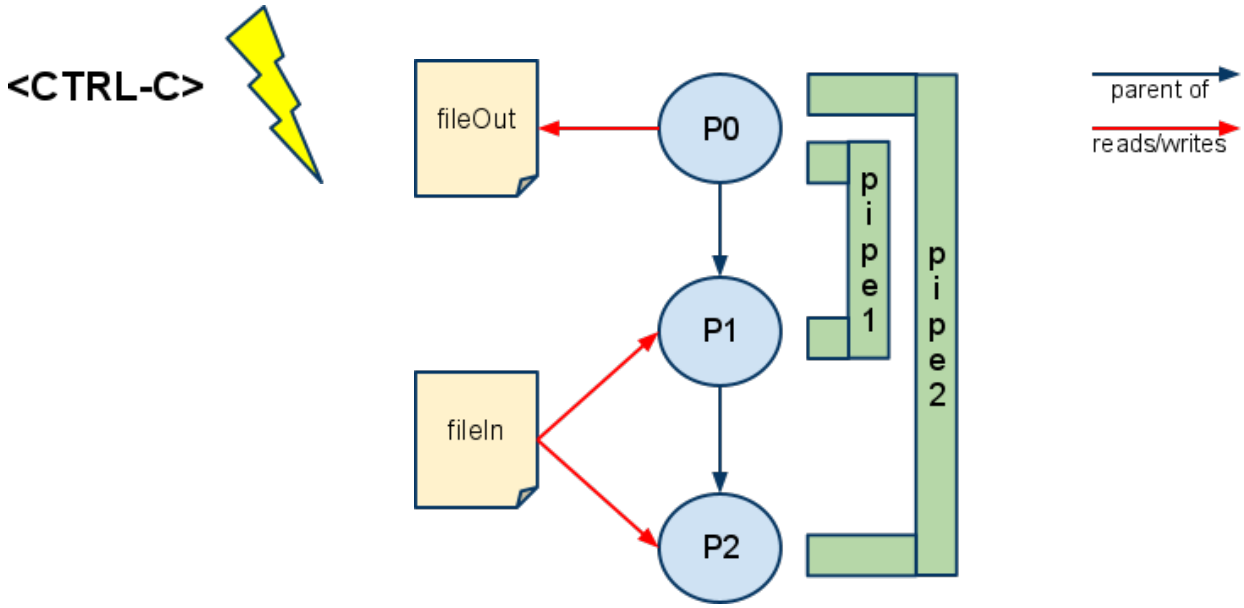
- Se <CTRL-C> viene premuto due volte in meno di 1 secondo, allora **P2** dovra` essere terminato
  - **P1** dovra` occuparsi di tutte le righe ancora non processate (anche quelle che terminano con '**-**').
  - Per semplicita`, **P1** non dovra` considerare le righe non ancora processate da **P2** che siano *prima* dell'I/O pointer di **P1**

- Se <CTRL-C> viene premuto una sola volta (nell'arco di un secondo), allora **P1** dovra` essere terminato
  - In maniera speculare al caso precedente, **P2** dovra` occuparsi di tutte le righe ancora non processate

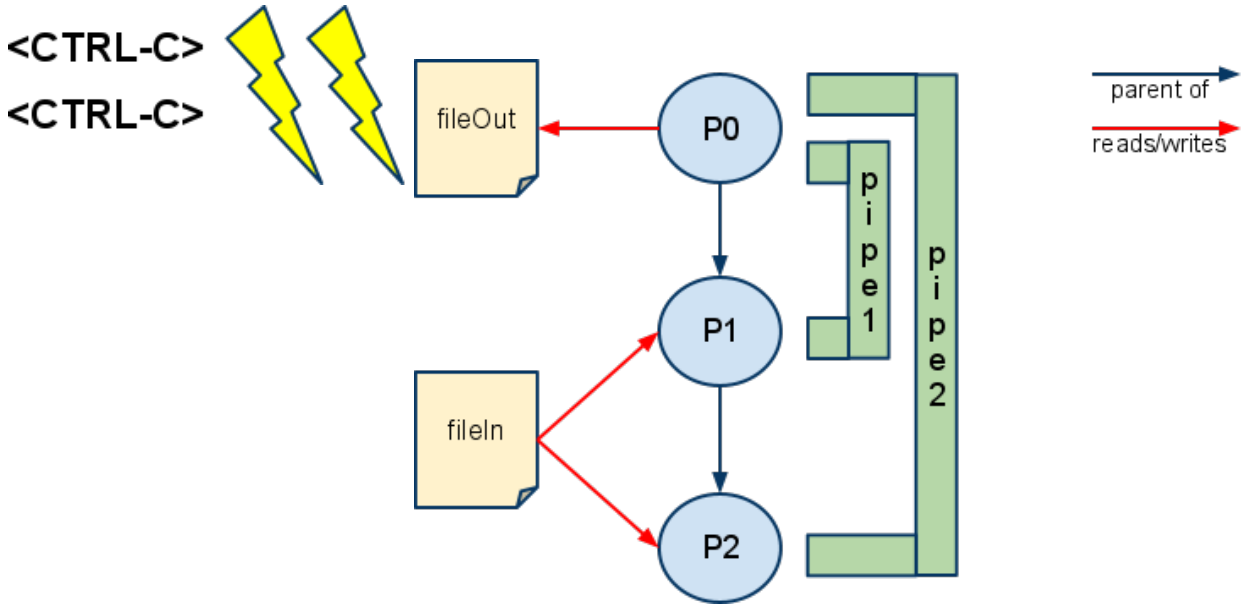
Modello di soluzione - Struttura di base



Modello di soluzione - Struttura di base



Modello di soluzione - Struttura di base



Esempio di input

$ ./contaMaiuscole /home/andrea/file_in ./file_out

/home/andrea/file_in:

In in**F**ormatica, u**N**a chiamata di sistema +
(in in**G**lese *systeM caLL*) è il meccanismo+
usato da un **P**ro**G**ramma a l**I**vello **UT**ente-
per richiedere un servi**Z**io a livello +
Ker**N**el del sistema **O**perat**I**vo.-

Esempio di input

$ ./contaMaiuscole /home/andrea/file_in ./file_out

/home/andrea/file_in:

P1

In inFormatica, uNa chiamata di sistema +
(in inGlese *systeM caLL*) è il meccanismo+
usato da un ProGramma a lIvello UTente-
per richiedere un serviZio a livello +
KerNel del sistema OperatIvo.-

Esempio di input

$ ./contaMaiuscole /home/andrea/file_in ./file_out

/home/andrea/file_in:

In inFormatica, uNa chiamata di sistema +
(in inGlese *systeM caLL*) è il meccanismo+
usato da un ProGramma a lIvello UTente-
per richiedere un serviZio a livello +
KerNel del sistema OperatIvo.-

P1

P2

Problematiche da affrontare (1/2)

a) Divisione del lavoro tra i processi

- **P1** e **P2** decidono autonomamente quali sono le righe che spettano ad uno, quali all'altro

b) Lettura concorrente da due pipe

- P0 deve leggere contemporaneamente (o quasi) da due sorgenti

Problematiche da affrontare (1/2)

c) Sincronizzazione ed inversione dei ruoli lettori/scrittori

- **P0** deve accorgersi quando entrambi i figli hanno terminato la lettura del file di ingresso *(senza che i figli abbiano chiuso la pipe)*
- **P1** e **P2** dovranno iniziare a leggere dati dalla pipe non appena P0 sara` pronto a scriverci

Problematiche da affrontare (2/2)

d) Gestione dei segnali da tastiera da parte dell'utent e

- Che succede quando viene schiacciato <CTRL-C>?
- Come riconosco un <CTRL-C> singolo rispetto ad un <CTRL-C> doppio?
- Il segnale puo` arrivare in qualsiasi momento dell'elaborazione

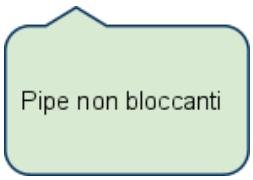
Problema a)

- **Come fa P1 (o P2) a capire se una riga spetta a se?**
  - Lettura di ogni riga ed individuazione del carattere '+' oppure '-' finale

- **Individuazione delle lettere maiuscole**
  - **char** rappresentazione di ogni lettera come 8 bit codificati in codice ASCII
  - Lettere da A-Z e a-z sono stringhe di bit consecutive
  - A=65, B=66, ..., Z=90 ; a=97, b=98, ..., z=122

Problema b)

- **Lettura concorrente da più di una pipe**
  - **Alternativa 1:** leggo dalla prima fin quando c'è da leggere, poi passo alla seconda
  - **Alternativa 2:** alterno una lettura di qua, una di là
  - **Alternativa 3:** ???



c) Inversione del ruolo lettori/scrittori

- Come fa **P0** a sapere quando **P1** e **P2** hanno finito di scrivere le loro elaborazioni sulla pipe?
  - Se **P1** e **P2** chiudessero il loro lato di scrittura resterebbe aperto quello di **P0**. Il solito schema **while(read (...) > 0) non funzionerebbe.**
  - **Alternative?**

- **P0**, **P1**, e **P2** si accordano per speciali **messaggi di terminazione**.
  - **es**. P1 invia il messaggio **"!"** quando ha finito di leggere la sua parte di file. P2, fa lo stesso.

- **P1** e **P2** devono iniziare a leggere sulle rispettive pipe, solo dopo che **P0** ha consumato tutti i loro messaggi
  - Altrimenti potrebbero leggere e consumare i loro stessi messaggi!!

Problemi b) e c)

- **Esempio (Alternativa 2) + Fine lettura**

```
   // Processo P0
while( !term1 || | !term2) {
   msg_received_1 = 0;
   while ( !term1 && !msg_received_1 && read(pipe1fd[0], &c, 1) > 0) ) {
      ...
      else if ( c == '\0') {
         msg_buf[i] = c;
         msg_received_1 = 1;
         /* scrivi il messaggio sul file di output*/
         /*conta le maiscole presenti nel messaggio
           e aggiorna il conteggio riguardante P1*/
      }
      else if ( c == '!' ) {
         term1 = 1;
      }
   }
   i=0; msg_received_2 = 0;
   while ( !term2 && !msg_received_2 && read(pipe2fd[0], &c, 1) > 0 ) {
      ...      ...
   }
}
```

d) Gestione dei segnali utente

- I segnali dell'utente da tastiera (pressione di <CTRL+C>) sono completamente **asincroni** rispetto all'esecuzione del programma
  - Possono arrivare in qualsiasi momento dell'esecuzione di ciascun processo

- La pressione di <CTRL-C> fa` si che sia inviato un segnale di tipo SIGINT a tutti i processi in foreground che stanno girando sul terminale attivo.
  - Nel nostro caso, a tutta la gerarchia di figli nipoti
  - Necessario gestirli **in maniera opportuna** sia per **P0** che per **P1** e per **P2**

d) Gestione dei segnali utente

- Come distinguere un singolo <CTRL-C> da **due** <CTRL-C> arrivati *entro un secondo di distanza* l'uno dall'altro?
- **Assunzioni**:
  - Modello dei segnali affidabili, con accodamento di tutti i segnali ricevuti.
  - Piu` di due <CTRL-C> in meno di un secondo contano come due.

- Il processi che hanno bisogno di fare la distinzione devono mantenere lo stato dei segnali SIGINT ricevuti nell'ultimo secondo.

- **Tre possibilita`** da memorizzare in una variabile (*da resettare opportunamente*):
  - nessun SIGINT ricevuto
  - ricevuto un SIGINT, in attesa di un eventuale secondo
  - ricevuti due SIGINT a distanza minore di un secondo

d) Gestione dei segnali utente

- **Lo stato** sara` memorizzato, per ciascun processo, ad esempio, impostando una variabile globale ad uno di tre valori
  - es. **int** sig_received = 0, 1 oppure 2

- Gestione combinata di SIGINT e SIGALRM per la rilevazioni di **doppi <CTRL+C>**.
  - Alla ricezione del primo SIGINT, l'handler *schedula* un allarme (SIGALRM) per se stesso dopo 1 secondo (vedi system call **alarm**)

- L'handler di SIGINT dovra` comportarsi diversamente a seconda dello stato che trovera` in int_received
  - **sig_received == 0 ?** incrementalo ad 1 e imposta un allarme fra un secondo.
  - **sig_received == 1 ?** Incrementalo a 2.
  - **sig_received == 2 ?** Inutile incrementarlo ulteriormente, non fare nulla.

d) Gestione dei segnali utente

- *Una volta distinta l'intenzione dell'utente* (segnale singolo o doppio) **ogni processo** dovra` gestire il segnale ricevuto in maniera diversa, a seconda delle specifiche di comportamento

	P0	**P1**	**P2**
<CTRL-C>	Considerare che **P2** e` terminato ai fini della sincronizzazione per l'inversione dei ruoli sulla pipe.	Stato Dal momento della ricezione del segnale in poi, dovra` elaborare tutte le righe del file di input	Chiudere tutte le risorse aperte e terminare
<CTRL-C> <CTRL-C>	Considerare che **P1** e` terminato ai fini della sincronizzazione per l'inversione dei ruoli sulla pipe.	stato Chiudere tutte le risorse aperte e terminare	Dal momento della ricezione del segnale in poi, dovra` elaborare tutte le righe del file di input

Variante 1

- Come andrebbe modificato il programma se volessimo che, ogniqualvolta uno tra **P1** e **P2** viene interrotto da un segnale utente, il processo sopravvisuto elabori - *se presenti* - a*nche le righe non ancora elaborate dal processo ucciso che siano precedenti al suo I/O Pointer*?

In inFormatica, uNa chiamata di sistema +
(in inGlese *systeM caLL*) è il meccanismo+
usato da un ProGramma a lIvello UTente-
per richiedere un serviZio a livello +
KerNel del sistema Operativo.-

P2 I/O Pointer

P1 I/O Pointer

Variante 2

- Cosa succederebbe se P0 comunicasse con P1 e P2, non più con un una pipe per processo, ma condividendo una sola pipe per entrambi i processi?
  - Quali nuove problematiche entrerebbero in gioco?
  - (Variante originale dell'esame)