Università di Ferrara
Laurea Triennale in Informatica
A.A. 2021-2022
Sistemi Operativi e Laboratorio

Lab-06a. Pipe

Prof. Carlo Giannelli

http://www.unife.it/scienze/informatica/insegnamenti/sistemi-operativi-laboratorio
http://docente.unife.it/carlo.giannelli
https://ds.unife.it/people/carlo.giannelli

Inter Process Communication (IPC)

Per realizzare una comunicazione tra processi che risiedono sulla stessa macchina si utilizzano le **pipe**

- **pipe (man 2 pipe)**

```
#include <unistd.h>
int  pipe(int fildes[2]);
```

  - la system call **pipe** alloca una coppia di file descriptor (fildes[2]) che identificano un **canale di comunicazione unidirezionale** tra processi (su una stessa macchina)
  - **pipe** ritorna 0 in caso di successo e -1 in caso di errore
  - possiamo leggere con **read** su fildes[0] quello che è stato scritto con **write** su fildes[1]
  - ricordarsi di chiudere il file descriptor non utilizzato
  - si può utilizzare la system call **dup** per duplicare i file descriptor

Esercizio 1 – Consegna 1/2

Si progetti un'applicazione concorrente multi processo che permetta al personale dell'Ufficio Segreteria di Ateneo di controllare il numero di esami sostenuti dagli studenti.
L'applicazione dovrà essere realizzata in C e presentare la seguente interfaccia:

controlla_esami file_dati

dove **file_dati** è un nome assoluto di file con le informazioni sugli esami sostenuti dagli studenti. In particolare, si supponga che **file_dati** contenga una riga per ogni esame sostenuto: numero di matricola dello studente, nome del corso, numero di crediti, voto riportato.

Esercizio 1 – Consegna 2/2

L'applicazione deve essere composta da un processo iniziale P0 che si interfaccia con l'utente da cui riceve (via terminale) il numero di matricola di uno studente.

Per ogni richiesta, P0 deve generare 2 figli:

- P1 si deve occupare di selezionare le informazioni sugli esami sostenuti dallo studente desiderato (prendendole da **file_dati**) e inviarle a P2
- P2 deve contare il numero di esami sostenuti dallo studente e stamparlo a video.

Dopodiché P1 e P2 terminano e P0 si mette in attesa di una nuova richiesta da parte dell'utente. L'applicazione termina quando l'utente inserisce la stringa "fine".

Esercizio 1 - Traccia

controlla_esami.c

- **#include** degli header file controllo degli argomenti
- chiedo una nuova matricola
- finché matricola != "fine"
- creazione pipe tra p1 e p2
- genero P1
- se sono nel codice di P1:
  - chiudo i descrittori non necessari
  - redirezione dello stdout
  - ricavo e invio le informazioni a P2 (system call)
- genero P2
- se sono nel codice di P2:
  - chiudo i descrittori non necessari alla redirezione dello stdin
  - eseguo il conteggio (system call)
- attendo la terminazione dei figli (e itero)

Esercizio 2 – Consegna 1/2

Si progetti un'applicazione concorrente multi processo che permetta al responsabile di produzione di una fabbrica di verificare la disponibilità della necessaria quantità di materie prime nei magazzini aziendali. L'applicazione dovrà essere realizzata in C e presentare la seguente interfaccia:

controlla_disponibilita magazzino-1 magazzino-2 ... magazzino-N

dove **magazzino-1, magazzino-2, ..., magazzino-N** sono nomi relativi di file.

L'applicazione deve essere composta da un processo iniziale P0 che prima di tutto genera N figli (P1, P2,..., PN), uno per ogni magazzino specificato. P0 si interfaccia quindi con l'utente, da cui riceve (via terminale) il nome della materia prima di interesse.

Esercizio 2 – Consegna 2/2

A questo punto, P0 deve comunicare il nome della materia prima a ciascun processo Pi, che a sua volta dovrà restituire a P0 il numero di unità di stoccaggio disponibili nel magazzino-i.

A questo fine, ogni processo Pi deve consultare il file di nome **magazzino-i** che si suppone contenere una riga (con nome della materia prima e informazioni sullo scaffale dove si trova) per ogni unità di stoccaggio presente in magazzino.

P0 deve quindi stampare a video il numero di unità di stoccaggio di materia prima disponibili in ciascun magazzino e infine rimanere in attesa della prossima richiesta. L'applicazione termina quando l'utente inserisce la stringa "fine".

Esercizio 2 – Traccia 1/2

- **#include** degli header file e controllo degli argomenti (N magazzini)
- creazione N pipe e N figli
- una volta per ciascun figlio:
  - apro pipe di input (da padre a figlio i-esimo) e output (da figlio i-esimo a padre)
  - genero il figlio i-esimo
  - se sono il figlio:
    - chiudo le pipe aperte precedentemente verso altri figli e quelle che non servono
    - finché ci sono nuove stringhe provenienti dal padre (lettura da pipe di input):
      - genero un nipote
      - se sono nel nipote:
        - chiusura delle pipe non necessarie
        - redirezione stdout su pipe di output verso il padre (ereditata dal figlio)
        - exec del comando per contare il numero di unità di stoccaggio
    - chiudo le pipe verso il padre e termino il figlio

Esercizio 2 – Traccia 2/2

- nel codice eseguito dal padre:
- chiudo le pipe che non servono
- per ogni stringa inserita dall'utente:
  - la invio a tutti gli N figli (scrittura sulle pipe di input)
  - leggo risposte da ciascun nipote e le stampo a video
  - chiudo pipe aperte e attendo terminazione dei figli

Esercizio 3 – Consegna 1/2

Un’applicazione multi-processo in C deve essere invocata come segue:

ricerca_files Timeout S1 S2 . . . Sn

dove **Timeout** è un intero e **S1 S2 . . . Sn** sono stringhe alfanumeriche.

Obiettivo dell’applicazione è quello di richiedere all’utente di fornire dei nomi di file, all’interno dei quali l’applicazione deve contare quante righe contengono le varie stringhe passate come parametro.

In dettaglio, l’applicazione deve essere composta da un processo iniziale P0 che, prima di tutto, genera n ulteriori processi figli P1, P2, ..., Pn (tanti quante sono le stringhe passate come parametri).

Esercizio 3 – Consegna 2/2

Il processo P0 si interfaccia con l'utente, da cui riceve (via terminale) i nomi (relativi) di file su cui eseguire la ricerca. P0 invia ogni nome di file, uno alla volta, a tutti i processi figli e ogni processo Pi ha il compito di **contare quante volte la stringa Si compaia all'interno del file** inserito dall'utente e inviare tale numero a P0, il quale si incaricherà di stampare i risultati in ordine (da P1 fino a Pn).

Dopo la stampa dei risultati, P0 richiede l'inserimento di un nuovo nome di file all'utente. L'operazione di conteggio deve essere ripetuta per ogni nome di file inserito dall'utente.

L'applicazione termina quando l'utente preme "ctrl-c". In tale caso, l'applicazione deve avere cura di lasciare Timeout secondi ai processi per cercare di finire eventuali ricerche in corso. Trascorsi Timeout secondi, tutti i processi devono terminare.

Esercizio 3 – Traccia 1/2

- **#include** degli header file

- definizione dei gestori dei segnali (ctrl-c e terminazione definitiva)
- controllo degli N+1 argomenti (N numero di stringhe)

- generazione delle N pipe e dei N figli
- per ciascun figlio:
  - ignorare i segnali gestiti dal padre
  - chiusura pipe non collegate al figlio i-esimo e gli estremi non utilizzati
  - while(true)
    - generazione nipote:
    - se nipote:
      - exec grep
    - se figlio:
      - attesa terminazione nipote

Esercizio 3 – Traccia 2/2

- se codice del padre:
  - installazione gestori dei segnali
  - chiusura degli estremi inutilizzati delle pipe
  - while(true):
    - interfacciamento con l'utente
    - invio nome file ai figli
    - lettura e stampa risultato