struct element_db

struct query

struct shm_out

enum semafori

Sistemi Operativi
prova di laboratorio
– 15 dicembre 2020 –

Creare un programma **`lookup-database.c`** in linguaggio C che accetti invocazioni sulla riga di comando del tipo:

```
lookup-database <db-file> <query-file-1> <query-file-2>
```

Il programma dovrà fondamentalmente leggere un file *database* con coppie del tipo *(nome, valore)* e ricercare al suo interno i nomi che compaiono nei file di *query*, sommando i valori riportati separatamente (un totale per ogni file di *query*). Il file database è un file testuale in cui ogni riga ha la struttura *"nome:valore"* dove *nome* può contenere degli spazi e/o altri simboli e *valore* è un numero intero; i file di *query* sono file testuali con un nome per ogni riga. Vedere i file di esempio allegati.

Il programma una volta avviato si istanzierà in un processo che chiameremo **P**: questo al suo avvio creerà n.4 processi figli: **IN1**, **IN2**, **DB** e **OUT**. I processi **IN1**, **IN2** e **DB** useranno un segmento di memoria condiviso per comunicare; i processi **DB** e **OUT** ne useranno un secondo distinto. Per coordinare i processi si dovranno usare semafori nel numero minimo indispensabile.

I ruoli dei quattro processi saranno i seguenti:

main
- il processo **P** dovrà occuparsi anche di creare le strutture di IPC e di distruggerle una volta terminati gli altri processi;

in_child
- il processi **IN1** e **IN2** avranno il compito di leggere il contenuto, rispettivamente, dal primo e dal secondo file di *query*; per ogni nome letto invieranno, usando il primo segmento e i semafori opportuni, una *query* al processo **DB**;

struct node e typedef node* list inserimento, ricerca, distruzione nella lista conversione riga del file in un elemento db (entry) e carica tutte le entru del db in una lista db_child
- il processo **DB** preliminarmente leggerà l'intero file *database* e ne manterrà il contenuto in memoria RAM in una o più strutture dati (a scelta); il file potrebbe avere un numero arbitrario di righe non noto a priori; successivamente, per ogni *query* ricevuta, ricercherà il nome nella struttura del *database* e, se riscontrato un match (di tipo *case sensitive*), allora invierà al processo **OUT** , usando il secondo segmento, i campi *(processo, nome, valore)*, dove *processo* indicherà l'identità del richiedente;

out_child
- il processo **OUT**, terrà traccia dei record ricevuti e alla fine visualizzerà i totali separati dei valori ricevuti (uno per ogni processo lettore).

I processi dovranno terminare correttamente e spontaneamente alla fine dei lavori, liberando qualunque struttura persistente di IPC.

Nota: è permesso ai due processi IN1 e IN2 di leggere preliminarmente il file per determinare il numero di record da allocare.

Tempo: 2 ore e 30 minuti

intente la lista e tramite una funzione caricherà il contenuto.

poniamo un elemento del database = cerchiamo il valore specificato nella query, all'interno del database. Se è stato trovato allora salviamo il valore alla memoria condivisa out includo l'id della query, done = 0 e sarà risvegliato s_out

In allegato saranno forniti dei file di esempio: *database.txt*, *query-1.txt* e *query-2.txt*

L'output tipo atteso su tali file è il seguente:

```
$ ./lookup-database database.txt query-1.txt query-2.txt

IN1: inviata query n.1  'distesero'
IN2: inviata query n.1 'aguglia'
IN2: inviata query n.2 'telecinecamera'
IN1: inviata query n.2 'bastione'
DB: letti n. 50000 record da file
IN1: inviata query n.3 'prossimale'
DB: query 'distesero' da IN1 trovata con valore 28809
DB: query 'aguglia' da IN2 non trovata
DB: query 'telecinecamera' da IN2 trovata con valore 26148
DB: query 'bastione' da IN1 non trovata
DB: query 'prossimale' da IN1 non trovata
[…]
IN1: inviata query n.120 'olà'
IN2: inviata query n.120 'frasami'
DB: query 'olà' da IN1 trovata con valore 17764
DB: query 'frasami' da IN2 non trovata
OUT: ricevuti n.93 valori validi per IN1 con totale 1513996
OUT: ricevuti n.86 valori validi per IN2 con totale 1428916
```