Sistemi di Calcolo - Modulo 2 (A.A. 2014-2015)

Secondo appello - 14 luglio 2015

Tempo a disposizione: 1h 30'.

> *Attenzione*: assicurarsi di compilare il file **studente.txt** e che il codice prodotto non contenga **errori di compilazione**, pena una valutazione negativa dell'elaborato.

Realizzazione di un'applicazione per il calcolo parallelo di statistiche sulla lunghezza delle parole di un testo

L'obiettivo di questa prova è completare il codice di un'applicazione che prende in input un file testuale e lo elabora in parallelo per contare le occorrenze delle parole in base alla loro lunghezza. L'output di questa elaborazione sarà un array di interi in cui l'elemento `i`-esimo contiene il numero di parole di lunghezza `i` presenti nel testo in input.

L'applicazione lancia N processi figlio, ognuno dei quali riceve via pipe dal processo padre una parte delle parole del testo e tiene traccia della loro lunghezza. Quando sono state inviate tutte le parole, il processo padre chiude tutte le pipe ed inizia a recuperare gli output parziali prodotti dai processi figlio. Tale recupero avviene tramite un buffer circolare virtuale gestito secondo il paradigma produttore/consumatore in cui:

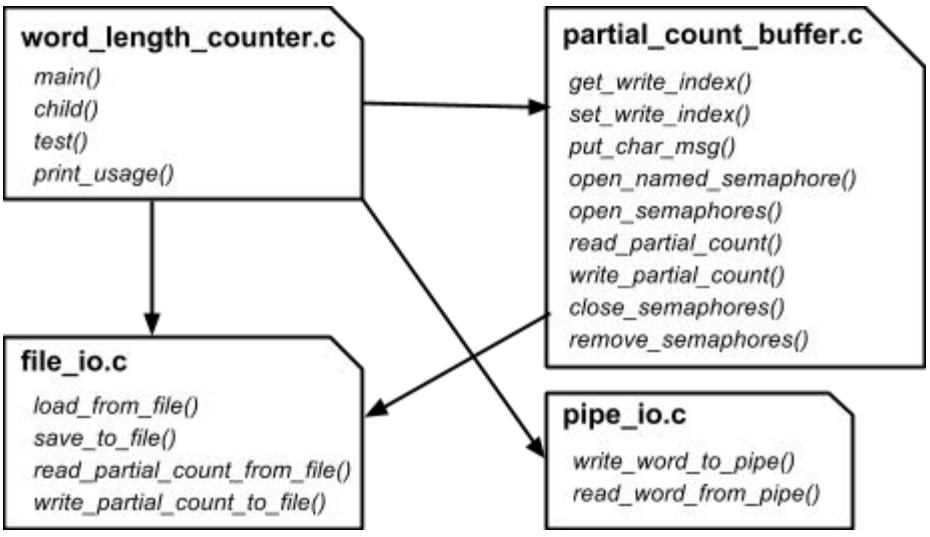
- il processo padre è l'unico consumatore
- i processi figlio sono i produttori
- trattandosi di processi e non di thread, il buffer circolare è virtualizzato usando dei file al posto delle celle di un array (es: l'item con indice 2 viene memorizzato nel file `buffer_2` invece che in `buffer[2]`), e viene utilizzato un file dedicato per salvare l'indice di scrittura
- i singoli item prodotti sono delle coppie `i,c` che rappresentano il fatto di aver trovato `c` parole di lunghezza `i`

Una volta inviato il proprio output parziale, i processi figlio terminano. Il processo padre aggrega questi output per produrre l'output finale, che viene infine confrontato con un output generato in maniera sequenziale per verificarne la correttezza.

Descrizione del codice dell'applicazione

Per ottimizzare la stesura e la leggibilità del codice, l'applicazione è organizzata in modo modulare, con due file di header `.h` e quattro file contenenti i sorgenti `.c` veri e propri:

common.h	Definizione delle costanti e delle macro (quelle per la gestione degli errori) in uso nei sorgenti `.c`
methods.h	Contiene i prototipi dei metodi definiti in ciascun modulo
word_length_counter.c	Modulo principale dove è definita la logica di alto livello sia del processo padre che di ciascun processo figlio
partial_count_buffer.c	Funzioni per inizializzazione, gestione e rilascio risorse del buffer circolare virtualizzato
file_io.c	Funzioni per operazioni di input/output su file
pipe_io.c	Funzioni per operazioni di input/output su pipe
Makefile	Compilazione automatizzata del codice



Architettura del codice sorgente. Una freccia indica che il modulo di origine utilizza uno o più metodi definiti nel modulo di destinazione.

Obiettivi

Vi viene richiesto di completare porzioni di codice - contrassegnate anche da blocchi di commenti nei rispettivi file - nei sorgenti C per implementare le seguenti operazioni:

(1) All'interno della funzione `main()` nel modulo `word_length_counter.c`, si gestisca

- il lancio dei processi figlio e l'apertura delle pipe necessarie (seguire con attenzione i suggerimenti riportati nel codice)
- la chiusura delle pipe una volta inviate tutte le parole del testo
- l'attesa del termine dei processi figlio ed il rilascio di tutte le risorse ancora allocate

(2) Nella funzione `child()` del modulo `word_length_counter.c`, si implementi il rilascio delle risorse ancora allocate.

(3) Il modulo `partial_count_buffer.c` contiene le funzioni da usare per la gestione del buffer circolare virtualizzato secondo il paradigma produttore/consumatore. Le variabili per i semafori sono già dichiarate, così come l'indice di lettura. Non viene dichiarata una variabile per l'indice di scrittura in quanto questo viene letto da/scritto su file. Si implementi l'inizializzazione dei semafori necessari (contenuta nelle due funzioni `open_named_semaphore()` e `open_semaphores()`), la gestione concorrente *race-free* delle operazioni di inserimento/rimozione di elementi nel/dal buffer, ed infine il rilascio delle risorse acquisite durante l'inizializzazione (contenuta nelle due funzioni `close_semaphores()` e `remove_semaphores()`).

(4) Il modulo `pipe_io.c` contiene due funzioni per la comunicazione via pipe.

Si completi la funzione `write_word_to_pipe()` affinché i byte della stringa contenuta nel buffer `msg` (incluso il carattere `\n` usato per segnalare la fine della stringa) vengano scritti sul descrittore di pipe `pipe_fd`. Una implementazione che assicuri che tutti i byte siano stati scritti verrà premiata in sede di valutazione.

Si completi poi la funzione `read_word_from_pipe()` affinché scriva nel buffer `word` una stringa di lunghezza fino a `MAX_WORD_LENGTH` byte leggendolo dal descrittore di pipe `pipe_fd`. La stringa da restituire non deve includere il carattere `\n` usato per segnalare la fine della stringa stessa, ma deve includere il carattere `\0` di terminazione. Nel caso in cui la pipe venga chiusa dal processo padre, la funzione deve restituire immediatamente `-1`, altrimenti il numero di byte realmente letti (escluso il carattere `\n`).

Testing

La compilazione è incrementale (ossia avviene un modulo per volta) ed automatizzata tramite `Makefile`, pertanto in assenza di errori di compilazione un binario `word_length_counter` verrà generato quando si esegue `make`. Per lanciare l'applicazione, digitare da terminale

```
./word_length_counter <input_filename> <N>
```

Dove `<input_filename>` è il path del file di testo da usare come input e `N` è il numero di processi figlio da lanciare. Nella cartella `input`, vengono forniti tre file di input di taglie diverse: `small_input.txt` (42 byte), `medium_input.txt` (1301 byte) e `big_input.txt` (23463 byte).

Durante l'esecuzione, vengono stampati a video dei messaggi di log che mostrano il progresso dell'applicazione, incluso l'esito del test per verificare la correttezza dell'output generato.

Altro

- in caso di cancellazione accidentale di uno o più file, nella cartella `backup/` è presente una copia di sicurezza della traccia di esame

- il file `dispensa.pdf` contiene una copia della dispensa *Primitive C per UNIX Sytem Programming* preparata dai tutor di questo corso

Raccomandazioni

Seguono alcune considerazioni sugli errori riscontrati più di frequente tra gli elaborati dei partecipanti, nonché delle raccomandazioni per un buon esito delle prove al calcolatore:

- dato un buffer `buf`, `sizeof(buf)` non restituirà il numero di byte presenti (i.e., scritti) attualmente nel buffer, bensì il numero di byte occupati da un puntatore (4 su IA32, 8 su x86_64) se allocato dinamicamente, o il numero di byte per esso allocati staticamente; si noti che nel caso in cui `buf` contenga una stringa NULL-terminated, è possibile utilizzare `strlen(buf)` per conoscerne l'effettiva lunghezza
- quando si deve invocare una funzione per la programmazione di sistema:
  - fare attenzione se i singoli parametri siano o meno dei puntatori
  - verificare dalla documentazione se può essere soggetta ad interruzioni, e in tal caso gestirle opportunamente
  - verificare il comportamento della funzione in caso di errori: inserire codice idoneo a trattare errori gestibili, utilizzare la macro opportuna per gli altri
  - per funzioni della libreria pthread, si vedano il capitolo 2 e l'appendice A.2 della dispensa di UNIX system programming
- quando viene acquisita una risorsa (socket aperta, area di memoria allocata, etc.), questa deve poi essere rilasciata in maniera opportuna
- utilizzare puntatori non inizializzati porta ragionevolmente un Segmentation fault!
- i cast di puntatori da e verso `void*` sono impliciti in C, renderli espliciti o meno è una scelta di natura puramente stilistica nella programmazione
- fare molta attenzione nell'aggiornamento dell'indice (o del puntatore) con cui si opera su un buffer: disallineamenti ±1 sono tra le cause più comuni di errori
- quando si vuole passare un puntatore ad una funzione, aggiungere al nome della funzione una coppia di parentesi tonde equivale invece ad invocarla!
- **i commenti nel codice contengono molte informazioni utili per lo svolgimento della prova, si consiglia quindi di tenerli in debita considerazione**

Regole Esame

- Domande ammesse
  Le domande possono riguardare solo la specifica dell'esame e la struttura di alto livello del codice, nessuna domanda può riguardare singole istruzioni.
- Oggetti vietati
  I seguenti oggetti non devono essere presenti sulla scrivania, né tantomeno usati: smartphone, telefonini, tablet, portatili, dispositivi di archiviazione USB, copie cartacee della dispensa, astucci e qualsiasi forma di libri ed appunti. **Chi verrà sorpreso ad usare uno di questi oggetti verrà automaticamente espulso dall'esame.**
- Azioni vietate
  È assolutamente vietato comunicare in qualsiasi modo con gli altri studenti. **Chi verrà sorpreso a comunicare con gli altri studenti per la prima volta verrà richiamato, la seconda volta verrà invece automaticamente espulso dall'esame**.