# Sistemi Operativi e Programmazione Distribuita
# Anno Accademico 2009-2010
# Specifiche progetto Unix

Giuseppe Lipari
g.lipari@sssup.it

19 maggio 2010

## 1 Distributed Memory

I candidati devono realizzare un sistema che realizzi un'astrazione di memoria distribuita con alcune funzionalità di base. Il sistema consiste di un certo numero di processi server, e di alcuni processi client.

### 1.1 Server

Il sistema da sviluppare consiste di uno o più *processi server* che forniscono l'astrazione di memoria globale distribuita. La memoria è suddivisa in blocchi di dimensione fissa pari a `DIMBLOCK` bytes (configurabile a tempo di compilazione). Ogni blocco di memoria è identificato da un indice numerico univoco (d'ora in poi chiamato `ID`).

Ognuno dei processi server fornisce alcuni blocchi predefiniti. Il numero dei server, e i blocchi serviti da ogni server sono contenuti in un file di configurazione che specifica appunto per ogni server la lista degli `ID` gestiti.

I server sono identificati da una coppia *indirizzo IP* e *porta*. Essi possono essere eseguiti sullo stesso nodo computazionale, o essere distribuiti su vari nodi. I server possono essere multi-thread.

### 1.2 Client

I processi client accedono alla memoria distribuita comunicando con i processi server. Per farlo, utilizzano delle funzioni di una libreria.

Scopo del progetto è quindi di fornire il codice dei processi server e il codice della libreria che i processi client possono utilizzare per svolgere la loro attività.

I processi client innanzitutto *mappano* i blocchi di memoria globale, memorizzati sui server, su indirizzi di memoria locale al processo. Definiamo come *copia globale* il blocco

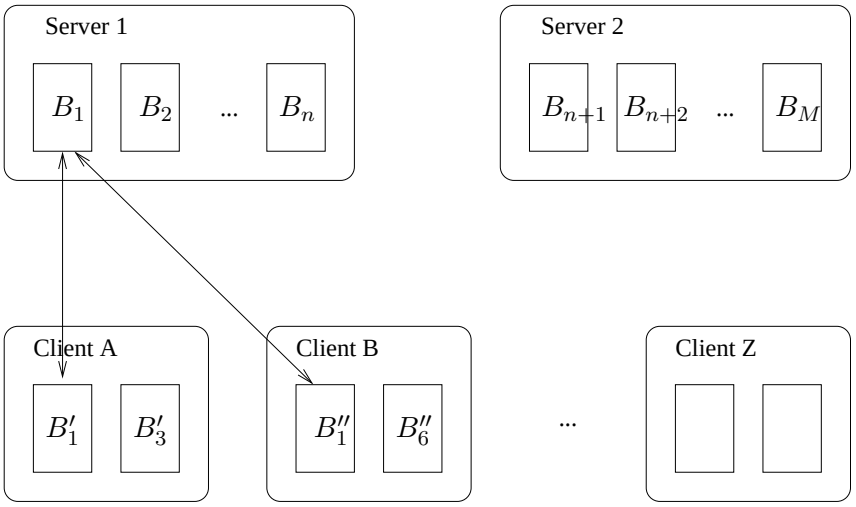


Figura 1: Schema generale client-server

di memoria contenuto sul server; e *copie locali* l'immagine di tali blocchi sulle memorie locali dei client.

Una copia locale può essere *valida* in un certo istante se, dal momento in cui è stata copiata dal server sulla memoria locale client, la copia globale non ha subito alcuna modifica; si dice *invalida* altrimenti.

Ad esempio, supponiamo che il blocco 2 abbia due copie locali sui client A e B. Ognuno dei due client può operare in lettura e scrittura sulle copie locali indipendentemente, e senza bisogno di sincronizzazione. Entrambe le copie rimangono valide, anche se il loro contenuto è diverso ed è diverso dal contenuto della copia globale.

Nel momento in cui il client A ricopia la sua copia locale sulla copia globale del server, la copia locale del client B diventa immediatamente invalida. Il client B potrà continuare a lavorare sulla sua copia locale senza accorgersi di niente; nel momento però in cui cercherà di copiare la sua copia locale sulla copia locale, l'operazione fallirà.

In altre parole, non deve essere possibile copiare una copia locale invalida sulla memoria globale.

### 1.3 Interfaccia

La libreria per i client deve mettere a disposizione le seguenti funzioni di interfaccia.

- void dm_init(char * config_file );
  - La funzione si occupa di inizializzare la libreria, caricando dal file config_file l'associazione tra `ID` dei blocchi e server. Inoltre, aprirà una serie di socket per

mettersi in connessione con i server, e (se lo si ritiene opportuno) uno o più thread di comunicazione con i suddetti server.

- int dm_block_map(int ID, void *address)
  - Questa funzione si occupa di stabilire l'associazione tra blocco di memoria e indirizzo di memoria locale. La funzione si collega al server corrispondente all'`ID` specificato, e richiede la copia del blocco a partire dall'indirizzo `address`. Contestualmente, il server memorizza il fatto che il blocco è utilizzato dal client in questione, mentre il client memorizza che il blocco è mappato sull'indirizzo.
  - Tipicamente, la funzione va chiamata una sola volta all'inizio del programma per ogni blocco che il client intende utilizzare.
  - La funzione torna con errore se l'associazione non può essere portata a termine (es. il server non risponde, oppure il blocco è già associato ad un altro indirizzo, ecc.)
- void dm_block_unmap(int ID)
  - Questa funzione notifica il server che gestisce il blocco `ID` che il client non è più interessato al blocco. Quindi, il server rimuove l'associazione client-blocco, e il client cancella l'associazione blocco-indirizzo. Tipicamente, questa funzione va chiamata all'uscita del client.
  - La funzione ritorna con errore se il blocco non può essere dissociato (es. il blocco non era associato ad alcun indirizzo, oppure il server non risponde, ecc.)
- int dm_block_update(int ID);
  - La funzione aggiorna il contenuto della copia locale del blocco con il contenuto globale sul server. Se la copia locale era invalida, dopo questa operazione torna valida.
  - Come ottimizzazione, si può prevedere che, se il contenuto è già aggiornato (cioè se la copia locale è ancora valida), non sia effettuata alcuna copia, riducendo il carico sulla rete;
  - La funzione fallisce se il blocco non era stato preventivamente associato ad alcun indirizzo locale da una dm_block_map();
- int dm_block_write(int ID)
  - Sincronizza il contenuto locale del blocco con il contenuto globale contenuto nel server di riferimento. In pratica, copia il contenuto locale contenuto all'indirizzo associato con il blocco sul server.

  - L'operazione può fallire (e in tal caso la funzione ritorna un errore) per vari motivi:
    * Il motivo più banale è che il blocco non sia stato preventivamente associato chiamando una dm_block_map().
    * Può succedere anche che dal momento della lettura del blocco fino alla sua scrittura, il blocco sia stato sovrascritto da un altro client. In qual caso, si dice che la copia locale del blocco è invalida. Quindi, se la copia locale è invalida, la funzione fallisce.
    * Infine, è possibile che il server non risponda entro un certo timeout, oppure sia caduta la connessione.
    * Il codice di errore ritornato dalla funzione deve indicare l'errore.

- int dm_block_wait(int ID)
  - La funzione controlla la validità della copia locale del blocco; se la copia è invalida, ritorna immediatamente, altrimenti, il thread che ha chiamato la funzione si blocca in attesa che la copia locale diventi invalida.
  - La funzione può fallire se il blocco non è stato preventivamente mappato su un indirizzo locale.

## 2 Programmi di test

Dopo aver sviluppato le libreria, i candidati devono sviluppare anche seguenti il seguente programma di test.

Il test consiste in un programma distribuito composto di due client, che comunicano tramite memoria condivisa implementata da 2 server distinti.

- Il primo client legge il contenuto di un file di testo, e lo carica sulla memoria globale. Quindi si mette a contare il numero di spazi nel file (carattere `' '`) e lo scrive sulla memoria globale in una opportuna locazione. Infine, attende che il secondo client abbia terminato il suo lavoro.
- Il secondo client attende che il primo abbia caricato il file, quindi conta il numero di parole nel testo, e lo scrive sulla memoria globale in una opportuna locazione. Quindi attende che il primo client abbia terminato.
- Infine entrambi i client scrivono sull'output il numero di spazi bianchi e il numero di parole del file.
- Assumere che la dimensione del blocco sia di 128 bytes. Assumere che ci siano due server che forniscono un numero uguale di blocchi ciascuno. Infine, assumere che il file di testo occupi blocchi di entrambi i server.

- Il test si intende corretto se entrambi i client stampano la stessa stringa finale.
- Per sincronizzare i due client, utilizzare esclusivamente funzioni della libreria.

# 3 Requisiti

Il codice sviluppato deve possedere i seguenti requisiti.

- Il codice va scritto utilizzando il linguaggio C/C++. E' consigliabile l'utilizzo del C++ rispetto al C per ragioni di modularità e leggibilità del codice.
- Il codice dei server e della libreria per i client va messo tutto in una directory `src/`.
- La compilazione avviene tramite *makefile*. Scrivere un makefile che compili il codice dei server, e successivamente anche la libreria per i client
- Il codice dei test va messo dentro una directory `test/`, anche questo va compilato con un makefile separato. Deve essere possibile lanciare il test con un semplice script di shell. Lo script deve lanciare il programma di test almeno 3 volte (anche sullo stesso file di testo), e stampare `'OK'` se il il programma di test è riuscito tutte le volte, e `'FAIL'` se il test fallisce almeno una volta.
- Per la documentazione del codice, utilizzare il tool `doxygen` (disponibile come pacchetto per le principali distribuzioni di Linux, oppure scaricabile al seguente indirizzo: `http://www.doxygen.org`). Inoltre, scrivere un file `README` che includa i nomi degli sviluppatori, e spieghi brevemente come compilare e come lanciare i test.

  Non è necessario commentare in dettaglio il codice sorgente (file `.c` o `.cpp`), in quanto il codice dovrebbe essere abbastanza chiaro anche senza bisogno di commenti dettagliati. Bisogna invece documentare i prototipi delle funzioni esterne e interne (presenti nei file `.h` o `.hpp`).

# 4 Modalità di consegna

Il progetto va svolto in gruppi di massimo 2 persone. Il progetto va spedito per e-mail almeno 2 giorni prima dell'esame all'indirizzo `g.lipari@sssup.it`, specificando nel subject:

```
[SISOP] Consegna progettino
```

e indicando chiaramente nel testo dell'e-email i nomi degli studenti del gruppo, e l'appello a cui si intende partecipare.