Esame di Programmazione Concorrente - appello del 17 luglio 2007

Nome: Cognome: Matricola:

Durata: 2 ore e 30 minuti. Non si può consultare documentazione. Usare solo ed esclusivamente i fogli che sono stati consegnati.

Quiz

Una sola risposta esatta a domanda. Non sono ammesse correzioni.
+1 risposta esatta; 0 nessuna risposta o risposta multipla; -1 risposta sbagliata

• Le tecniche per la scrittura di codice thread-safe tramite oggetti immutabili è particolarmente indicata:

- ◦ in presenza di operazioni di aggiornamento frequenti
- ◦ quando l'insieme di tutti i possibili stati distinti posseduti da oggetti del tipo considerato è finito e di cardinalità contenuta
- ◦ in presenza di oggetti con uno stato di dimensioni rilevanti
- ◦ se si vuole contenere il numero totale di oggetti creati
- ◦ nessuna delle precedenti

• A quale delle seguenti affermazioni è equivalente l'assunzione di progresso finito?

- ◦ bisogna evitare ogni forma di attesa attiva
- ◦ le sequenze di interleaving probabilisticamente molto rare non possono compromettere la correttezza di una soluzione concorrente
- ◦ non è possibile fare ipotesi sulla velocità relativa dei processori virtuali che avanzano i flussi di esecuzione
- ◦ è possibile costruire soluzioni concorrenti corrette basandosi sull'uso di pause di lunghezza predeterminata
- ◦ nessuna delle precedenti

• Le regioni critiche condizionali sono un costrutto di alto livello per la specifica di codice concorrente che:

- ◦ favorisce la coesione del codice relativo alla gestione di una risorsa condivisa
- ◦ consente di mettere in comunicazione diretta ed esplicita i flussi di esecuzione che sono interessati ad un evento con quelli che lo fanno verificare
- ◦ è stato pensato per risolvere solo problemi di cooperazione
- ◦ è stato pensato per evitare di rivalutare le condizioni di sincronizzazione più di una sola volta
- ◦ nessuna delle precedenti

• Con riferimento all'Algoritmo del Banchiere, si considerino tre clienti A, B, C con un fido pari rispettivamente a 8, 2, 5 ed un prestito attuale rispettivamente pari a 4, 1, 3. Se la cassa corrente del banchiere è pari a 1:

- lo stato corrente è sicuro
- lo stato corrente non è sicuro perché le richieste di A e C potrebbero non essere mai soddisfatte
- lo stato corrente non è sicuro perché B potrebbe non chiedere mai l'unità che gli manca per raggiungere il tetto del proprio fido
- lo stato corrente non è sicuro perché le potenziali richieste di A e C potrebbero essere soddisfatte solo dopo che B raggiunge il proprio tetto
- nessuna delle precedenti

• Con riferimento alle primitive wait e signal di Hoare per i monitor, qual'è lo svantaggio della semantica *signal_and_wait*:

- il flusso di esecuzione che effettua la signal è costretto ad attendere la fine della entry-procedure corrente prima di rilasciare l'uso del monitor
- il flusso di esecuzione che effettua la wait deve fermarsi in attesa di ottenere l'uso esclusivo del monitor
- il flusso di esecuzione che effettua la signal potrebbe invalidare l'evento segnalato prima di rilasciare l'uso del monitor
- tende a far aumentare il numero dei cambi di contesto
- nessuna delle precedenti

Esercizio Java

11 punti Si è deciso un intervento teso a migliorare le prestazioni di una libreria java mono-thread per il calcolo numerico. In particolare si è deciso di migrare ad una piattaforma multiprocessore dotata di N CPU e di riscrivere in versione multi-thread un metodo `BinaryTreeSum.computeSum(Node root)` molto oneroso. Tale metodo permette di calcolare la somma dei valori ottenuti applicando un metodo statico molto oneroso `Computer.compute(int nodeValue)` su tutti i valori interi contenuti nei nodi di un albero binario di enormi dimensioni.

Scrivere una classe che realizzi l'intervento implementando l'unico metodo di questa interfaccia:

```
public interface BinaryTreeSum {
  public int computeSum(Node root);
}
```

dove `Node` è una interfaccia utilizzata dalla libreria per modellare i nodi di un albero binario a partire dalla radice che `computeSum` riceve come parametro:

```
public interface Node {
  Node getSx(); //null se non esiste figlio sinistro
  Node getDx(); //null se non esiste figlio destro
  int getValue();
}
```

Notare che i valori interi su cui applicare `Computer.compute(int nodeValue)` sono ottenibili invocando `Node.getValue()` e che si suppone che tutti i nodi dell'albero contengano valori interi distinti.

Si richiede esplicitamente di evitare: (*i*) ogni forma di attesa attiva ed ogni forma di interferenza (*ii*) l'elaborazione multipla degli stessi nodi dell'albero da parte di thread diversi (*iii*) thread che sopravvivano all'esecuzione del metodo `BinaryTreeSum.computeSum(Node root)` pur essendo stati creati al suo interno.

```
import java.util.concurrent.*;
public class Executors {
  static ExecutorService newFixedThreadPool(int n);
  static ExecutorService newCachedThreadPool();
  static ExecutorService newSingleThreadExecutor()
}
public interface Executor {
  void execute(Runnable command);
}
public interface Callable {
  V call() throws Exception;
}
public interface Future<V> {
  V get();
  boolean isDone();
}
```

```
public interface ExecutorService
                implements Executor {
  Future<?> submit(Runnable task);
  Future<T> submit(Callable<T> task);
  void execute(Runnable command);
  void shutdown();
}

public class FutureTask<V>
            implements Runnable, Future<V>{
  FutureTask(Callable<V> callable);

  V get();
  boolean isDone();
}
```

Esercizio C

11 punti Una grande compagnia di intermediazione riesce a soddisfare le richieste di tre tipologie di clienti: venditori, compratori e trasportatori. Questi sono interessati a venire in contatto tra di loro per realizzare lo scambio di container di merci aventi tutti esattamente lo stesso contenuto. Affinché una compravendita abbia luogo è infatti necessario l'accordo comune tra un compratore che compra il container, il venditore che lo vende ed un trasportatore che si occupa del trasporto.

Il contratto che la compagnia di intermediazione fa sottoscrivere ai propri clienti prevede almeno questi punti:

- ciascun utente appartiene ad una ed una sola tipologia: *venditore*, *compratore* o *trasportatore*
- è garantita la gestione *fair* dei clienti, ovvero le loro richieste, raggruppate per tipologia, sono gestite secondo l'ordine di arrivo
- la compagnia ha lo scopo di favorire gli scambi contenendo il tempo di attesa dei clienti e pertanto dispone lo scambio e definisce le parti non appena è a conoscenza di almeno un utente per ciascuna delle tre tipologie di clienti
- la compagnia si impegna a far conoscere a ciascun interessato gli identificativi delle altre controparti e del container scambiato. La compagnia tuttavia precisa che le notifiche avverrano solo al momento della definizione completa e vincolante di tutte le parti interessate, e mai prima
- al momento della definizione dello scambio, la compagnia informa il compratore dell'identificativo del venditore, del trasportatore, e del container; il trasportatore viene informato dell'identificativo del compratore, del venditore e del container, e similarmente il venditore viene informato dell'identificativo del compratore, del trasportatore e del container
- i clienti non possono fare alcuna scelta sulle controparti ma devono accettare quelle stabilite insindacabilmente dalla compagnia
- i clienti non possono scegliere i container, i quali, sebbene muniti di un codice identificativo, sono tutti perfettamente identici ed indistinguibili per contenuto e caratteristiche e saranno venduti/acquistati/trasportati il più velocemente possibile compatibilmente con l'ordine di arrivo delle richieste e della loro tipologia
- le richieste sono di uno ed un solo container, se qualcuno è interessato a comprare/vendere/trasportare più container, deve fare molteplici richieste unitarie ed indipendenti tra loro

- i clienti sono tenuti a specificare un periodo di validità per ciascuna delle proprie richieste (di acquisto/vendita/trasporto). Allo scadere di tale periodo, l'intermediario considera decaduta la validità della richiesta non ancora esaudita senza dargli alcun seguito. Al contrario, durante il periodo di validità, la richiesta risulta vincolante per il cliente che l'ha inoltrata, e non può essere revocata
- il periodo di validità è specificato contestualmente a ciascuna richiesta, e può variare da richiesta a richiesta anche nel caso di richieste provenienti dal medesimo cliente
- l'istante al quale una determinata richiesta risulta scaduta viene stabilito dalla compagnia a suo insindacabile giudizio ma sulla base del periodo di validità indicato dal cliente contestualmente all'avanzamento delle richiesta
- la compagnia effettua le notifiche ai suoi clienti di tutte le richieste scadute che non è stata in grado di esaudire

Si vuole realizzare un server prototipale per la gestione della compagnia di intermediazione. Il servizio consente agli utenti, dotati di appositi client, di avanzare una richiesta con il relativo periodo di validità e di ricevere, al momento della definizione vincolante dello scambio, i codici identificativi delle due controparti e del container scambiato.

Impostare una soluzione concorrente per realizzare il servizio di intermediazione con il massimo grado di parallelismo possibile ed evitando ogni forma di attesa attiva o di interferenza come segue:
a) 3 punti su 11
Descrivere quanti flussi di esecuzione sono creati e, per ciascun flusso di esecuzione, le sue interazioni con gli altri flussi. Descrivere sinteticamente le strutture dati utilizzate a supporto delle loro attività ed indicare esplicitamente gli strumenti implementativi e gli eventuali meccanismi IPC utilizzati per realizzare tali flussi e loro interazioni (processi/thread; mutex/semafori/var.cond...; pipe/shm...).
b) 8 punti su 11
Implementare un'applicazione client/server scritta in C sotto Linux. Dettagliare il codice del solo *server di intermediazione* trascurando il codice dei vari tipi di client, ma chiarire il protocollo (sintassi e semantica) alla base dei messaggi scambiati con i vari client in tutte le fasi dell'elaborazione.

È possibile fare le seguenti ipotesi: il server di intermediazione è raggiungibile dai client all'indirizzo `SI_ADDRESS` con porte rispettivamente `PORT_C`, `PORT_V` e `PORT_T` per client compratori, venditori e trasportatori; gli utenti del servizio sono noti a priori e, per ciascuna tipologia, sono univocamente identificabili con un intero progressivo già noto ai client stessi. Il codice identificativo dei container è un intero progressivo distinto dai primi ed univocamente stabilito dal server. I codici identificativi e la durata del periodo di validità di una richiesta sono interi positivi agevolmente rappresentabili con il tipo primitivo `int`; è già disponibile un libreria `pc.h` per la comunicazione client/server di tali interi:

```
#include <pc.h>
int sendInt(int socket, int data); // spedisce intero sulla socket
int receiveInt(int socket);        // riceve intero dalla socket
```

Si osservi che tutte le funzioni sono bloccanti, ma sono state implementate in modo da garantire il ritorno e la restituzione del controllo al flusso di esecuzione invocante in un tempo finito anche in presenza di errori (in tal caso restituiscono `-1`).

Segnature Si riportano di seguito le segnature di alcune chiamate di sistema che potrebbero risultare utili per lo svolgimento dell'esercizio. Si noti che si dovrà scegliere quali utilizzare ed eventualmente aggiungerne altre simili.

Socket

```
#include <netdb.h>
struct hostent *gethostbyname(const char *name);
struct hostent {
    char    *h_name;        /* official name of host */
    char    **h_aliases;    /* alias list */
    int     h_addrtype;     /* host address type */
    int     h_length;       /* length of address */
    char    **h_addr_list;  /* list of addresses */
}
#define h_addr  h_addr_list[0]  /* for backward compatibility */

#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
int send(int s, const  void *msg, int len, unsigned int flags);
int sendto(int s, const void *msg, int len, unsigned int flags, const struct sockaddr *to, int tolen);
int recv(int s, void *buf, int len, unsigned int flags);
int recvfrom(int s, void *buf, int len, unsigned int flags, struct sockaddr *from, int *fromlen);

int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
int listen(int s, int backlog);
int accept(int s, struct sockaddr *addr, int *addrlen);

#include <netinet/in.h>
#include <arpa/inet.h>
char *inet_ntoa(struct in_addr in);

struct sockaddr {
    unsigned short sa_family; // address family, AF_xxx (AF_INET)
    char sa_data[14];         // 14 bytes of protocol address
};
struct sockaddr_in {
    short int sin_family;             // Address family
    unsigned short int sin_port;      // Port number
    struct in_ in_addr addr sin_addr; // Internet address
    unsigned char sin_zero[8];        // Same size as struct sockaddr
};
struct in_addr {
    unsigned long s_addr; // 32-bit long, 4 byte
};

unsigned long int htonl(unsigned long int hostlong); //htons(), htohl(), ntohs() sono analoghe
```

Segnali

```
#include <sys/types.h>
#include <signal.h>
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
int kill(pid_t pid, int sig); //sig: SIGUSR1, SIGUSR2, SIGALRM, SIGINT...
unsigned int alarm(unsigned int seconds);
```

Segmenti di Memoria Condivisa

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, int size, int shmflg);
key_t ftok(const char *pathname, int proj_id);
int shmctl(int shmid, int cmd, struct shmid_ds *buf); // cmd: IPC_STAT, IPC_SET, IPC_RMID
void *shmat(int shmid, const void *shmaddr, int shmflg);
int shmdt(const void *shmaddr);
```

Semafori

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semop(int semid, struct sembuf *sops, unsigned nsops);
int semget(key_t key, int nsems, int semflg);
```

```
int semctl(int semid, int semnum, int cmd, ...); // cmd: SET_VAL, IPC_RMID...
struct sembuf {...
   unsigned short sem_num;  /* semaphore number */
   short sem_op;            /* semaphore operation */
   short sem_flg;           /* operation flags: IPC_NOWAIT, SEM_UNDO */
...}
union semun {
  int val;                  /* value for SETVAL */
  struct semid_ds *buf;     /* buffer for IPC_STAT, IPC_SET */
  unsigned short *array;    /* array for GETALL, SETALL */
  /* Linux specific part: */
  struct seminfo *__buf;    /* buffer for IPC_INFO */
};
```

Thread

```
#include <pthread.h>
int pthread_create(pthread_t *thread, pthread_attr_t *attr, void * (*start_routine)(void *), void *arg);
void pthread_exit(void *retval);
int pthread_join(pthread_t th, void **thread_return);

pthread_mutex_t fastmutex = PTHREAD_MUTEX_INITIALIZER;
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutex-attr_t mutexattr);
int pthread_mutex_lock(pthread_mutex_t *mutex));
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
int pthread_mutex_destroy(pthread_mutex_t *mutex);

pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
int pthread_cond_init(pthread_cond_t *cond, pthread_condattr_t *cond_attr);
int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
int pthread_cond_timedwait(pthread_cond_t *cond, pthread_mutex_t *mutex, const struct timespec *abstime);
int pthread_cond_destroy(pthread_cond_t *cond);
```

```
#include <semaphore.h>
int sem_init(sem_t *sem, int pshared,
             unsigned int value);
int sem_wait(sem_t * sem);
int sem_trywait(sem_t * sem);
int sem_post(sem_t * sem);
int sem_getvalue(sem_t * sem, int * sval);
int sem_destroy(sem_t * sem);
```

Pipe

```
#include <unistd.h>
int pipe(int filedes[2]);
int dup(int oldfd);
int dup2(int oldfd, int newfd)

#include <stdio.h>
FILE *popen(const char *command, const char *type);
int pclose(FILE *stream);
```

I/O , Miscellanea

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
int creat(const char *pathname, mode_t mode);
int close(int fd);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);

#include <string.h>
void *memset(void *s, int c, size_t n);
```