Sistemi di Calcolo 2

Esercitazione Finale

Esercizio 1 - Realizzazione di un server multi-thread con comunicazione su socket.

Un server espone N-1 risorse identificate da un numero compreso tra 1 e N-1. I client possono connettersi al server tramite socket e richiedere l'utilizzo di una risorsa inviando un numero tra 1 e N-1. Il server mantiene un array shared_counters con N contatori, dove il contatore i-esimo tiene traccia del numero di richieste ricevute per la risorsa i-esima.

Ogni client viene gestito con un thread (su cui non è previsto fare join) che esegue la funzione connection_handler(). Questo thread riceve comandi dal client ed invoca la funzione process_resource(), che incrementa il contatore di shared_counters relativo alla risorsa richiesta. L'accesso al contatore i-esimo di shared_counters è protetto dal semaforo i-esimo in semaphores. Es: se il client invia al server "1", viene incrementato il contatore in posizione 1. Qualsiasi altro comando viene considerato come "0", ed in tal caso è il contatore in posizione 0 ad essere incrementato. L'unica eccezione è il comando "QUIT", che fa terminare il thread.

Obiettivi:

1. Gestione semafori
    - Inizializzazione semafori
    - Gestione sezione critica nella funzione process_resource()
2. Gestione multi-thread
    - Creazione thread di gestione connessione client
    - Rilascio risorse allocate
3. Gestione scambio messaggi su socket
    - Invio/ricezione messaggi su socket con gestione interruzioni
    - Chiusura descrittori

Domanda 2

Spiegare che cosa sono le operazioni di "compare-and-swap" e "exchange", come possono essere utilizzate per sincronizzare processi per l'accesso ad una sezione critica, possibilmente tramite pseudo-codice. Elencare vantaggi e svantaggi di queste primitive.

Domanda 3

Considerate il seguente algoritmo di Dijkstra per l'accesso in mutua esclusione ad una sezione critica:

```
 /* global storage */

boolean interested[N]  = {false, ..., false}
boolean passed[N]  = {false, ..., false}
int k = <any>   // k ∈ {0, 1, ..., N-1}

/* local info */
int i = <entity ID>  // i ∈ {0, 1, ..., N-1}
while (True) {
        < NCS >
        1.  interested[i] = true
        2.  while (k != i) {
        3.        passed[i] = false
        4.        if ( interested[k] ) then k = i
             }
        5.  passed[i] = true
        6.  for j in 1 ... N except i do
        7.        if ( passed[j] ) then goto 2
        8.  <Critical Section>
        9.  passed[i] = false; interested[i] = false

}
```

Sostituendo nella riga 4 "**if not** interested[k] **then** k=i" con "**if interested**[k] **then** k=i" quali sono le problematiche che si creano legate alle proprietà di correttezza di Mutua_Esclusione e no deadlock? Se e quali vengono violate? Motivare la risposta.