SISTEMI OPERATIVI IEL/IDT
prova scritta preliminare del 16.04.2003

Nome: _____ Cognome: _____

Sia dato il seguente problema:

> Un processo server S eroga un determinato servizio. Il processo S gestisce una richiesta alla volta, leggendo le informazioni necessarie da una coda di lunghezza massima pari a N. Se non ci sono richieste, S si sospende. I processi client C_i (per depositare le informazioni che caratterizzano la richiesta) ed il processo server (per leggere le richieste) devono accedere alla coda in maniera sincronizzata. Nel caso in cui la coda abbia già raggiunto la lunghezza massima, il client non si può accodare e deve procedere nella sua esecuzione.

Il candidato illustri anzitutto una soluzione al problema di sincronizzazione che faccia uso di semafori; successivamente discuta le modifiche che devono essere apportate (se necessarie) per il caso in cui siano disponibili piu' processi server.

Traccia:

Server S	*Client C_i*

While(true) {
  Richiesta r;
 	
  ???
 	
  erogaServizio(r);
 	
  ???
 	
}
	{
	  Richiesta r = new Richiesta();
	  /* imposta i dati che descrivono la richiesta */
	
	  ???
	
	  richiediServizio();
	
	  ???
	
	}

-Il tipo dati Richiesta è una struttura contenente le informazioni necessarie per l'erogazione del servizio;
-la procedura *erogaServizio(Richiesta r)* del processo server provvede all'erogazione del servizio elaborando la specifica richiesta *r*;
-la procedura *richiediServizio()* richiede al processo server l'erogazione del servizio secondo quanto specificato dalla richiesta inserita nella coda.

Possibile soluzione

Variabili condivise e semafori

```
//      semaforo binario di mutua esclusione
Semaphore mutex = new Semaphore(1);
//      semaforo binario usato per segnalare che ci sono richieste pendenti
Semaphore richiestePendenti = new Semaphore(0);
//      semaforo usato per indicare che il server è pronto a erogare il servizio richiesto
Semaphore serverPronto = new Semaphore(0);
//      lunghezza massima della lista
int lunghezza = N;
//      numero di richieste dei clienti in coda
int clientiInAttesa = 0;
//      coda delle richieste
MemoriaCircolare coda = new MemoriaCircolare( lunghezza );
```

Server, S

```
while ( true ) {
    richiestePendenti.wait();       //      si sospende se non ci sono richieste da elaborare
    mutex.wait();                   //      accede alla sezione critica
    Richiesta r = coda.estrai();    //      legge richiesta dalla coda
    clientiInAttesa- -  ;           //      decrementa numero processi in attesa
    serverPronto.signal();          //      segnale che è pronto per gestire nuova richiesta
    mutex.signal();                 //      rilascia la sezione critica
    service( r );                   //      serve il processo fuori dalla sezione critica
}
```

Client, C_i

```
…
    Richiesta r = new Richiesta();
    //    imposta i dati per la richiesta
    mutex.wait();                   //      accede alla sezione critica
    if ( clientiInAttesa < N) {     //      se la coda non è piena
        clientiInAttesa++;          //              aggiorna lunghezza coda
        coda.inserisci( r );        //              inserisce dati in coda
        richiestePendenti.signal(); //              segnala presenza richiesta al server
        mutex.signal();             //              uscita dalla sezione critica
        serverPronto.wait();        //              attende di poter colloquiare col server
        requestService();
    } else {
        mutex.signal();             //      se la lista è piena, il processo non riprova
    }
…
```