SIMULAZIONE DI UN SISTEMA A CODA M/M/Y/Y

Progetto di un Sistema a Coda M/M/Y/Y per il corso di Reti di Telecomunicazioni tenuto dal professor Andrea Conti, Corso di Laurea Triennale in Informatica, A.A. 2019 – 2020.
Università di Ferrara.

Mebri del gruppo: Martina Tenani, Alberto Paparella.

La simulazione è stata realizzata nel linguaggio di programmazione C.
Il progetto e' stato sviluppato in più file per migliorarne la leggibilità in fase di lavoro, di debugging e per la lettura di terzi, nonché per assicurarne un miglior riutilizzo.

INDICE:

OVERVIEW e SCELTE IMPLEMENTATIVE

E' stato scelto di utilizzare come struttura dati per l'implementazione della coda una Lista Collegata, ottima per la simulazione di una coda virtualmente "infinita", seppur non necessaria nel sistema in questione essendo, almeno teoricamente, sempre vuota ma con occhio al riutilizzo per la simulazione ad altri sistemi. Sfruttando l'allocazione dinamica della memoria questa non necessita infatti di una dichiarazione a priori della capacita'.
Le operazioni svolte sulla struttura di coda sono quelle classiche di enqueue (inserimento in coda) e dequeue (eliminazione in testa).
Oltre a queste, sempre in funzione della simulazione e del debugging, vengono aggiunte una funzione per la stampa del numero di elementi presenti nella coda o eventualmente degli id degli stessi pacchetti in coda.
Un miglioramento potrebbe essere tener conto dell'indirizzo dell'ultimo elemento in coda, la cosiddetta "tail" della coda. Viene comunque inserita una funzione per il calcolo di questo indirizzo, seppur non sfruttata nella simulazione.

GENERAZIONE DEGLI ARRIVI

La generazione dei tempi di interarrivo e di intersevizio (numeri pseudo-casuali) sfrutta la tecnica di trasformazione inversa, ricavando dalla formula di densita' di probabilita' cumulata di interarrivo dei processi markoviani

$$F_{\tau_i}(t) = 1 - e^{-f_{\tau_i}(0)t}$$

la seguente formula:

$$t = -\left(\frac{1}{\lambda}\right)\ln(1-u)$$

dove per log si intende il logaritmo naturale e per u una variabile aleatoria distribuita uniformemente fra 0 e 1. Ovviamente, per il calcolo dei tempi di interservizio basta invertire lambda con mu. A questo calcolo viene infine sommato 1 per evitare di generare il valore 0 (testato sperimentalmente, il cambiamento che questo comporta e' trascurabile).

DESCRIZIONE DEI FILE

`pacchetto.h`

Nel file pacchetto.h viene definita una struttura dati Pacchetto il cui unico campo non e' altro che un intero “id” contenente un numero identificativo da 0 a del pacchetto.

`servitore.h`

Nel file servitore.h viene definita una struttura dati Servitore di campi “pacchetto”, di tipo predefinito Pacchetto, utile per conoscere l'id del pacchetto che sta venendo servito; un campo “Occupato” che funge da valore booleano per conoscere se il servitore sta servendo un pacchetto (1) o meno (0), e un campo “tServizio” di tipo intero che indica fra quanti cicli il servitore tornera' ad essere libero.

`coda.h`

Nel file servitore.h viene definita la struttura dati della Coda sfruttando le conoscenze sulle Liste Collegate e la loro implementazione in C.
“Coda” è definita come un puntatore a “Nodo”, quest'ultimo a sua volta definito come una struttura contenente un “pacchetto” (di tipo predefinito Pacchetto) e un campo “next”, puntatore al nodo successivo (l'indirizzo a cui si trova l'elemento successivo della coda).
Sono inoltre definite le interfacce per le funzioni che operano sulla coda visibili al cliente, in particolare:

- `void nuovaCoda(Coda* pc)`: prende come argomento un puntatore a coda e serve ad inizializzare la coda.
- `void enqueue(Coda* pc, Pacchetto p)`: prende come argomenti un puntatore a coda e un pacchetto ed effettua l'inserimento in coda.
- `void dequeue(Coda* pc)`: prende come argomento un puntatore a coda ed effettua l'eliminazione del pacchetto dalla testa della Coda.
- `void getHead(Coda c)`: prende come argomento una coda e ne stampa il pacchetto in testa.
- `void getTail(Coda c)`: prende come argomento una coda e ne stampa l'ultimo pacchetto entrato nella coda.
- `int getStato(Coda c)`: prende come argomento una coda e restituisce il numero di elementi presenti nella coda, di tipo intero.
- `void stampaCoda(Coda c)`: prende come argomento una coda e stampa gli id dei pacchetti in coda, nell'ordine dal prossimo che verra' servito fino all'ultimo pacchetto entrato in coda (a livello grafico a specchio rispetto ad una rappresentazione canone di coda).

`coda.c`

Il file coda.c contiene l'implementazione delle funzioni che operano sulla coda. Si evidenzia la presenza di due funzioni, insTesta e ricerca, nascoste al cliente in quanto vengono solo sfruttate nell'implementazione di altre funzioni (l'operazione di enqueue).
In particolare:

- `void insTesta(Coda* pc, Pacchetto pacchetto)` che prende come argomenti un puntatore a coda e un pacchetto, ed effettua l'inserimento del pacchetto nel punto della coda specificato dal puntatore “pc”.
- `Coda* ricerca(Coda* pc, pacchetto p)` che prende come argomenti un puntatore a coda e un pacchetto e scorre gli elementi della coda sino a che non ne

trova uno che punti a NULL, nel nostro caso è l'ultimo elemento, quello attualmente in "coda alla coda".

Funzionamento di enqueue:
l'operazione di enqueue, o di inserimento in coda, sfrutta la ricerca sulla coda dell'ultimo elemento della coda (quello che nel campo next contiene un puntatore a NULL), crea un nuovo Nodo ausiliario contenente l'id del nuovo pacchetto, un campo next assegnato al valore NULL. L'indirizzo del pacchetto ausiliario verra' sostituito nel campo next dell'ultimo pacchetto della coda.

`poisgen.h`
Il file poisgen.h contiene semplicemente il prototipo della funzione per la generazione di numeri pseudo-casuali da una legge di distribuzione poissoniana con parametro x.

- `int Poisson(double x)`

`poisgen.c`
Il file poisgen.c contiene l'implementazione della funzione per la generazione di numeri pseudo-casuali che seguono la legge poissoniana (sfruttati come tempi di interarrivo e interservizio).

`main.c`
Il cliente della simulazione.

RISULTATI SPERIMENTALI:

Misurazione del parametro E{k} (stato del sistema) al variare del numero di servitori.
Le misurazioni sono state effettuate per valori di λ che variano tra 0.001 e 0.2, con un incremento di 0.001. Il tasso di morte μ è rimasto fisso a 0.01 per tutta la durata dell'esperimento.