Università di Bergamo

Reti di Telecomunicazione

Esempio di simulatore di un sistema a coda in C++

Fabio Martignon

Struttura della classi

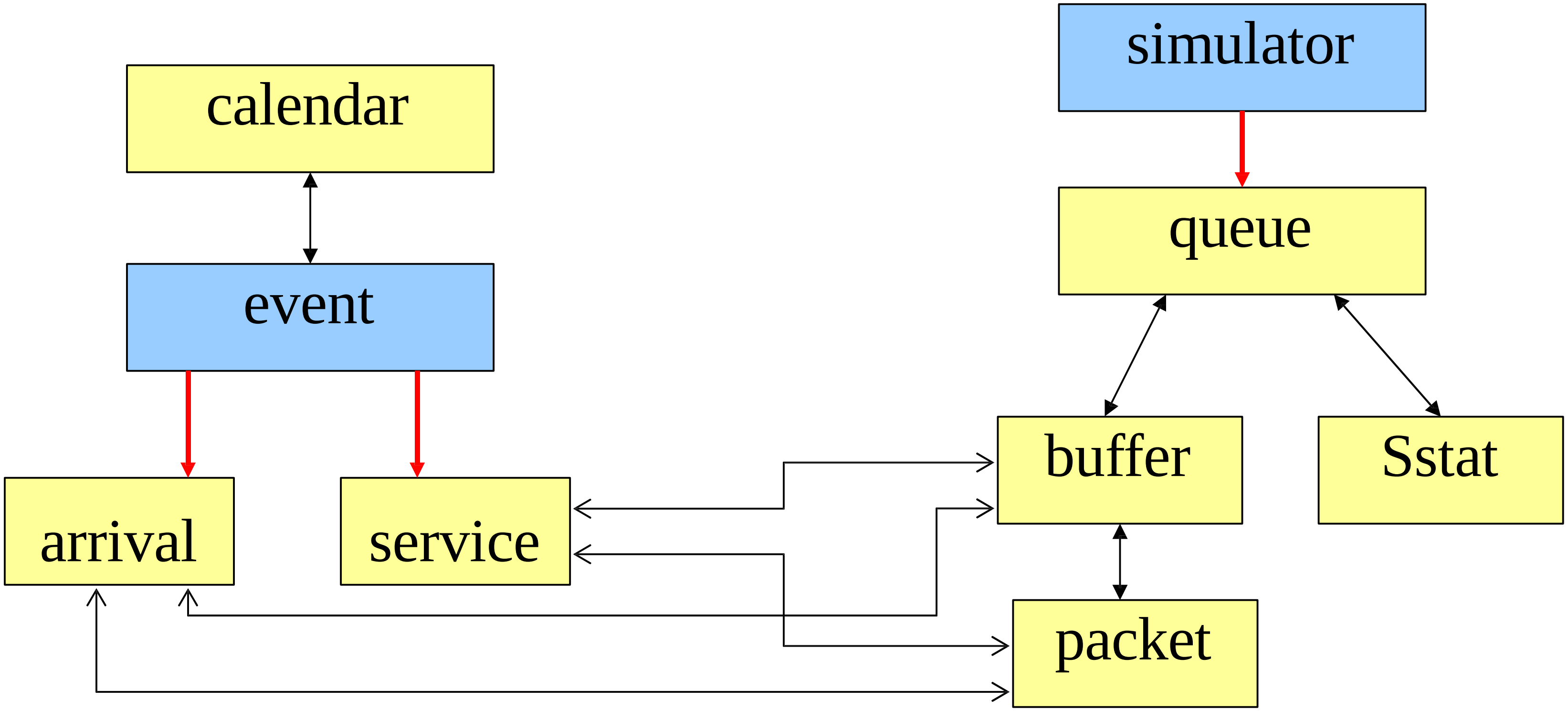
🟦 classi astratte

🟨 classi reali

↕ interazione funzionale

↕ relazione diretta

↓ rel. di ereditarietà



Routine principale: Main.c

```
/* -*- C++ -*- */
#include <stdio.h>
#include "global.h"
#include "queue.h"
#include "simulator.h"


main(int argc,char *argv[])
{

simulator *eval;

printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
printf("*********************************************************\n\n");
printf("               G/G/1 QUEUE SIMULATION PROGRAM\n\n");
printf("*********************************************************\n\n");

      eval=new queue(argc,argv);
      eval->init();
      eval->run();
      eval->results();
```

Una classe astratta per un simulatore ad eventi discreti

```
/* -*- C++ -*- */
/********************************************************
            SIMULATOR.H
      Abstract class for simulation program

*******************************************************/

#ifndef _SIMULATOR_H
#define _SIMULATOR_H

#include <stdio.h>

class simulator{
protected:
      FILE *fptrc;
      FILE *fpout;
      void read_args(int argc,char* argv[]);
      virtual void input(void) = 0;
      virtual void print_trace(int i) = 0;
public:
      simulator(int argc,char* argv[]);
      virtual ~simulator(void) {;}
      virtual void init(void) = 0;
      virtual void run(void) = 0;
      virtual void results(void) = 0;
};
```

Una classe astratta per un simulatore ad eventi discreti

```
/* -*- C++ -*- */
/********************************************************
            SIMULATOR.C
      Abstract class for simulation program

********************************************************/

#include <stdio.h>
   (…)
simulator::simulator(int argc,char* argv[])
{
  fptrc=NULL;
  fpout=NULL;
  read_args(argc,argv);
}
```

legge gli argomenti passati dall’OS
>> usage: -o output_file -t trace_file

```
void simulator::print_trace(int i)
{
       fprintf(fptrc, "********************************************\n");
      fprintf(fptrc, "                 TRACE RUN %d                \n", i);
      fprintf(fptrc, "********************************************\n\n");
      fflush(fptrc);

}
```

La classe queue

```
/*******************************************************
                 G/G/1 QUEUE SIMULATOR
*******************************************************/
#ifndef _QUEUE_H
#define _QUEUE_H

#include "simulator.h"
#include "calendar.h"
#include "event.h"
#include "buffer.h"
#include "packet.h"
#include "stat.h"

class queue: public simulator{

      virtual void input(void);
      buffer* buf;            // queue buffer
      int   traffic_model;
      double      load;
      int   service_model;
      // counters
      double      packets;
      double      tot_delay;
      // statistics
      Sstat*      delay;
public:


      virtual void init(void);
      virtual void run(void);

      virtual void clear_counters(void);
      virtual void clear_stats(void);
      virtual void update_stats(void);
      virtual void print_trace(int Run);
      virtual void results(void);
```

parametri d'ingresso:

- traffic_model:
  flag che identifica il processo degli arrivi (solo Poisson implementato)
- load:
  traffico offerto in erlang
- service_model:
  flag che identifica il processo dei servizi

contatori statistici

variabile statistica (un campione per run)

La classe queue:
costruttore e distruttore

```
queue::queue(int argc,char *argv[]): simulator(argc,argv)
{
cal=new calendar();
buf = new buffer();
delay=new Sstat();
}

queue::~queue()
{
delete delay;
delete cal;
delete buf;
```

NOTA: cal è una variabile globale instanziata all’interno del modulo queue.c e dichiarata extern dagli altri moduli che ne fanno uso (event.c)

La classe queue:
lettura dei parametri d’ingresso

```
void queue::input(){
printf("MODEL PARAMETERS:\n\n");
      printf("\n Arrivals model:\n");
      printf("1 - Poisson:>\n");
      traffic_model=read_int("",1,1,1);
      load=read_double("Traffic load (Erlang)",0.4,0.01,0.999);
      printf("\n Service model:\n");
      printf("1 - Exponential:>\n");
      service_model=read_int("",1,1,1);
      duration=read_double("Average service duration (s)",0.4,0.01,100);
      inter=duration/load;
printf("SIMULATION PARAMETERS:\n\n");
      Trslen=read_double("Simulation transient len (s)", 100, 0.01, 10000)
      Trslen=Trslen;
      Runlen=read_double("Simulation RUN len (s)",  100, 0.01, 10000);
      Runlen=Runlen;
      NRUNmin=read_int("Simulation number of RUNs", 5, 2, 100);
}
```

default minimo massimo

NOTA: la lettura dei parametri avviene usando le funzioni di input/output definite nel modulo “easyio”

La classe queue: inizializzazione

```
void queue::init()
{
input();
event* Ev;
Ev=new arrival(0.0, buf);
cal->put(Ev);
buf->status=0;
}
```

inserimento del primo evento di *arrivo* nel calendario all’istante 0.0

La classe queue: funzione run()

```
void queue::run(){

      extern double     Trslen;
      extern double     Runlen;
      extern int  NRUNmin;
      extern int  NRUNmax;

        double clock=0.0;
        event* ev;
        while (clock<Trslen){
           ev=cal->get();
            ev->body();
            clock=ev->time;
            delete(ev);
            }
      clear_stats();
      clear_counters();
      int current_run_number=1;
```

esecuzione degli eventi durante il periodo considerato come transitorio

La classe queue: funzione run()

```
while(current_run_number<=NRUNmin){
      while
            (clock<(current_run_number*Runlen+Trslen))
            { ev=cal->get();
                ev->body();
                  clock=ev->time;
            delete(ev);
            }
      update_stats();
      clear_counters();
      print_trace(current_run_number);
      current_run_number++;
      }
}
```

esecuzione degli eventi durante ciascun run

aggiunta di un campione alle statistiche

azzeramento dei contatori

La classe queue: funzioni di misura

```
void queue::clear_counters()
{
      buf->tot_delay=0.0;
      buf->tot_packs=0.0;
}
```

azzeramento dei contatori

```
void queue::clear_stats()
{
      delay->reset();
}
```

azzeramento delle statistiche

```
void queue::update_stats()
{
      *delay+=buf->tot_delay/buf->tot_packs;
}
```

aggiunta di un campione alle variabili statistiche

La classe queue:
stampa dei risultati

```
void queue::results()
{
      extern double      Trslen;
      extern double      Runlen;
      extern int  NRUNmin;
      extern int  NRUNmax;

      fprintf(fpout, "**********************************************\n");
      fprintf(fpout, "            SIMULATION RESULTS                 \n");
      fprintf(fpout, "**********************************************\n\n");
      fprintf(fpout, "Input parameters:\n");
      fprintf(fpout, "Transient length (s)           %5.3f\n", Trslen);
      fprintf(fpout, "Run length (s)                 %5.3f\n", Runlen);
      fprintf(fpout, "Number of runs                 %5d\n", NRUNmin);
      fprintf(fpout, "Traffic load                   %5.3f\n", load);
      fprintf(fpout, "Average service duration       %5.3f\n", duration);
      fprintf(fpout, "Results:\n");
      fprintf(fpout, "Average Delay                  %2.6f   +/- %.2e  p:%3.2f\n",
                  delay->mean(),
                  delay->confidence(.95),
                  delay->confpercerr(.95));
}
```

La classe packet

```
/****************************************************************************
                  PACKET.H
****************************************************************************/

#ifndef PACKET_H
#define PACKET_H

#include "global.h"

class
      packet      {

      double gen_time;
      public:
      packet(double Gen_time);
      ~packet(){}
      packet* next;
      public:
      double get_time(){ return gen_time; }

      };

inline packet::packet(double Gen_time){
      gen_time=Gen_time;
      next=NULL;
      }

#endif
```

La classe buffer

```
/**************************************************************************
                  BUFFER.H
**************************************************************************/

#ifndef BUFFER_H
#define BUFFER_H

#include "packet.h"

class buffer      {

      packet* head;
      packet* last;
      public:
      int   status;

public:
      buffer();
      ~buffer(){}
      void insert(packet* pack);
      packet* get();
      packet* full(){return head;}
      double tot_delay;
      double tot_packs;
      };

#endif
```

La classe buffer

```
buffer::buffer(){
   head=NULL;
   last=NULL;
   status=0;
   tot_delay=0.0;
   tot_packs=0.0;
   }
```

```
void  buffer::insert(packet* pack){
   if(head==NULL){
      head=pack;
      last=pack;
      last->next=head;
      }
   else  {
      last->next=pack;
      last=pack;
      last->next=head;
      }
   }
```

```
packet* buffer::get(){

   packet* pack;
   if(head==NULL)
      return NULL;
   if(last==head){
      pack=head;
      last=NULL;
      head=NULL;
      }
   else  {
      pack=head;
      head=head->next;
      last->next=head;
      }
   return pack;
   }
```

La classe calendar

```
/*******************************************************
                 CALENDAR H
*******************************************************/


#ifndef _CALENDAR_H
#define _CALENDAR_H

#include "simulator.h"
#include "event.h"

class calendar{

      event*      head;
      event*      last;

      public:


      calendar();
      ~calendar();
      event*      get();
      void  put(event* New_event);
      };
```

La classe calendar

```
inline      calendar::calendar(){
      head=NULL;
      last=NULL;
      }
inline      calendar::~calendar(){
      event* temp=head;
      last->next=NULL;
      while(temp!=NULL){
            temp=temp->next;
            delete head;
            head=temp;
            }
      }
```

```
event* calendar::get(){

      if(head==NULL)
            return NULL;
      event* ev;
      if(head==last){
            ev=head;
            head=NULL;
            last=NULL;
            return ev;
            }
      ev=head;
      head=head->next;
      last->next=head;
      return ev;
      }
```

La classe calendar

```
void calendar::put(event* New){
        event* temp=head;
        event* pippo;
        pippo=New;
        if(head==NULL){
                head=New;
                head->next=New;
                last=New;
                }
        else if (New->time<head->time){
                New->next=head;
                head=New;
                last->next=head;
                }
        else if (last==head){
                if(New->time<head->time){
                        head=New;
                        head->next=last;
                        last->next=head;
                        }
                else    {
                        last=New;
                        head->next=last;
                        last->next=head;
                        }
                }
        else if (last->time<New->time){
                last->next=New;
                last=New;
                last->next=head;
                }
        else    {
                while(temp->next->time < New->time)
                        temp=temp->next;
                New->next=temp->next;
                temp->next=New;
                }
        }
```

Gli eventi: classe base

Dichiarazione

```
/*******************************************************
                 EVENT . H
*******************************************************/


#ifndef _EVENT_H
#define _EVENT_H

#include "global.h"
#include "buffer.h"


class event{
public:
      event*      next; // next event
      double      time; // event time
      event();
      event(double Time);
      event(event* Next, double Time);
      ~event(){}
      virtual void body(){}
};
```

Gli eventi: classe base

Costruttori

```
inline event::event(){
      next=NULL;
      time=-1;
      }

inline event::event(event* Next, double Time){
      next=Next;
      time=Time;
      }

inline event::event(double  Time){
      time=Time;
      }
```

Gli eventi: classi figlio

```
class  arrival: public event{

       buffer* buf;

       public:
       int source_id;
       virtual void body();
       arrival(double Time, buffer* Buf);
       };
```

```
class service: public event{

      buffer* buf;

      public:
      virtual void body();
      service(double Time, buffer* Buf): event(Time){buf=Buf;}
      };
```

Gli eventi: i corpi degli eventi

```
void arrival::body(){
        event* ev;

      // generation of next arrival
      double esito;
      GEN_EXP(SEED, inter, esito);
      ev=new arrival(time+esito, buf);
      cal->put(ev);

      // insert the new packet in the queue
      packet* pack=new packet(time);
      // if some packet is already in the buffer, just insert the new one
      if(buf->full()||buf->status){
            buf->insert(pack);
            }
      // otherwise let the packet get in the service
      else  {
            buf->tot_packs+=1.0;
            delete pack;
            GEN_EXP(SEED, duration, esito);
            ev=new service(time+esito, buf);
            cal->put(ev);
            buf->status=1;
            }
        }
```

Gli eventi: i corpi degli eventi

```
void service::body(){
      packet*pack;
      pack=buf->get();
      event* ev;
      double esito;
      GEN_EXP(SEED, duration, esito);
      if(pack!=NULL){
            ev=new service(time+esito, buf);
            cal->put(ev);
            buf->tot_delay+=time-pack->get_time();
            buf->tot_packs+=1.0;
            delete pack;
            }
      else
            buf->status=0;
      }
```

La classe Sstat

```
//----------------------------------------------------------------------------
//                      CLASS Sstat
//----------------------------------------------------------------------------

class       Sstat       {
      protected:
            int   n;
            double      x;
            double      x2;
            double  last;
            double      minValue;
            double      maxValue;
      public :
            Sstat();
            virtual ~Sstat();
            virtual void reset();
            virtual void operator+=(double);
            int   num_samples();
            double      last_sample();
            double  sum();
            double      mean();
            double      stddev();
            double      var();
            double      min();
            double      max();
            double      confidence(int p_percentage);
            double      confidence(double p_value);
            double      confpercerr(int p_percentage);
            double      confpercerr(double p_value);
            int   isconfsatisfied(double perc=1.0, double pconf=.95);
};
```

La classe Sstat

```
void Sstat::operator+=(double value) {
      n += 1;
      last = value;
      x += value;
      x2 += (value * value);
      if ( minValue > value) minValue = value;
      if ( maxValue < value) maxValue = value;
      }


          double Sstat::mean() {
                if (n > 0) return(x/n);
                else return(0.0);
                }

          double Sstat::var() {
                if (n > 1) return((x2 - ((x * x) / n)) / (n - 1));
                else return(0.0);
                }

          double Sstat::stddev() {
                return(sqrt(var()));
```

La classe Sstat

```
double Sstat::confidence(double p_value) {
      int df = n - 1;
      if (df <= 0) return HUGE_VAL;
      double t = tval((1.0 + p_value) * 0.5, df);
      if (t==  HUGE_VAL) return t;
      else return (t * stddev()) / sqrt(double(n));
      }
```

NOTA: La funzione `tval(x,k)` restituisce la CDF della t-student con k gradi di libertà

Esempio di output: file di trace

```
**********************************************
                 TRACE RUN 1
**********************************************

Average Delay                0.067655   +/- inf   p:inf
**********************************************
                 TRACE RUN 2
**********************************************

Average Delay                0.066408   +/- 1.58e-02  p:23.86
**********************************************
                 TRACE RUN 3
**********************************************

Average Delay                0.067213   +/- 4.65e-03  p:6.91
**********************************************
                 TRACE RUN 4
**********************************************

Average Delay                0.066161   +/- 4.16e-03  p:6.29
**********************************************
                 TRACE RUN 5
**********************************************

Average Delay                0.067361   +/- 4.36e-03  p:6.48
```

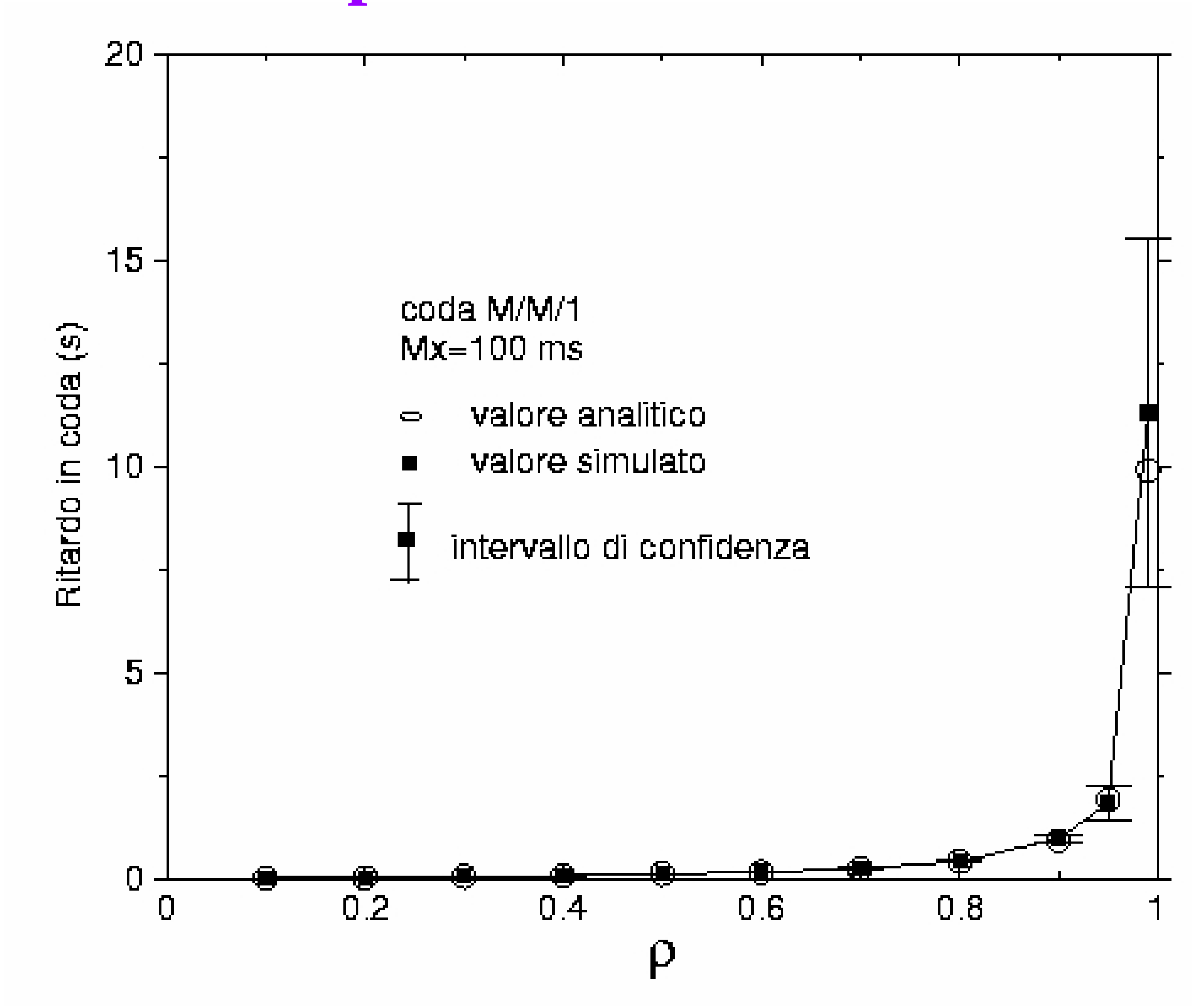
(…)

Esempio di output: file di output

```
*********************************************
           SIMULATION RESULTS
*********************************************

Input parameters:
Transient length (s)         1000.000
Run length (s)               1000.000
Number of runs                  10
Traffic load                 0.400
Average service duration     0.100
Results:
Average Delay                0.066389   +/- 2.33e-03  p:3.52
```

Esempio di risultati ottenibili


coda M/M/1
Mx=100 ms
○ valore analitico
■ valore simulato
intervallo di confidenza
Ritardo in coda (s)
ρ