CSIM

Emiliano Casalicchio
emiliano.casalicchio@uniroma2.it

CSIM Simulation Engine

- CSIM è un *process-oriented discrete-event* simulation package integrabile in programmi C/C++.
  - E' una libreria di routines che implementano le operazioni e le strutture dati necessarie a costruire un modello di simulazione ad eventi discreti
- Un programma CSIM modella un sistema come una collezione di processi CSIM i quali interaggiscono tra di loro utilizzando le strutture dati CSIM
- Il modello mantiene un tempo simulato, così da poter studiare il comportamento dinamico del sistema modellato

Oggetti CSIM

- Processes: entità attive che
  - usano risorse (facility, storage, mailboxes),
  - attendono eventi,
  - Collezionano statistiche

> "It is the processes which mimic the behaviour of active entities in the simulated system."

- Facilities: modellano code e serventi, sono utilizzate dai processi
- Storages: risorse che possono essere parzialmente/ discretamente allocate da un processo
  - Ad es. Una porzione di memoria, un descrittore di file o processo, un socket…
- Events: usati per sincronizzare i processi
- Mailboxes: usate per la comunicazione tra processi

Oggetti CSIM(2)

- Data collection structures: usate per collezionare dati durante una simulazione
  - Ad es. l'andamento temporale di una variabile di stato (lunghezza di una coda)
- Process classes: usate per caratterizzare I processi nell'uso di risorse, eventi e statistiche
- Streams - streams di numeri casuali

Oggi vedremo

- Processes: entità attive che
  - usano risorse (facility, storage, mailboxes),
  - attendono eventi,
  - Collezionano statistiche
- Facilities: modellano code e serventi, sono utilizzate dai processi
- Storages: risorse che possono essere parzialmente/discretamente allocate da un processo
  - Ad es. Una porzione di memoria, un descrittore di file o processo, un socket…
- Events: usati per sincronizzare i processi
- Mailboxes: usate per la comunicazione tra processi
- Data collection structures: usate per collezionare dati durante una simulazione
  - Ad es. l'andamento temporale di una variabile di stato (lunghezza di una coda)
- Process classes: usate per caratterizzare I processi nell'uso di risorse, eventi e statistiche
- Streams - streams di numeri casuali

Processes

- Entità attive in un modello CSIM
  - Sistema Client-Server: client modellato con un processo, server con facility
- E' una procedura C che esegue una *create*
  - Processo CSIM ≠ Processo Unix
  - *create* simile ad una *fork* unix.
- + istanze di uno stesso processo attive "simultaneamente"
  - Eseguite in parallelo rispetto al Tempo Simulato…
  - …Ma eseguite serialmente sul processore
- Ogni istanza di processo CSIM ha il proprio ambiente di esecuzione
  - Variabili locali e argomenti di input (non può ritornare un valore)
  - Variabili globali a cui hanno accesso tutti i processi

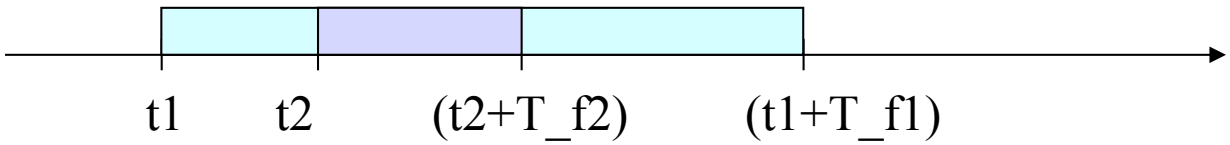
Processes (2)

- Un processo CSIM può trovarsi in 4 diversi stati:
  - In esecuzione (active)
  - Pronto (Ready) per essere eseguito (ad esempio ha ottenuto la facility o la quantità di storage attesa)
  - Bloccato (Hold) x consentire al tempo simulato di trascorrere
  - In Attesa (Wait) del verificarsi di un evento

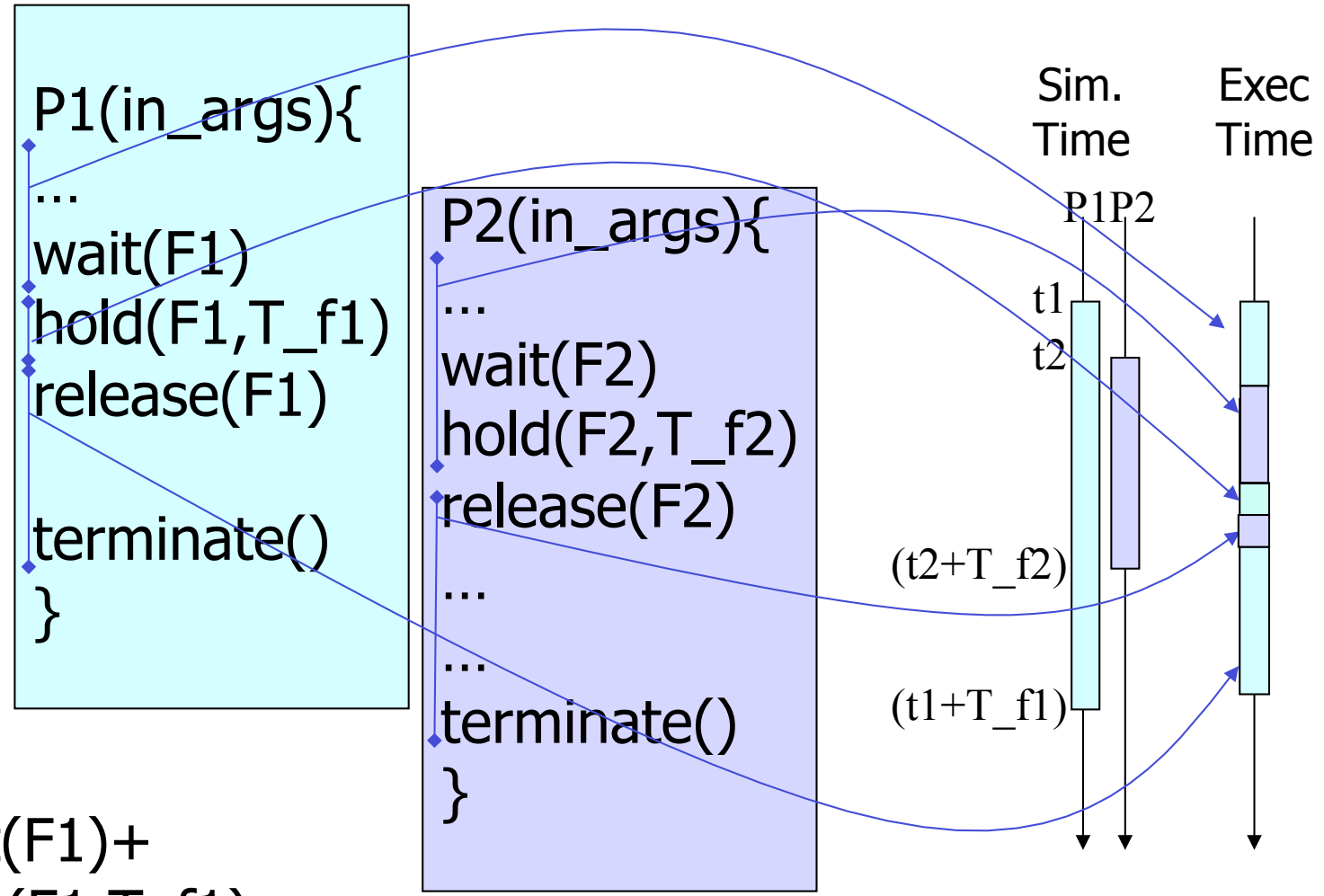
- Quando un processo termina il suo ambiente CSIM (variabili e statistiche) viene eliminato
  - Terminate(): terminazione esplicita
  - Exit (e.g. for an external interrupt): terminazione implicita.
- Ogni processo ha un
  - ID unico,
  - Priorità

Process

- Processi eseguiti simultaneamente
  - Il CSIM process manager crea questa illusione attivando e sospendendo I processi quando il tempo avanza e gli eventi si verificano
- ES: P1 e P2 attendono di utilizzare rispettivamente le facility F1 e F2



Process



wait(F1)+
hold(F1,T_f1)+
release (F1) = use(F1,T_f1)

Process

- I processi vengono eseguiti finche non invocano una
  - hold
  - use
  - allocate
  - wait

  } Accodano il processo
  - Terminate
- Processi vengono ri-avviati quando
  - Il tempo di hold termina,
  - L'attesa in coda termina (per l'uso di una facility/storage/evento)
- Il tempo simulato trascorre solo se viene eseguita una
  - hold
  - use

- Processes: entità attive che
  - usano risorse (facility, storage, mailboxes),
  - attendono eventi,
  - Collezionano statistiche
- Facilities: modellano code e serventi, sono utilizzate dai processi
- Storages: risorse che possono essere parzialmente/discretamente allocate da un processo
  - Ad es. Una porzione di memoria, un descrittore di file o processo, un socket…
- Events: usati per sincronizzare i processi
- Mailboxes: usate per la comunicazione tra processi
- Data collection structures: usate per collezionare dati durante una simulazione
  - Ad es. l'andamento temporale di una variabile di stato (lunghezza di una coda)
- Process classes: usate per caratterizzare I processi nell'uso di risorse, eventi e statistiche
- Streams - streams di numeri casuali

Facilities

- Usate x modellare risorse
  - CPU, disk drive possono essere modellate con una facility
- Una simple facility consiste di un sigolo servente con una coda singola
  - Solo un processo alla volta può utilizzare il servente
- Una multiserver-server facility contiene una coda singola ed n-server
  - Tutti i processi sono messi nella coda di attesa e sbloccati non appesa si libera un server.
- I processi sono ordinati nelle code CSIM (di facility, storage o eventi) in base alla loro priorità
  - Un processo ad alta priorità viene processato prima di uno a bassa priorità
  - In caso di stessa priorità l'ordine di processamento è fcfs
  - la disciplina di servizio di una facility può essere cambiata

Esempio 1: modello C/S semplice

- Il client è modellato con un processo CSIM
- Il server è modellato con una facility
- Il processo degli arrivi (esponenziale) è modellato generando una sequenza di client in un ciclo while.

```
/* Esempio 1*/
/* !!!!!!!!!Pseudo Codice!!!!!!!!!! */
#include “csim.h”;

void sim( int argc,
        char *argv[]) {

FACILITY Server; /* Definizione di una facility */

generated_client=0;
resetted=false;
stop_client=false;

Server = facility(“Server”); /* Inizializzazione di una facility */
```

```
/* simtime() mi restituisce il tempo di simulzione*/

while( simtime() <= SIMULATION_TIME )        {

interarrival_time = exponential( 0.1 ); /* Genera il tempo di interarrivo delle
    richieste distribuito come un'esponenziale di media 0.1*/

hold( (double)interarrival_time );

Web_client( generated_client );
generated_client++;

if ((simtime()>TRANSIENT_TIME)&&(!resetted)) {
reset();
resetted=true;
}
}
```

```
void Web_client( long int my_number ) {

    sprintf( strbuf, "Web_client%d", my_number );
    create( strbuf ); /* Creazione del processo */

use(Server, exponential(0.4) ); /* Utilizzo della facility */

/*
L'operazine use() realizza la sequenza delle 3 operazioni:
reserve(facility); //accoda se occupato riserva se libero
hold(hold_time); //fa avanzare il tempo simulato
release(facility); //rilascia la facility
*/

    terminate();

}
```

Facilities: reports

- Rapporti statistici per le facility possono essere prodotti chiamando le *report functions*.
- La funzione report_facilities stampa i rapporti relativi a tutte le facility instanziate.
- Per ogni facility vengono riportati:
  - facility name
  - service discipline
  - service time
  - utilization
  - throughput rate
  - queue length
  - response time
  - completed service requests

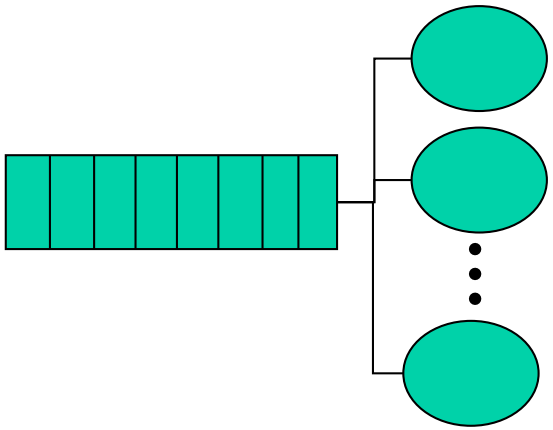
FACILITY: Report

FACILITY SUMMARY

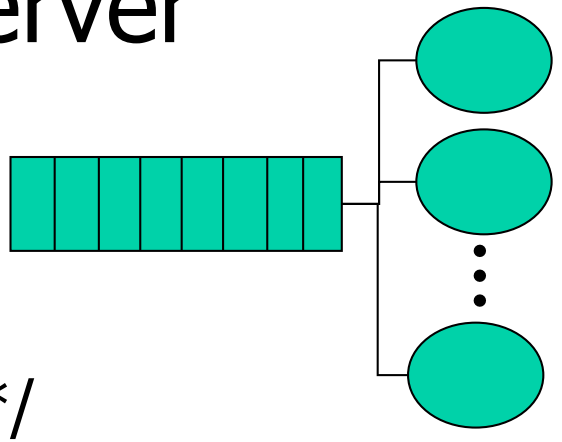
facility name	services disc	service time	util.	through put	queue length	response time	compl count
f	fcfs	0.40907	0.208	0.50900	0.27059	0.53162	509
ms fac	fcfs	1.50020	0.764	0.50900	0.83821	1.64678	509
> server	0	1.55358	0.494	0.31800			318
> server	1	1.41133	0.270	0.19100			191
q	rnd_rob	0.73437	0.507	0.69000	0.95522	1.38438	690

Facility Multiserver

- coda unica, centri servizio indipendenti
  - Network Dispatcher
  - HTTPd con architettura a “dispatcher” e numero fisso di istanze di processo
  - Ufficio postale, Banca, panettiere!!!



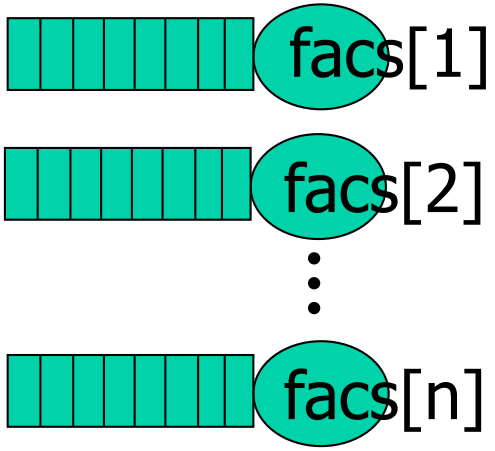
FACILITY: Multiserver



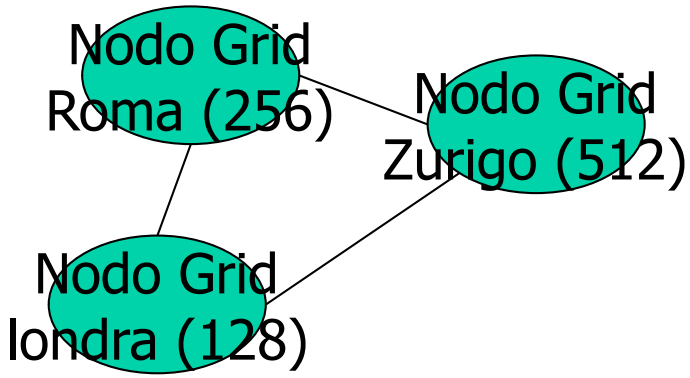
```
/* set number of servers to 3 */
#define NUM_SRVRS 3
FACILITY multi_server; /* declare facility*/
...
multi_server = facility_ms("multi srvr", NUM_SRVRS); /
   *initialize srvr with 3 srvrs*/
...
use(multi_server, service_time); /*use facility for length of
   service_time*/
```

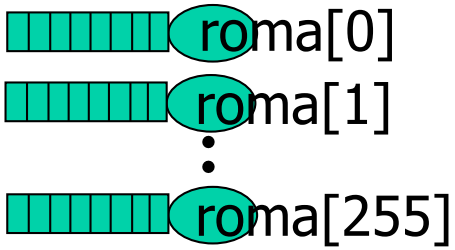
Array of Facilities

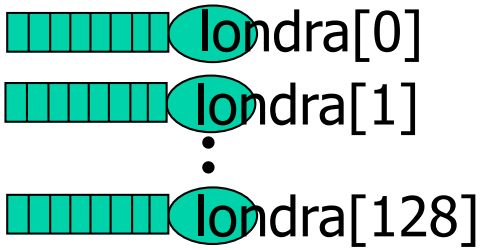
- code indipendenti
  - Cluster di server
  - Array di dischi
  - Postazioni di lavoro
  - Supermercato

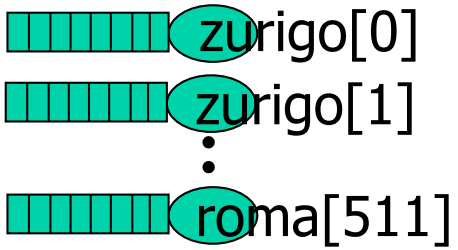


- Array di Facility rappresenta serventi tra cui esiste una qualche relazione logica

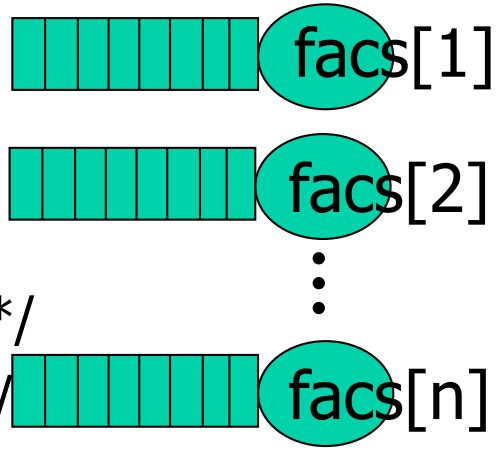








FACILITY: ARRAY of




```
#define NUM_FACS 10 /*set the number of facilities */
FACILITY facs[NUM_FACS]; /* declare facility array */
...
facility_set(facs, "facs", NUM_FACS); /*initialize a set of 10 facilities */

i = random(0, NUM_FACS-1); /*select a facility*/

use(facs[i], service_time); /*use facility[i] for service_time time unit*/
```

Esempi

- licenceserver.c
- arrayofserver.c
  - Homeworks:
    - implementare la politica di selezione round-robin
    - Confrontare i risultati tra la politica unif. e rr al variare del valor medio di intT.
- multiserver.c
  - Homework
    - Modellare il dbase con un array of server (selezione uniforme)
    - Confrontare i risultati ottenuti con i due modelli al variare del valor medio di intT

FACILITY: Service Function

```
FACILITY cpu; /*declare facility variable cpu */
    ...
    cpu = facility("cpu"); /*initialize facility and name it cpu */
    set_servicefunc(cpu, pre_res) /*set service protocol to
    preempt-resume*/
    ...
    priority = 100; /*make process high priority */
    use(cpu, service_time); /*preempt lower priority process and
    use facility*/
    ...
```

FACILITY: service discipline (1)

- fcfs - first come, first served
- inf_srv - infinite servers
- lcfs_pr - last come, first served, preempt
  - Un nuovo processo è sempre servito, affettuando preeption sul processo in servizio.
  - Priorità non considerata
- prc_shr - processor sharing
  - Tempo di servizio è determinato in base al numero di processi alla facility. Di default il tasso è n (numero di proc alla facility), ossia se ci sono n processi alla facility il tempo di servizio è moltiplicato di un fattore n
  - No queuing delay
  - Massimo 100 processi possono condividere una facility prc_shr

Prototype: void set_loaddep(FACILITY f, double rate[], long n)
Example: set_loaddep(f, rate, 10);

FACILITY: service discipline (2)

- pre_res - preempt resume
  - Processi alta priorità effettuano preemption su quelli a priorità + bassa
  - Processi con priorità uguale sono serviti con politica fcfs
- rnd_rob - round robin
  - Processi serviti in modalità RR.
  - Ogni processo utilizzerà la facility per un TIMESLICE specificato in set_timeslice
  - NO priority
- rnd_pri - round robin with priority
  - Processi ad alta priorità serviti prima.
  - Processi con la stessa priorità serviti in modalità RR.

Esempio

- servicefunc.c
  - Fare confronti tra le varie discipline di servizio al variare di intT

Esempio rr vs fcfs

IntT	policy	Serv-time	util.	Through put	Queue-lenght	Response time	Compl-count
1	fcfs	1.61508	0.99	0.61289	24.77065	40.41622	55
2	fcfs	1.6367	0.797	0.48723	1.52322	3.12628	43
3	fcfs	1.59144	0.553	0.34752	0.77289	2.22405	31
4	fcfs	1.45464	0.383	0.26336	0.42427	1.61101	24
5	fcfs	1.46897	0.331	0.22552	0.35204	1.56101	21
1	rnd_rob	0.75347	0.991	1.31493	31.75144	24.14695	118
2	rnd_rob	0.84793	0.797	0.94047	1.92905	2.05115	83
3	rnd_rob	0.897	0.553	0.61656	0.93743	1.52041	55
4	rnd_rob	0.99747	0.383	0.38406	0.49011	1.27612	35
5	rnd_rob	1.14253	0.331	0.28996	0.38426	1.32523	27

Uso della priorità dei processi

- priority.c

risultati

facility name	service disc	service time	util.	through-put	queue length	response time	compl count
httpd	rnd_pri	0.01436	0.992	69.10073	7.43680	0.10762	6266
> class low pri		0.01031	0.630	61.06140	6.98122	0.11433	5537
> class high pri		0.04511	0.363	8.03933	0.45264	0.05630	729

PROCESS CLASS SUMMARY

id	name	number	lifetime	hold count	hold time	wait time
0	default	58	1.78488	1.00000	1.56343	0.22144
1	low pri	37	17.11693	154.62162	1.54334	15.57359
2	high pri	21	1.95452	157.04762	1.56613	0.38839

- Processes: entità attive che
  - usano risorse (facility, storage, mailboxes),
  - attendono eventi,
  - Collezionano statistiche
- Facilities: modellano code e serventi, sono utilizzate dai processi
- Storages: risorse che possono essere parzialmente/discretamente allocate da un processo
  - Ad es. Una porzione di memoria, un descrittore di file o processo, un socket…
- Events: usati per sincronizzare i processi
- Mailboxes: usate per la comunicazione tra processi
- Data collection structures: usate per collezionare dati durante una simulazione
  - Ad es. l'andamento temporale di una variabile di stato (lunghezza di una coda)
- Process classes: usate per caratterizzare I processi nell'uso di risorse, eventi e statistiche
- Streams - streams di numeri casuali

Esempio di uso

- Allocazione spazio su disco.
  - Unità di allocazione blocchi 2Kbyte.
  - nblocchi=sup(filesize_kbyte/2), es.: filesize=3548byte, nblocchi=sup(3548/2048)=sup(1.732)=2.
  - Operazione di memorizzazione su disco (allocate)
  - Operazione di cancellazione da disco (deallocate)
- Connettori ad un db o server licenza con postazioni limitate (es accesso ieee dlibrary).

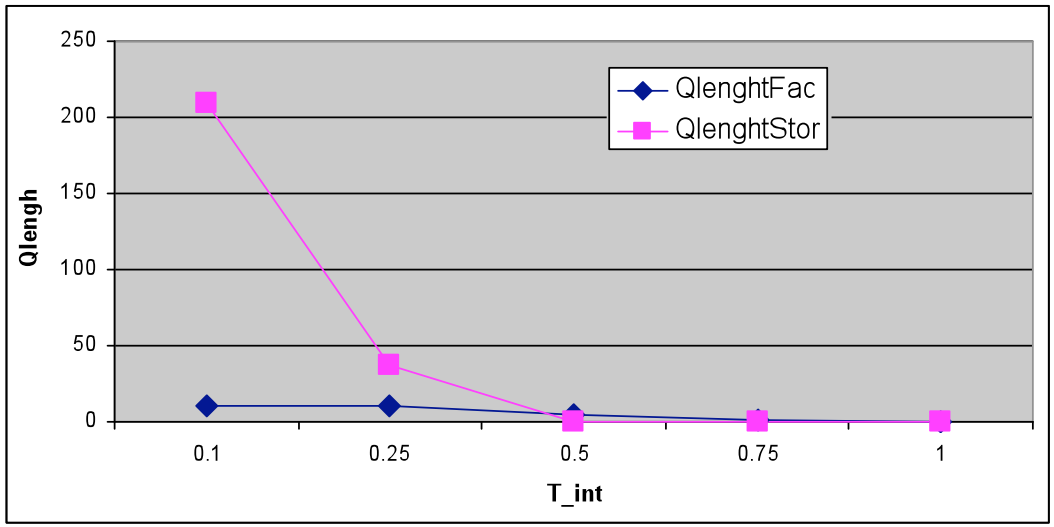
STORAGE: inizialize and use

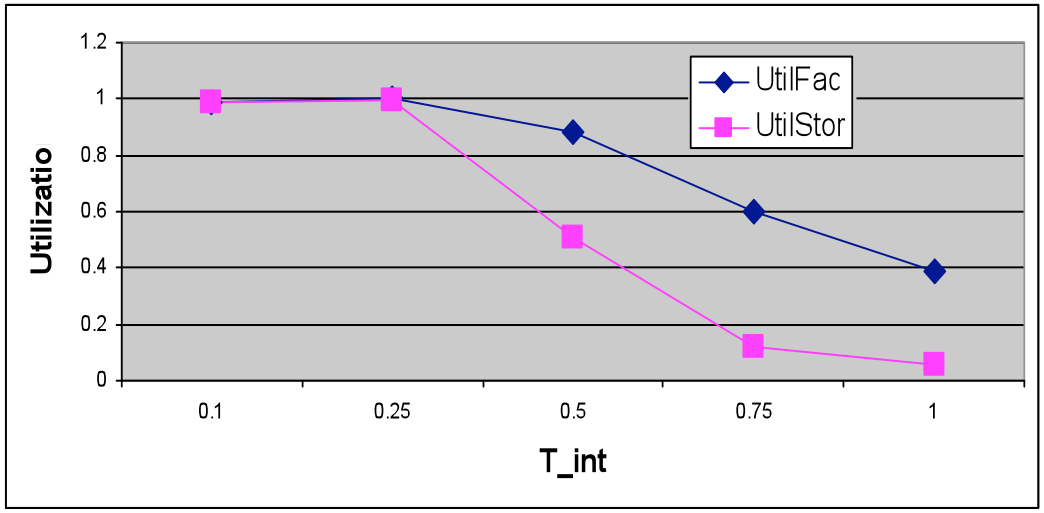
```
#define STORE_AMT 100 /*set amount of storage to 100 units
   */
   STORE mem; /*declare storage variable mem */
   ...
   mem = storage("mem", STORE_AMT); /*initialize storage
   named mem with 100 units */
   ...
   amt = random(1, STORE_AMT); /*decide how much storage
   to allocate this time */
   allocate(amt, mem); /*get amount of storage decided upon*/
   ...
   deallocate(amt, mem); /* release storage which is no longer
   needed */
   ...
```

Esempi

- licenceserver.c

Risultati licencerver.c





Tint	Req	Active
0.1	514	412
0.25	186	72
0.5	109	11
0.75	72	2
1	48	1

STORAGE: statistics

char* storage_name(STORE s) pointer to name of store

long storage_capacity(STORE s) number of storages defined for store

long avail (STORE s) number of storages currently available at store

long storage_qlength(STORE s) number of processes currently waiting at store

double storage_request_amt(STORE s) sum of requested amounts from store

long storage_number_amt(STORE s) time-weighted sum of requesters for store

double storage_busy_amt(STORE s) busy time-weighted sum of amounts for store

double storage_waiting_amt(STORE s) waiting time weighted sum of amounts for store

long storage_request_cnt(STORE s) total number of requests for store

long storage_release_cnt(STORE s) total number of completed requests for store

long storage_queue_cnt(STORE s) number of queued requests at store

double storage_time(STORE s) time at store that is spanned by report

- Processes: entità attive che
  - usano risorse (facility, storage, mailboxes),
  - attendono eventi,
  - Collezionano statistiche
- Facilities: modellano code e serventi, sono utilizzate dai processi
- Storages: risorse che possono essere parzialmente/discretamente allocate da un processo
  - Ad es. Una porzione di memoria, un descrittore di file o processo, un socket…
- Events: usati per sincronizzare i processi
- Mailboxes: usate per la comunicazione tra processi
- Data collection structures: usate per collezionare dati durante una simulazione
  - Ad es. l'andamento temporale di una variabile di stato (lunghezza di una coda)
- Process classes: usate per caratterizzare I processi nell'uso di risorse, eventi e statistiche
- Streams - streams di numeri casuali

EVENTS: wait e queue

- Un evento può trovarsi nello stato
  - OCC (occurred) o NOT_OCC (not occurred)
- Se un processo WAIT per un evento ***e***
  - Se stato(e)=NOT_OCC il processo viene sospeso. Tutti i processi in attesa vengono risvegliati quando stato(e)=OCC (settato da qualche processo)
  - Se stato(e)=OCC il processo continua e viene settato stato(e)=NOT_OCC
- Se un processo QUEUE per un evento ***e***
  - Se stato(e)=NOT_OCC il processo viene accodato. Quando stato(e)=OCC (settato da qualche processo) il primo processo in coda viene risvegliato
  - Se stato(e)=OCC e non ci sono altri processi in attesa, il processo continua e viene settato stato(e)=NOT_OCC

EVENTS

```
EVENT ev; /*declare event variable ev */
...
ev = event("ev"); /*initialize an event named ev */
....
wait(ev); /*wait for event to occur before proceeding */
...
queue(ev); /*wait for event to occur, for processes to
respond before proceeding*/
set(ev); /*indicate that an event has occurred */
...
```

EVENTS: array of

```
#define NUM_EVENTS 25 /*set number of events in array */
EVENT ev_arr[NUM_EVENTS]; /*declare event array */
....
event_set(ev_arr, "ev arr", NUM_EVENTS); /*initialize array of 25
events*/
....
wait(ev_arr[5]); /*wait for sixth event to occur before proceeding */
...
set(ev_arr[5]); /*indicate that sixth event has occurred*/
...

i = wait_any(ev_arr); /*i is index of event which occurred */
OR
i = queue_any(ev_arr); /*i is index of event which occurred */
...
....
```

EVENTS: wait with TimeOut

```
st = timed_wait(ev, 50.0); /*wait for a maximum of 50 time
units */
if(st ! = TIMED_OUT) { /*did not timed out */
.....
}
```

Esempi

- Interrupt.c
- Interrupt_multihandler.c

Interrupt.c

EVENT SUMMARY

event name	number of queue vst	avg que length	avg time queued	number of wait vsts	avg wait length	avg time waiting	number of set ops
end_sim	0	0.00000	0.00000	0	0.00000	0.00000	0
int_vect.se	0	0.00000	0.00000	419	0.90149	0.96818	0
int_vect[0]	0	0.00000	0.00000	0	0.00000	0.00000	43
int_vect[1]	0	0.00000	0.00000	0	0.00000	0.00000	36
int_vect[2]	0	0.00000	0.00000	0	0.00000	0.00000	35
int_vect[3]	0	0.00000	0.00000	0	0.00000	0.00000	38
int_vect[4]	0	0.00000	0.00000	0	0.00000	0.00000	55
int_vect[5]	0	0.00000	0.00000	0	0.00000	0.00000	47
int_vect[6]	0	0.00000	0.00000	0	0.00000	0.00000	63
int_vect[7]	0	0.00000	0.00000	0	0.00000	0.00000	48
int_vect[8]	0	0.00000	0.00000	0	0.00000	0.00000	41
int_vect[9]	0	0.00000	0.00000	0	0.00000	0.00000	53

Interrupt_multihadler.c

EVENT SUMMARY

event name	number of queue vst	avg que length	avg time queued	number of wait vsts	avg wait length	avg time waiting	number of set ops
end_sim	0	0.00000	0.00000	0	0.00000	0.00000	0
int_vect.se	0	0.00000	0.00000	0	0.00000	0.00000	0
int_vect[0]	0	0.00000	0.00000	12	0.97696	36.63590	12
int_vect[1]	0	0.00000	0.00000	19	0.99672	23.60654	19
int_vect[2]	0	0.00000	0.00000	17	0.94184	24.93105	17
int_vect[3]	0	0.00000	0.00000	12	0.92019	34.50701	12
int_vect[4]	0	0.00000	0.00000	14	1.06473	34.22334	14
int_vect[5]	0	0.00000	0.00000	10	1.05877	47.64456	10
int_vect[6]	0	0.00000	0.00000	11	0.88718	36.29381	11
int_vect[7]	0	0.00000	0.00000	12	1.00357	37.63384	12
int_vect[8]	0	0.00000	0.00000	18	0.88113	22.02822	18
int_vect[9]	0	0.00000	0.00000	18	1.01889	25.47217	18

Download CSIM19

http://www.ce.uniroma2.it/~casali/download/CD-CSIM19/

Manuale

http://www.mesquite.com/documentation/index.htm

Installazione CSIM19

- gcc
  - Creare una directory csim19 in /usr/local
  - Decomprimere l'archivio csim19 nella directory csim19
    - tar -xf csim19_linux_gcc.tar (oppure tar –xf csim19_linux_gcc.tar)
  - Entrare nella directory /usr/local/csim19/lib
    - make
- gpp
  - Creare una directory csim19 in /usr/local
  - Decomprimere l'archivio csim19 nella directory csim19
    - tar -xf csim19_linux_gpp.tar (oppure tar –xf csim19_linux_gpp.tar)
  - Entrare nella directory /usr/local/csim19/lib
    - make

Compilazione

- csim.gcc
  - gcc -Ilib $* lib/csim.gcc.a –lm
- csim.gcc <nomefile.c>