CSIM
(2ª parte)

Emiliano Casalicchio
emiliano.casalicchio@uniroma2.it

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*/

...

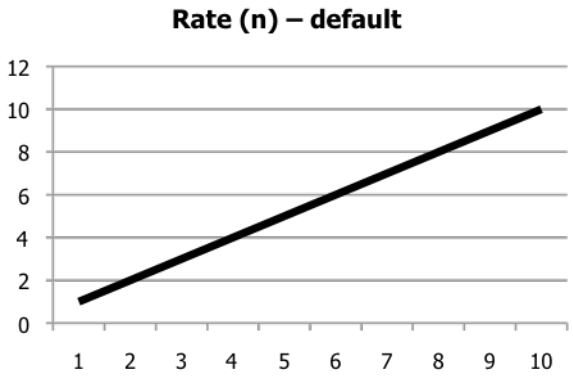
FACILITY: service discipline (1)

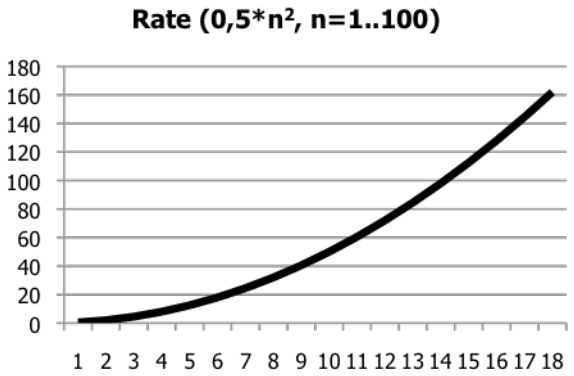
- **fcfs** - first come, first served
- **inf_srv** - infinite servers
- **lcfs_pr** - last come, first served, preempt
  - Un nuovo processo è sempre servito, affettuando preeption sul processo in servizio.
  - Priorità non considerata

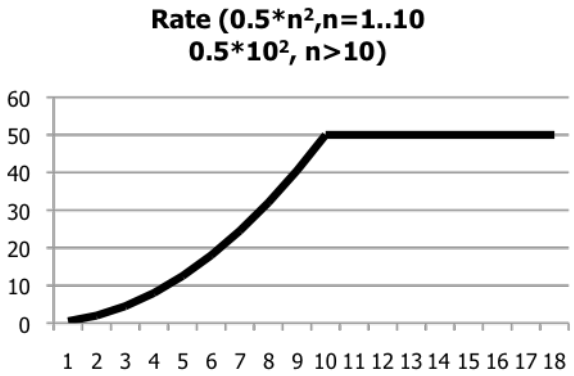
FACILITY: service discipline (1)

- **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).

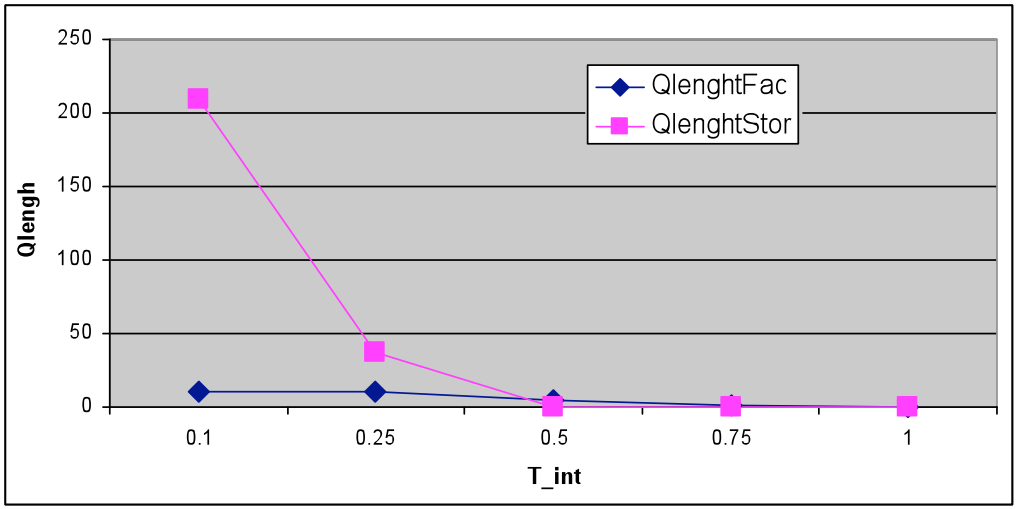
STORAGE: inizialize and use

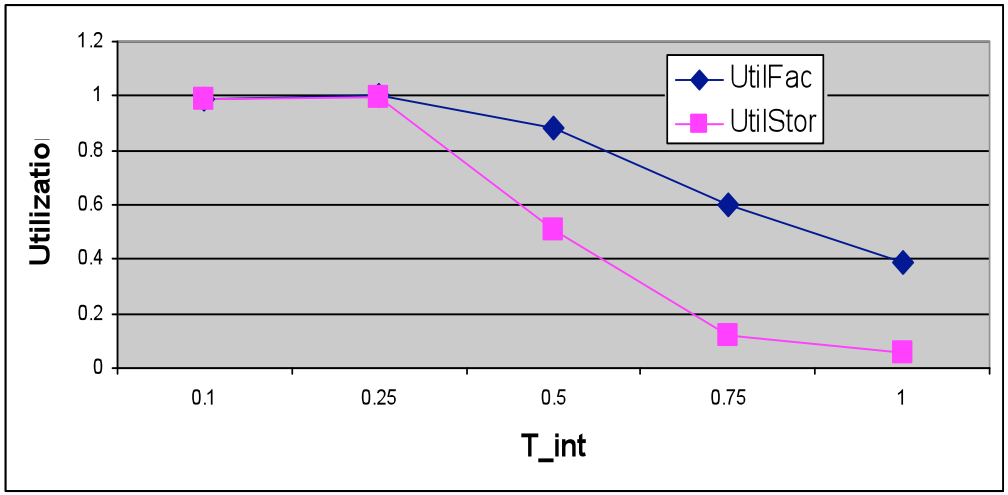
```
#define STORE_AMT 100 /*set amount of storage to 100 units
   */
STORE mem; /*declare storage variable mem */
…
mem = storage("mem", STORE_AMT); /*initialize storage
   named mem with 100 units */
…
amt = random(1, STORE_AMT); /*decide how much storage to
   allocate this time */
allocate(amt, mem); /*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

Generazione di numeri casuali e controllo della simulazione in CSIM

Emiliano Casalicchio
emiliano.casalicchio@uniroma2.it

Agenda

- Generazione di numeri casuali in CSIM
- Terminazione della simulazione in CSIM

Random Numbers

- Possibilità di generare
  - Single stream
  - Multiple stream
- Ditribuzioni Continue
- Distribuzioni Discrete
- Distribuzioni Empiriche

Continuous Distributions

- double uniform(double min, double max)
- double triangular(double min, double max, double mode)
- double beta(double min, double max, double shape1, double shape2)
- double exponential(double mean)
- double gamma(double mean, double stddev)
- double erlang(double mean, double var)
- double hyperx(double mean, double var)
- double weibull(double shape, double scale)
- double normal(double mean, double stddev)
- double lognormal(double mean, double stddev)
- double cauchy(double alpha, double beta)
- double hypoexponential(double mn, double var)
- double pareto(long a)
- double zipf(long n)
- double zipf_sum(long n, double *sum)

Discrete Distributions

- long random_int(long min, long max)
- long bernoulli(double prob_success)
- long binomial(double prob_success, long
- num_trials)
- long geometric(double prob_success)
- long negative_binomial(long success_num, double prob_success)
- long poisson(double mean)

Empirical Distributions

- **void setup_empirical(long n, double prob[], double cutoff[], long alias[])**
  - Deve essere invocata in fase di setup dei generatori di numeri casuali
  - **prob[]** contiene le probabilità di generare un numero

- **double empirical(long n, double cutoff[], long alias[], double value[])**
  - Permette di estrarre un valore dal set value[] in base alle probabilità specificate in prob[]
  - **prob[i]= probabilità di estrarre il valore value[i]**

Value[]	Prob[]
5	0.3
7	0.2
9	0.4
11	0.07
15	0.03
-	**1.0**

Single stream

- Di default CSIM usa un'unico stream di numeri, ed il seme è automaticamente impostato ad 1
- Per cambiare seme:
  - void reseed(STREAM s, long n)
  - Es: reseed(NIL, 13579);
  - Il seme può essere cambiato in qualsiasi punto della simulazione
- Per conoscere lo stato attuale di uno stream
  - long stream_state(STREAM s)
  - Es: stream_state(NIL) ritorna lo stato della stream unica.
  - Es: reseed(NIL, 13579); i=stream_state(NIL) /* i=13579 */

Multiple stream

- **Variabili aleatorie indipendenti richiedono sequenze indipendenti. Utililizzando stream separate si è sicuri di avere stream indipendenti**

- CSIM mette a disposizione 100 stream indipendenti i cui seed sono spaziati di 100.000 valori.
  - STREAM create_stream(void)
  - Es:

  STREAM s;

  s = create_stream();

  reseed(STREAM s, long n);
- double stream_<distribFuncName>(STREAM s, <distrib_parameters>);

Esempi CSIM

- single-stream.c, multistream.c
- single-stream.c
  - Seed=35 Seed=37

```
t=0.011882, srv=2       t=3.47511, srv=2
t=0.11806, srv=2        t=3.54453, srv=3
t=0.615691, srv=3       t=4.76106, srv=4
t=1.27062, srv=4        t=5.05914, srv=0
t=1.72347, srv=1        t=5.3492, srv=3
t=1.8161, srv=2         t=6.11876, srv=4
t=1.83036, srv=3        t=6.73165, srv=2
t=1.86344, srv=0        t=6.85498, srv=1
t=1.93414, srv=3        t=8.16424, srv=2
t=3.95347, srv=4        t=11.5625, srv=4
t=5.74728, srv=4        t=11.6885, srv=2
t=7.05873, srv=0        t=12.5472, srv=4
t=9.43674, srv=3        t=12.9105, srv=0
t=9.68803, srv=2        t=14.002, srv=4
t=9.81166, srv=3        t=14.2299, srv=1
t=9.90979, srv=4        t=14.9446, srv=2
t=10.3409, srv=0        t=16.217, srv=1
t=10.7878, srv=2        t=18.1702, srv=3
```

Trace delle simulazioni

- Trace simulation
  - Possibilità di tracciare i processi
    - trace_on(); trace_process(processName);
  - Possibilità di tracciare gli oggetti, e.g. eventi, storages, facility, etc…

Esempio di trace

time	process	id	pri	status
21.382	AT2	4	1	use facility httpd for 0.004
21.382	AT2	4	1	reserve facility httpd
21.382	AT3	5	1	use facility httpd for 0.030
21.382	AT3	5	1	reserve facility httpd
21.382	AT4	6	1	use facility httpd for 0.089
21.382	AT4	6	1	reserve facility httpd
21.382	AT5	7	1	use facility httpd for 0.025
21.382	AT5	7	1	reserve facility httpd
21.382	AT6	8	1	use facility httpd for 0.028
21.382	AT6	8	1	reserve facility httpd
21.382	AT7	9	1	use facility httpd for 0.004
21.382	AT7	9	1	reserve facility httpd
21.382	AT8	10	1	use facility httpd for 0.357
21.382	AT8	10	1	reserve facility httpd
21.382	AT9	11	1	use facility httpd for 0.114
21.382	AT9	11	1	reserve facility httpd
21.382	AT10	12	1	use facility httpd for 0.028
21.382	AT10	12	1	reserve facility httpd
21.382	AT0	33	1	wait event burst (#9d49e28)
21.382	AT1	34	1	wait event burst (#9d49e28)
21.382	AT2	35	1	wait event burst (#9d49e28)
21.382	AT3	36	1	wait event burst (#9d49e28)
21.382	AT4	37	1	wait event burst (#9d49e28)
21.382	AT5	38	1	wait event burst (#9d49e28)
21.382	AT6	39	1	wait event burst (#9d49e28)
21.382	AT7	40	1	wait event burst (#9d49e28)
21.382	AT8	41	1	wait event burst (#9d49e28)
21.382	AT9	42	1	wait event burst (#9d49e28)
21.382	AT10	43	1	wait event burst (#9d49e28)
21.510	WC10	23	1	terminate process
22.535	WC20	44	1	use facility httpd for 1.258
22.535	WC20	44	1	reserve facility httpd
22.548	WC11	24	1	terminate process

Intervallo di confidenza e controllo della lunghezza dei run.

- Possibilità di calcolare l'intervallo di confidenza del valore medio osservato (per ogni oggetto per il collezionamento delle statistiche)
  - void table_confidence(TABLE t)
  - void qtable_confidence(QTABLE qt)
  - void meter_confidence(METER m)
  - void box_time_confidence(BOX b)
  - void box_number_confidence(BOX b)
- La tecnica utilizzata per calcolare l'intervallo di confidenza è la: ***Batch means analysis.***
  - Viene calcolato l'intervallo di confidenza per 3 livelli di accuratezza: 90, 95, 98%

Esempio

- confidenceInterval.c
  - out.confidenceIntervalYesNo
  - out.confidenceIntervalYesYes

Run length control

- E' possibile controllare la terminazione di una simulazione basandosi sull'osservazione di una o più variabili e sulla convergenza del loro valore medio all'interno di un'intervallo di confidenza.
  - void table_run_length(TABLE t, double accuracy, double conf_level, double max_time)
  - **accuracy** specifica il massimo errore relativo che sarà consentito al valore medio della variabile osservata (0.1, 0.01, 0.001, …)
  - **conf_level** specifica l'intervallo di confidenza, eg=0.95
  - **max_time** specifica il tempo massimo da simulare se la variabile osservata non converge
- Quando la simulazione terminerà, l'evento converged sarà impostato ad occurred
  - wait(converged);

Esempi

- runLenghtControl.c