CSIM

Emiliano Casalicchio
emiliano.casalicchio@uniroma2.it

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

FACILITY: reserve with TimeOut

Esempio: Vogliamo modellare un timeout di connessione ad un server (web, o ssh, o ftp)

FACILITY: reserve with TimeOut

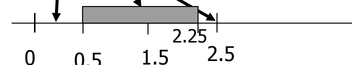
```
#define TIME_OUT 5.0 /*set length of time to wait for
                        facility*/
.....
st = timed_reserve(single_server, TIME_OUT); /*reserve
                        facility in 5 time units*/
if(st != TIME_OUT) { /*if facility was, in fact, reserved in
                        time*/
    hold(service_time); /*simulate servicing customer for
                        service_time*/
    release(single_server); /*release facility since service is
                        now complete*/
} else { /*request timed out */
....
}
```


Esempi

- ReserveTimeout.c
  - Studiare il comportamento del sistema al variare del TIME_OUT e al variare di intT

FACILITY: Synchronized

```
#define PHASE 0.5 /*set time to onset of first clock cycle to 0.5 */
#define PERIOD 1.0 /*set length of clock cycle to 1 time unit*/
FACILITY bus; /*declare facility variable bus */
...
bus = facility("bus"); /*initialize facility and name it bus */
synchronous_facility(bus, PHASE, PERIOD); /*make the facility
synchronous */
...
reserve(bus); /*reserve the facility */
hold(bus,1.75);
release(bus); /*release facility since process no longer needs it*/
...
```



FACILITY: Service Function

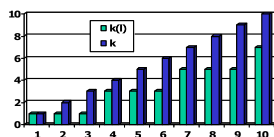
```
FACILITY cpu; /*declare facility variable cpu */
   ...
   cpu = facility("cpu"); /*initialize facility and name it cpu */
   set_servicefunc(cpu, pre_res) /*set service protocol to
   preempt-resume*/
   ...
   priority = 100; /*make process high priority */
   use(cpu, service_time); /*preempt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

Esempio proc. sharing

- Proc. sharing è una politica load dependent: ts=k(l) x d
  - di default k(l)=n: n=1 ts=d, n=2 ts=2d, n=3 ts=3d,….n=N ts=Nd
- Posso impostare una distribuzione diversa per k(l) ad esempio



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

Example: set_loaddep(f, rate, 10);

Rate=[1,1,1,3,3,3,5,5,5,7]

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	util.	Through put	Queue-lenght	Respons e time	Compl-count
1	fcfs	0.99	0.61289	24.77065	40.41622	55
2	fcfs	0.797	0.48723	1.52322	3.12628	43
3	fcfs	0.553	0.34752	0.77289	2.22405	31
4	fcfs	0.383	0.26336	0.42427	1.61101	24
5	fcfs	0.331	0.22552	0.35204	1.56101	21
1	rnd_rob	0.991	1.31493	31.75144	24.14695	118
2	rnd_rob	0.797	0.94047	1.92905	2.05115	83
3	rnd_rob	0.553	0.61656	0.93743	1.52041	55
4	rnd_rob	0.383	0.38406	0.49011	1.27612	35
5	rnd_rob	0.331	0.28996	0.38426	1.32523	27

FACILITY: statistics

Per collezionare le statistiche relative alla facility f per ogni classe di processi

Prototype: void collect_class_facility(FACILITY f)
Example: collect_class_facility(f);

Per collezionare le statistiche relative a tutte le facility per ogni classe di processi

Prototype: void collect_class_facility_all(void)
Example: collect_class_facility_all();

FACILITY: statistics

char* facility_name(FACILITY f) pointer to name of facility
long num_servers(FACILITY f) number of servers at facility
char* service_disp(FACILITY f) pointer to name of service discipline at facility
double timeslice(FACILITY f) time in each time-slice for facility (which has a round robinservice discipline)
long num_busy(FACILITY f) number of servers currently busy at facility
long qlength(FACILITY f) number of processes currently waiting at facility
long status(FACILITY f) current status of facility Busy if all servers are in use FREE if at least one server is not in use

FACILITY: statistics

long completions(FACILITY f) number of completions at facility
long preempts(FACILITY f) number of preempted requests at facility
double qlen(FACILITY f) mean queue length at facility
double resp(FACILITY f) mean response time at facility
double serv(FACILITY f) mean service time at facility
double tput(FACILITY f) mean throughput rate at facility
double util(FACILITY f) utilization (% of time busy) at facility

FACILITY: statistics

long server_completions(FACILITY f, long sn) number of completions for server *sn* at facility
double server_serv(FACILITY f, long sn) mean service time for server *sn* at facility
double server_tput(FACILITY f, long sn) mean throughput rate for server *sn* at facility
double server_util(FACILITY f, long sn) utilization for server *sn* at facility

FACILITY: statisics

- long class_qlength(FACILITY f, CLASS c) number of processes from class at facility
- long class_completions(FACILITY f, CLASS c) number of completions for class at facility
- double class_qlen(FACILITY f, CLASS c) mean queue length for class at facility
- double class_resp(FACILITY f, CLASS c) mean response time for class at facility
- double class_serv(FACILITY f, CLASS c) mean service time for class at facility
- double class_tput(FACILITY f, CLASS c) mean throughput rate for class at facility
- double class_util(FACILITY f, CLASS c) utilization for class at facility

Uso della priorità dei processi

- priority.c

risultati

Num. di visite al centro di servizio. Essendo la politica R-R il numero di visite è >> del numero di processi (58) che hanno visitato la facility httpd. Infatti ogni richiesta e' divisa in un numero di sottorichieste che è funzione della domanda di servizio e del timeslice

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

Num. Di processi nella classe i-esima.

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

Tutti i processi quando nascono appartengono alla classe default

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 */
...
```


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: Array of

```
#define NUM_STORES 5 /*set number of storage blocks in array*/
#define STORE_AMT 100 /*set amount of storage in each storage block*/
STORE mem[NUM_STORES]; /*declare storage block array*/
...
storage_set(mem, "mem", STORE_AMT, NUM_STORES); /*initialize stor mem with 100 units per block*/
...
amt = random(1, STORE_AMT); /*decide how much storage to allocate */
allocate(amt, mem[3]); /*get storage from the fourth storage block*/
...
deallocate(amt, mem[3]); /*release storage which is no longer needed*/
....
```


STORAGE: allocate with TimeOut

```
...
st = timed_allocate(amt, mem, 1.0); /*get storage if possible within 1 time unit */
if(st != TIMED_OUT) { /*if storage was gotten within the time limit */
...
deallocate(amt, mem); /*release storage which is no longer needed */
}
else { /* allocate timed out */
...
}
```


STORAGE: Synchronous

```
#define PHASE 0.5 /*set time to onset of first clock cycle to 0.5*/
#define PERIOD 1.0 /*set length of clock cycle to 1 time unit*/
#define STORE_AMT 100 /*set amount of storage in block to 100 units*/
STORE mem; /*declare storage variable mem */
mem = storage("mem", STORE_AMT); /*initialize storage named mem with 100 units*/
synchronous_storage(mem, PHASE, PERIOD); /*make storage allocations synchronous*/
...
allocate(5, mem); /*get 5 units of storage for this process */
...
deallocate(5, mem); /*release storage which is no longer needed*/
...
```

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

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

TABLE

Azzerate quando viene invocata una reset

Prototype: TABLE table (char* name);
Example: t = table ("response times");

No modificate quando viene invocata una reset

Prototype: TABLE permanent_table (char* name)
Example: t = permanent_table ("response times");

Prototype: void tabulate(TABLE t, double value)
Example: tabulate(t, 1.0);

Prototype: void report_table(TABLE t)
Example: report_table(t);

TABLE

TABLE 1: table

minimum	0.000016	mean	1.000040
maximum	10.336942	variance	0.999862
range	10.336926	standard deviation	0.999931
observations	10000	coefficient of var	0.999890

lower limit	frequency	proportion	cumulative proportion	
0.00000	6322	0.632200	0.632200	********************
1.00000	2288	0.228800	0.861000	*******
2.00000	878	0.087800	0.948800	***
3.00000	322	0.032200	0.981000	*
4.00000	112	0.011200	0.992200	
5.00000	48	0.004800	0.997000	
6.00000	22	0.002200	0.999200	
7.00000	5	0.000500	0.999700	
8.00000	1	0.000100	0.999800	
9.00000	1	0.000100	0.999900	
>= 10.00000	1	0.000100	1.000000	

TABLE: histogram

Prototype: void table_histogram(TABLE t, long nbucket, double min, double max)

```
TABLE tbl; /*declare table variable tbl */
    ...
    tbl = table("tbl"); /*initialize a table named tbl */
    table_histogram(tbl,10,0.0,20.0); /*add a historgram to a table named tbl */
    ...
    t = clock; /*get current time */
    reserve(single_server); /*reserve a single server facility */
    x = clock - t; /*calculate time spent on queue (delay interval)*/
    record(x, tbl); /*record delay interval in table */
    ...
```


Esempi

- Table.c
- Histogram.c