**Struttura di un Monitor**

Tipico esempio di un programma che usa i monitor.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*PROGRAMMA PRINCIPALE* ***HEADER.H***

*//sostituisce alla parola NUM\_COND il valore 2, rappresenta il numero di condizioni che userà il*

*//monitor generando un semaforo per ognuna di esse.*

**#define NUM\_COND 2**

*//sostituisce alle parole con numeri interi partendo da 0 (0,1 in questo caso), questi saranno //appunto i semafori generati dal monitor e che useremo per sincronizzare i processi.*

**enum**{ **OK\_CONSUMA**, **OK\_PRODUCI** };

*//struttura dati che utilizza il monitor e la nostra shared memory.*

**typedef struct**{

**int** dato; *//dato che viene prodotto e consumato .*

**int** ok\_consuma; *//serve per effettuare dei controlli all'accesso della shm.*

**int** ok\_produci;

**Monitor** monitor; *//gestione dell'accesso e utilizzo della shm.*

}**shm**;

*//procedure per la gestione e la sincronizzazione delle procedure Produttore() Consumatore().*

**void** inizioProduzione(**shm \***);

**void** fineProduzione(**shm \***);

**void** inizioConsumo(**shm \***);

**void** fineConsumo(s**hm \***);

*//operazione che eseguiranno i processi produttori e consumatori.*

**void** Produttore(**shm \*** );

**void** Consumatore(**shm \***);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

LIBRERIA ***MONITOR.H***

*//sostituisce alla parola MUTEX il valore 0.*

**#define MUTEX 0**

**#define URGENT\_SEM 0**

*//struttura Monitor utilizzata.*

**typedef struct** {

**int** id\_mutex; *//serve a creare un MUTEX (vedere in init\_monitor(); ).*

**int** urgent\_sem; *//semaforo per la coda urget dove verranno sospesi i processi.*

**int** id\_cond; *//id del gruppo semaforico associato alle variabili condition.*

**int** num\_var\_cond; *//numero di variabili condition.*

**int** \*cond\_count; *//array delle variabili conditions.*

**int** \*urgent\_count; //contatore numero processi sospesi sulla coda urgent(semplice //contatore).

**int** id\_shm; *//id memoria condivisa per i contatori delle variabili condition e della //coda urget.*

}**Monitor**;

**//Procedure per gestire il monitor.**

*//inizializza il monitor.*

**void** init\_monitor(**Monitor\***, **int**);

*//entrata dei processi nel monitor.*

**void** enter\_monitor(**Monitor\***);

*//uscita processi dal monitor.*

**void** leave\_monitor(**Monitor\***);

*//rimozione del monitor.*

**void** remove\_monitor(**Monitor\***);

*//gestione del monitor.*

**void** signal\_condition(**Monitor\***);

**void** wait\_condition(**Monitor\***);

*//procedure gestione semafori interni al monitor.*

**void** signal\_sem(**int**,**int**);

**void** wait\_sem(**int**,**int**);

//restituisce il numero di processi presenti sospesi sulle varaibili condition passata in input

**int** queue\_condition(**Monitor\*,int**);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***MONITOR.C*** (implementazione delle procedure della libreria del MONITOR.H).

**void** init\_monitor( **Monitor** \*monitor, **int** num\_cond){

**int** i;

*//assegno un id di una struttura semaforica per gestire l'accesso al monitor (MUTEX).*

monitor→id\_mutex = **semget**( IPC\_PRIVATE, 1, IPC\_CREAT | 0664);

**semctl**( monitor→id\_mutex, **MUTEX**, **SETVAL**, 1);

*//alloca e inizializza il semaforo per la coda urgent dove verranno sospesi i processi //inizialmente è 0 perché non vi sono processi sospesi.*

monitor→urget\_sem = semget(IPC\_PRIVATE, 1, IPC\_CREAT | 0664 );

**semctl**( monitor→urgent\_sem, **URGENT\_SEM**, **SETVAL**, 0);

*//gruppo di semafori con cui realizzare le variabili condition.*

monitor→id\_cond = **semget**( IPC\_PRIVATE, num\_cond, IPC\_CREAT | 0664);

**for**(i=0; i<num\_cond; i++){

**semctl**(monitor→id\_cond, i, **SETVAL**,0);

}

*//creo una memoria, alloco un contatore per ogni variabile condition, più un contatore per //la coda urgent.*

monitor→id\_shm = shmget(IPC\_PRIVATE,(num\_cond+1)\***sizeof**(**int**),IPC\_CREAT0664);

*//effettuo attach dell'array di contatori appena allocato.*

monitor→cond\_count = (**int\***)**shmat**(monitor→id\_shm, 0, 0);

monitor→num\_var\_cond = num\_cond;

*//stiamo dando una dimensione in baytes a urgent\_count.*

monitor→urgent\_count = (monitor→cond\_count )+( monitor->num\_var\_cond);

*//inizializzo i contatori per le variabili condition e per la coda urgent.*

**for**(i=0; i<num\_cond; i++){

monitor→cond\_count[i] = 0;

}

*//assegno il valore 0, al contatore di nuomero di rocessi sulla coda urgent.*

\*(monitor->urgent\_count) = 0;

}

//restituisce il numero di processi presenti sospesi sulle varaibili condition passata in input

**int** queue\_condition(**Monitor \***monitor**,int** num\_sem){

//restituisce il numero di processi sospesi sul semaforo num\_sem del vettore

**return**( monitor->cond\_count[num\_sem]);

}

*//viene invocata all'ingresso del monitor, procedura che serve a entrare in mutua esclusione* *//all'interno del monitor.*

**void** enter\_monitor(**Monitor** \*monitor){

*//il processo entra nel monitor non appena è disponibile, mediante la wait sul MUTEX di //entrata.*

**wait\_sem**(monitor→id\_mutex, **MUTEX**);

}

*//viene invocata all'uscita del monitor, procedura che serve a uscire dall'interno del monitor.*

**void** leave\_monitor(**Monitor** \*monitor){

*//se il contatntore a urgent è maggiore di 0 allora vi sono processi sospesi in attesa di una //signal sulla coda urgent.*

**if**( \*(monitor→urgent\_count) > 0 ){

signal\_sem(monitor→urgent\_sem, **URGENT\_SEM**);

}

**else**{

*//nel momenti in cui la condizione è falsa allora posso uscire dal monitor facendo //una signal sul MUTEX.*

**signal\_sem**(monitor→id\_mutex, **MUTEX**);

}

}

*//viene invocata alla rimozione del monitor, rimuove tutte le strutture creata.*

**void** remove\_monitor(**Monitor** \*monitor){

//rimuovo le tre strutture semaforiche, e la memoria condivisa.

**semctl**(monitor→id\_mutex, **MUTEX**, IPC\_RMID);

**semctl**(monitor→urgent\_sem, **URGENT\_SEM**, IPC\_RMID);

//semafori sulle variabili condition.

**semctl**(monitor→id\_cond, 0, IPC\_RMID);

**shmctl**(monitor→id\_shm, IPC\_RMID, 0);

}

*//PER LA GESTIONE DEI SEMAFORI INTERNI AL MONITOR (1) (2)*

**void** wait\_sem(**int** id\_sem, **int** numero\_sem){ *//(1)*

**struct** sembuf sem;

sem.sem\_num = numero\_sem;

sem.sem\_flg = 0;

sem.sem\_op = -1;

**semop**(id\_sem, &sem, 1); *//semaforo rosso*

}

**void** signal\_sem(**int** id\_sem, **int** numero\_sem){ *//(2)*

**struct** sembuf sem;

sem.sem\_num = numero\_sem;

sem.sem\_flg = 0;

sem.sem\_op = -1;

**semop**(id\_sem, &sem, 1); *//semaforo verde*

}

*//serve a riprendere un processo in esecuzione dentro il monitor, che attende una signal su una //condition (num\_sem sarebbe la condition OK\_CONSUMA, OK\_PRODUCI).*

**void** wait\_condition(**Monitor** \*monitor, **int** num\_sem){

*//è un controllo per vedere se il semaforo passato come parametro esiste.*

**if**(num\_sem < 0 || num\_sem >= monitor->num\_var\_cond){

**perror**(“errore invocazione della wait \n ”);

}

*//incremento di 1 il valore presente nel semaforo nella posizione del nostro array coditions.*

monitor→cond\_count[num\_sem] = monitor->cond\_count[num\_sem]+1;

*//vado a controllare se sulla coda urgent è presente qualche processo, se si gli mando una //signal per riattivarlo.*

**if**( \*(monitor→urgent\_conut) > 0 ){

**signal\_sem**( monitor→urgent\_sem, **URGENT\_SEM** );

}

**else**{

*// altrimenti faccio entrare un nuovo processo con signal MUTEX.*

**signal\_sem**(monitor→id\_mutex, **MUTEX** );

}

*//attendo una signal sul semaforo num\_sem passato come parametro, da parte di un //processo che si trova in coda oppure che deve entrare ancora nel monitor e invocherà una //signal che riattiverà il mio processo.*

**wait\_sem**(monitor→id\_cond, num\_sem);

*//decremento il valore sull'array cond\_count nell'indice num\_sem passato come parametro, //che porta il conteggio dei processi sospesi su quel semaforo (num\_sem).*

monitor→cond\_count[num\_sem] = monitor→cond\_count[num\_sem] -1;

}

//*serve a sospendere un processo e metterlo in coda.*

**void** signal\_condition(**Monitor** \*monitor, **int** num\_sem){

*//è un controllo per vedere se il semaforo passato come parametro esiste.*

**if**(num\_sem < 0 || num\_sem >= monitor->num\_var\_cond){

**perror**(“errore invocazione della signal \n ”);

}

*//quando invoco la signal sospendo il processo, e va nella coda urgent, allora incremento il //contatore di processi sulla coda urgent.*

\*(monitor→urgent\_conut)++;

*//se vi sono processi in attesa al semaforo num\_sem (OK\_CONSUMA…), mi sospendo e //mando una signal su num\_sem per attivarli.*

**if**( monitor→cond\_count[num\_sem] > 0 ){

**signal\_sem**(monitor→id\_cond, num\_sem);

*//attendo sul semaforo urgent una wait per continuare*

**wait\_sem**(monitor→urgent\_sem, URGENT\_SEM);

}

*//il processo passa dalla coda al monitor, decremento il conteggio sulla cosa urgent*

\*(monitor→urgent\_conut)--;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

HEADER.C

//inclusione delle librerie

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <unistd.h>

**#include** <sys/ipc.h>

**#include** <sys/types.h>

**#include** <sys/wait.h>

**#include** <sys/shm.h>

**#include** <sys/sem.h>

**#include** "monitor.h"

**#include** "header.h"

//processo poduttore e consumatore eseguiranno queste procedure.

**void** produttore(**shm** \*memoria){

**inizioProduzione**(memoria);

//genera un numero casuale da 0 a 33 e lo inserisce nella mia shm.

memoria→dato = rand()%33;

**printf**(“Dato prodotto <%d> \n”,memoria->dato);

**fineProduzione**(memoria);

}

**void** consumatore(shm \*memoria){

**inizioConsumo**(memoria);

**printf**(“Dato letto <%d> \n”,memoria->dato);

**fineConsumo**(memoria);

}

//procedure di gestione del produttore inizio produzione

**void** inizioProduzione(**shm** \*memoria){

//effettuo acceso nel monitor

**enter\_monitor**( &(memoria->monitor) );

//se la varaibile della shm ok\_produci = 1 allora significa che devo produrre un dato, //se uguale a 0 significa che devo attendere che il dato venga consumato quindi faccio la //wait sul semaforo OK\_PRODUCI che aspetto una signal da parte del consumatore.

**if**( memoria->ok\_produci == 0 ){

**wait\_condition**( &(memoria->monitor), **OK\_PRODUCI**);

}

//da questo punto in poi sono nel monitor lo lascero quando invocherò fineProduzione()

**printf**("Entro nel monitor, Produttore <%d>\n", getpid());

}

//procedura di fine produzione

**void** fineProduzione(**shm** \*memoria){

**printf**("Esco dal monitor, Produttore <%d>\n", getpid());

//cambio i valore delle variabili della shm

memoria->ok\_produci = 0;

memoria->ok\_consuma = 1;

//avvetrto i consumatori con signal su OK\_CONSUMA e abbandono il monitor

**signal\_condition**( &(memoria->monitor), **OK\_CONSUMA**);

**leave\_monitor**( &(memoria->monitor) );

}

//procedure di gestione del consumatore inizio della consumazione

**void** inizioConsumazione(shm \*memoria){

//effettuo l'accesso al monitor

**enter\_monitor**( &(memoria->monitor) );

//controllo se la variabile ok\_consuma = 1 significa che posso consumare altrimenti attendo //il segnale su OK\_CONSUMA fatto dal produttore

**if**(memoria->ok\_consuma == 0){

**wait\_condition**( &(memoria->monitor), **OK\_CONSUMA** );

}

//da questo punto in poi sono nel monitor dove uscirò a fineConsumazione()

**printf**("Entro nel monitor, Consumatore <%d>\n", getpid());

}

**void** fineConsumazione(**shm** \*memoria){

**printf**("Esco dal monitor, Consumatore <%d>\n", getpid());

//cambio i valore delle variabili della shm

memoria->ok\_consuma = 0;

memoria->ok\_produci = 1;

//segnalo che ho consumato su OK\_PRODUZIONE e abbandono il monitor

**signal\_condition**( &(memoria->monitor), **OK\_PRODUCI**);

**leave\_monitor**( &(memoria->monitor) );

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

MAIN.C (main del programma principale)

//inclusione delle librerie

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <unistd.h>

**#include** <time.h>

**#include** <sys/ipc.h>

**#include** <sys/types.h>

**#include** <sys/wait.h>

**#include** <sys/shm.h>

**#include** <sys/sem.h>

**#include** "monitor.h"

**#include** "header.h"

**int** main(){

//dichiarazione delle variabili

**pid\_t** pid;

**shm** \*memoria;

**int** id\_shm;

**int** status;

**int** i,k;

**printf**("\n\n \_\_INIZIO\_\_ \n\n");

//creo segmento memoria condivisa

id\_shm = **shmget**(**IPC\_PRIVATE**, **sizeof**(**shm**), **IPC\_CREAT** | 0664);

memoria = (**shm**\*)**shmat**(id\_shm, 0, 0);

memoria->ok\_consuma = 0;

memoria->ok\_produci = 1;

memoria->dato = 0;

//inizializzo il monitor

**init\_monitor**( &(memoria->monitor), **NUM\_CONDITIONS**);

//genero i processi

**for**(i=0; i<**NUM\_PROC**; i++){

pid = **fork()**;

**srand**(**time**(NULL)^**getpid()**);

**if**(pid == 0){

**if**( i%2 == 0){

//Produttore

**printf**("Sono PRODUTTORE <%d>\n",**getpid()**);

**produttore**(memoria);

}**else**{

//Consumatore

**printf**("Sono CONSUMATORE <%d>\n",**getpid()**);

**consumatore**(memoria);

}

**exit**(**0**);

}

}

//attendo che terminano tutti i processi

**for**(i=0; i<**NUM\_PROC**; i++){

**pid** = **wait**( &status);

**printf**("TERMINATO <%d> status <%d>\n", pid, status);

}

//rimuovo monitor e memoria condivisa

**shmctl**(id\_shm, **IPC\_RMID**, 0);

**remove\_monitor**( &(memoria->monitor) );

**printf**("\n\n \_\_FINE\_\_ \n\n");

**return 0**;

}