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(shm *);

//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_CREAT 0664);

	//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

```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)

```c
//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;
}
```