Sapienza

Università di Roma

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

anno accademico 2007-2008

Tesina di Sistemi Operativi I

Prof. Andrea Santoro (Canale A-L)

Bacheca Elettronica Volatile

di:
Pier Paolo Ciarravano
Matr. 773970

Indice:

Introduzione e specifiche del progetto:

Nell'ambito del corso di "Sistemi Operativi I" di Ing.Informatica (A.A. 2007-2008), si richiede la progettazione e la realizzazione di un software in grado di gestire le funzionalità di una bacheca elettronica, su Sistema Operativo UNIX/LINUX.

Più precisamente si richiede la realizzazione di una bacheca elettronica residente su memoria condivisa.

Una bacheca elettronica e' un servizio che permette ad ogni utente del sistema di inviare messaggi che possono essere letti da un qualsiasi utente del sistema interessato a consultare la bacheca stessa.
In questo caso la bacheca viene attivata da uno specifico programma il cui compito e' creare la memoria condivisa e le necessarie strutture di sincronizzazione.

Il programma per accedere alla bacheca deve fornire ad un utente interattivo un menù con le seguenti funzioni:

1. Elenco di tutti i messaggi sulla bacheca elettronica.
2. Lettura di uno specifico messaggio della bacheca elettronica.
3. Spedizione di un nuovo messaggio sulla bacheca elettronica.
4. Rimozione di un qualsiasi messaggio dalla bacheca elettronica.

Un messaggio deve contenere almeno i campi Oggetto e Testo.
Si raccomanda di usare i semafori per evitare che due o più istanze del programma di accesso generino inconsistenza dei dati sulla memoria condivisa.
Si richiede di realizzare sia il programma di attivazione che il programma per la lettura/invio/cancellazione dei messaggi, ovvero il programma di accesso vero e proprio.

Scelte realizzative e tecniche usate:

Il Sistema Operativo UNIX mette a disposizione, tra numerosi metodi di comunicazione tra processi, un metodo per accedere ad aree di memoria in modo condiviso tra processi.
Un'area di memoria condivisa è una porzione di memoria accessibile allo stesso tempo da più processi. Proprio questa sua caratteristica di accessibilità a più processi nello stesso tempo, da un lato la rende un'ottima metodologia per far comunicare i processi, dall'altro la rende estremamente delicata per tutte quelle che sono le problematiche di accesso contemporaneo in lettura e scrittura da parte dei vari processi che condividono la zona di memoria. Per evitare inconsistenza dei dati nella memoria bisogna gestire la sincronizzazione dei processi per l'accesso alla memoria condivisa, specie per quanto riguarda i processi che agiscono modifiche sulla memoria.

Nella realizzazione della Bacheca Elettronica Volatile si richiede da specifiche che la comunicazione tra i processi avvenga attraverso un'area di memoria condivisa. Pertanto, per quanto riguarda tutte quelle che sono le problematiche di sincronizzazione tra i processi, si sono utilizzati semafori nell'implementazione System V di UNIX.

Il problema di sincronizzazione, che entra in gioco nella realizzazione della Bacheca Elettronica Volatile, è un classico problema di sincronizzazione di processi Lettori-Scrittori.
In questo contesto, supponendo di avere una sezione critica per i processi di lettura e scrittura, abbiamo i seguenti casi:

1. Ogni processo lettore può trovarsi nella sezione critica di lettura simultaneamente.
2. Gli scrittori devo avere accesso esclusivo alla sezione critica di scrittura, bloccando ogni altro processo che sta effettuando la lettura, fino a quando l'operazione di scrittura non sia terminata.

In altre parole, uno scrittore non può accedere alla modifica della memoria, fintanto che ogni altro processo legge o scrive la memoria, e quando lo scrittore accede alla memoria condivisa, nessun altro processo può accedervi, sia in lettura che in scrittura.
Questo metodo di accesso è chiamato "categorical mutual exclusion": ogni processo nella sezione critica, infatti, non necessariamente esclude gli altri (nel nostro caso uno scrittore escude ogni altro scrittore), ma la presenza di una "categoria" (lettori o scrittori) nella sezione critica, esclude la presenza delle altre "categorie".

Per implementare la comunicazione attraverso memoria condivisa, si è scelto di condividere un'area di memoria contenente una struttura dati così definita:

```
typedef struct s_bulettinboard {
                        int max_messages;
                        int idSequence;
                        message_bb messages[];
                } bulettinboard;
```

Tale struttura dati contiene a sua volta un array di strutture di tipo message_bb così definite:

```
typedef struct s_message_bb {
               int id;
               int isFree;
               char title[100];
               char text[2048];
               char userLogin[20];
               char email[100];
               struct tm insertDate; //time inserimento
       } message_bb;
```

La struttura s_bulettinboard memorizza al suo interno il numero di elementi dell'array stesso nella variabile max_messages e una variabile idSequence utilizzata per assegnare un identificativo univoco ad ogni messaggio.
La struttura s_message_bb, oltre a contenere come da specifiche:

- una variabile per la descrizione testuale per l'oggetto del messaggio (campo title),
- una variabile per il campo testo (campo text)

contiene:

- id: id univoco del messaggio,
- isFree: permette di indicare se l'elemento contiene un messaggio valido oppure no,
- userLogin: contiene la username unix/linux dell'utente che ha inserito il messaggio,
- email: campo email dell'utente che ha inserito il messaggio,
- insertDate: campo data e ora, per permettere il tracciamento temporale del messaggio.

La struttura dati così definita è facilmente gestibile permettendo alle varie funzioni del software di accedere in modo "strutturato" alla lettura e scrittura nella memoria stessa.
Si è scelto pertanto di creare un'istanza della struttura s_bulettinboard e di inizializzare l'array message_bb al suo interno in fase di creazione e inizializzazione della memoria condivisa, specificando la lunghezza dell'array stesso.
In questo modo è facile individuare quanto occuperà l'area della memoria condivisa:

```
 = 2*sizeof(int) + MAX_NUMERO_MESSAGGI*sizeof(message_bb)
```

In fase di creazione della memoria condivisa, viene creata una memoria di queste dimensioni e attaccata allo spazio di indirizzamento del processo attraverso la sua chiave identificativa intera positiva, che può essere specificata come parametro o semplicemente impostata come default.

All'atto del collegamento della memoria condivisa allo spazio di indirizzamento del processo, si otterrà l'inizializzazione di un puntatore così definito:

```
bulettinboard *MBULLETTINBOARD
```

che permetterà facilmente l'accesso, la modifica e la lettura dei messaggi presenti in bacheca. Più precisamente attraverso (*MBULLETTINBOARD) possiamo accedere tramite l'operatore "." a tutti i campi della struttura s_bulletinboard e attraverso la variabile

(*MBULLETTINBOARD).messages[i] possiamo accedere all'i-esimo messaggio contenuto nella memoria condivisa.

Si è scelto di individuare i messaggi non tramite l'indice dell'array, ma tramite un identificativo univoco crescente, per permettere che un utente che voglia leggere un determinato messaggio lo identifichi attraverso il suo identificativo univoco. Identificando altrimenti il messaggio solo tramite l'indice nell'array messages, si sarebbe potuto verificare il caso di un utente che, volendo leggere il messaggio di indice I, si potrebbe trovare a leggere un messaggio non più contenente quello che lui credeva, ma un nuovo messaggio che, per la scelta di un altro utente, pochi istanti prima lo ha rimpiazzato cancellandolo. Questa scelta, d'altro canto, costringe la visita di tutto l'array ogni qual volta un utente sceglie di leggere un certo messaggio, ma tale visita ha complessità O(N) con N=MAX_NUMERO_MESSAGGI, ed e' pertanto trascurabile nel nostro problema.
Si è scelto inoltre, nell'azione di rimozione di un messaggio nell'array condiviso, di eliminare l'elemento che lascia vuoto, spostando tutti gli elementi di seguito di una posizione in avanti, così da non lasciare "buchi" nell'array dei messaggi e avere sempre gli identificativi univoci id ordinati in maniera crescente al crescere dell'indice dell'array.

Per quanto riguarda la sincronizzazione all'accesso in lettura e scrittura alla memoria condivisa, sono state individuate due "categorie" di sezioni critiche: una per le parti di codice che effettuano la lettura e una per le parti di codice che effettuano la scrittura.
Per risolvere questa problematica è stato creato un array di semafori di 2 elementi, il primo semaforo inizializzato al numero massimo di sezioni critiche contemporanee in lettura e il secondo semaforo inizializzato a 1, per permettere l'accesso esclusivo alla sezione critica di scrittura sulla memoria condivisa.

Utilizzando le primitive di accesso ai semafori wait e signal si illustra di seguito la problematica di accesso alle sezioni critiche che è stata cosi risolta:

Inizializzazione semafori:

```
SEMAPHORE_READERS = MAX_NUM_READERS_CONTEMPORANEI;
SAMAPHORE_WRITERS = 1;
```

Accesso alla sezione critica per READERS:

```
wait(SEMAPHORE_READERS);
      <SEZIONE CRITICA DI LETTURA DELLA MEMORIA CONDIVISA>
signal(SEMAPHORE_READERS)
```

Accesso alla sezione critica per WRITERS:

```
wait(SAMAPHORE_WRITERS); //Mutua esclusione tra tutti i processi di scrittura
      for(1 to MAX_NUM_READERS_CONTEMPORANEI)
            wait(SEMAPHORE_READERS); //Blocca tutti i processi in lettura
                  <SEZIONE CRITICA DI SCRITTURA DELLA MEMORIA CONDIVISA>
      for(1 to MAX_NUM_READERS_CONTEMPORANEI)
            signal(SEMAPHORE_READERS);
signal(SAMAPHORE_WRITERS);
```

Come già illustrato precedentemente, si è scelto di implementare le primitive wait e signal attraverso i semafori nella loro implementazione della versione System V di UNIX.

Si è scelto di ignorare alcuni segnali del sistema operativo verso il processo nel momento dell'accesso alle sezioni critiche di lettura o scrittura su memoria condivisa e di riattivare il comportamento di default di questi segnali all'uscita dalla sezioni critiche; questo per evitare al minimo i casi in cui un processo venga interrotto prima di effettuare un'operazione di signal su un semaforo, in modo da evitare deadlock di un processo che richieda il semaforo non rilasciato.
I segnali ignorati nelle sezione critiche di scrittura e lettura sono:

- `SIGINT`
- `SIGTERM`
- `SIGHUP`
- `SIGQUIT`

Si è scelto di operare una realizzazione quanto più possibile strutturata attraverso funzioni che permettono la lettura e la scrittura sulla memoria condivisa in modo semplice.
In questo modo è stato possibile testare attraverso "test unitari" le singole funzioni realizzate, implementando anche simulazioni di deadlock attraverso una funzione di attesa, creata solo per il test.
Sono state realizzate numerose funzioni, (anche non tutte utilizzate dal programma di gestione, ma implementate comunque per un loro possibile uso futuro) di seguito descriviamo le più importanti, facendo riferimento al sorgente VolatileBulletinBoard.h riportato in fondo a questa relazione per ogni altro dettaglio funzionale.

Operazioni di accesso in lettura e scrittura alla memoria condivisa richieste da specifiche:

```
bulettinboard *list_bb(); //Ritorna una lista di messaggi clone
message_bb readMessage_bb(int idMessage); //Legge uno specifico messaggio
int writeMessage_bb(message_bb message); //Spedisce un messaggio
int removeMessage_bb(int idMessage); //Cancella un messaggio
```

Inoltre è stata creata una funzione aggiuntiva che permette anche la modifica di un messaggio, pur non essendo stata poi inserita questa funzionalità all'interno del menù interattivo:

```
int updateMessage_bb(message_bb message); //Modifica un messaggio
```

Per le operazioni di creazione della memoria condivisa, dei semafori e la loro rispettiva inizializzazione, collegamento allo spazio di indirizzamento del processo, rimozione e scollegamento, sono state realizzate le seguenti funzioni:

```
//Creazione e inizializzazione memoria condivisa e semafori
int init_bb_arg(long int sk, int long mk, int mr, int mm);
int init_bb();

//Rimozione memoria condivisa e semafori
int destroy_bb_arg(long int sk, int long mk);
int destroy_bb();
int destroyUsingId_bb();
```

```
//Apertura semafori e collegamento spaz.ind.processo memoria condivisa esistente
int open_bb_arg(long int sk, int long mk);
int open_bb();

//Detach memoria condivisa
int detachSharedMemory();
```

Sono state implementate altre due funzioni (non richieste da specifiche) che permettono il salvataggio della memoria condivisa su file e il successivo ripristino da file a memoria condivisa, utilizzando le comuni funzioni standard di UNIX per i file e richiamabili tutte tramite parametri da riga di comando (vedere manuale d'uso), in modo da permettere il salvataggio dei messaggi della bacheca prima dello shutdown del sistema, e il suo ripristino successivamente al riavvio; le funzioni sono:

```
int fileBackupMessage_bb(char *filename); //OPERAZIONE BACKUP MEMORIA COND.
int fileRestoreMessage_bb(char *filename); //OPERAZIONE RIPRISTINO MEMORIA COND.
```

All'atto dell'inizializzazione della bacheca, l'utente può scegliere le chiavi intere positive della memoria condivisa e dell'array di semafori, specificando questi valori da parametri sulla linea di comando, oppure scegliere di usare valori di default per queste chiavi.
Specificando in modo diverso queste chiavi, si ha la possibilità di gestire diverse bacheche elettroniche condivise e indipendenti.
Inoltre l'utente in fase di inizializzazione bacheca può scegliere anche il massimo numero di messaggi contenuti nella bacheca e il massimo numero di processi che accedono alle sezioni critiche di lettura, oppure usare valori di default per questi parametri.
La gestione dell'input dei parametri da linea di comando è stata eseguita utilizzando la funzione GNU "getopt" di <unistd.h>.

Si è scelto, per maggiore funzionalità, di implementare la creazione e inizializzazione memoria condivisa e semafori, rimozione e gestione interattiva dei messaggi della bacheca in un unico programma, specificando le diverse possibili azioni attraverso il passaggio di parametri da linea di comando (vedere manuale d'uso). Come da specifiche però è rilasciata anche una versione con le sole funzioni di gestione interattiva dei messaggi della bacheca, così da rilasciare questa versione a soli utenti che non hanno permessi di creazione-rimozione della bacheca.

La gestione interattiva della bacheca avviene attraverso una modalità testuale, utilizzando le comuni funzioni di <stdio.h> in una modalità simile a diversi programmi di gestione del sistema operativo Linux, come per esempio "fdisk". Viene presentato un menù di scelta e tramite la selezione di un'azione nel menù, è possibile effettuare l'operazione indicata.
Ad ogni interazione utente è sempre effettuato un controllo formale sulla validità dei dati inseriti dall'utente, come la lunghezza massima dei campi o il controllo su inserimenti vuoti o stringhe al posto di numeri e viceversa.

Tutte le parti sono state sviluppate, compilate e testate su sistema operativo LINUX Fedora 9 e compilatore GNU GCC versione 4.3.0, inoltre per la realizzazione del codice è stato utilizzato l'IDE Eclipse "Ganymede" 3.4.0 con plug-in CDT 5.0 per sviluppo in C/C++.

Manuale d'uso:

Compilazione:

Per procedere all'istallazione e compilazione procedere come di seguito riportato.
Scompattare il file VolatileBulletinBoard.tgz usando la seguente istruzione:

```
tar –xzf VolatileBulletinBoard.tgz
```

Verrà creata una cartella VolatileBulletinBoard contenente i sorgenti dell'applicazione, questo file di documentazione in pdf, e i relativi file per la compilazione in batch.

Per compilare l'applicazione si può procedere nei seguenti modi:

1. Se si ha l'utility make, si può semplicemente digitare:

   ```
   cd VolatileBulletinBoard
   make
   ```

2. Lanciare la procedura batch di compilazione digitando:

   ```
   cd VolatileBulletinBoard
   chmod +x compile.sh
   ./compile.sh
   ```

3. Procedere alla compilazione manuale digitando:

```
cd VolatileBulletinBoard

gcc -O3 -Wall -c -fmessage-length=0 -MMD -MP \
-MF"src/VolatileBulletinBoardFunctions.d" - \
MT"src/VolatileBulletinBoardFunctions.d" \
-o"src/VolatileBulletinBoardFunctions.o" "src/VolatileBulletinBoardFunctions.c"

gcc -O3 -Wall -c -fmessage-length=0 -MMD -MP -MF"src/VolatileBulletinBoard.d" \
-MT"src/VolatileBulletinBoard.d" -o"src/VolatileBulletinBoard.o" \
"src/VolatileBulletinBoard.c"

gcc -O3 -Wall -c -fmessage-length=0 -MMD -MP \
-MF"src/VolatileBulletinBoardUsers.d" -MT"src/VolatileBulletinBoardUsers.d" \
-o"src/VolatileBulletinBoardUsers.o" "src/VolatileBulletinBoardUsers.c"

gcc  -o"VolatileBulletinBoard" ./src/VolatileBulletinBoardFunctions.o \
./src/VolatileBulletinBoard.o

gcc  -o"VolatileBulletinBoardUsers" ./src/VolatileBulletinBoardFunctions.o \
./src/VolatileBulletinBoardUsers.o
```

In tutti e tre i metodi verrano creati i due file esecutivi:

1. VolatileBulletinBoard : per la creazione, l'inizializzazione, la rimozione della memoria condivisa, dei semafori, le operazioni di backup e ripristino della memoria da file e la gestione interattiva della bacheca elettronica.
2. VolatileBulletinBoardUsers : per la sola gestione interattiva della bacheca Elettronica.

Utilizzo:

Programma "VolatileBulletinBoard":

Lanciando il programma con il parametro –h passato da linea di comando si visualizza il seguente manuale d'uso dei parametri per questo programma:

VolatileBulletinBoard [-a|-d|-b|-l|-t] [OPZIONI]
OPZIONI:

-a	Creazione Bacheca: memoria condivisa e semafori
-d	Rimozione Bacheca: memoria condivisa e semafori
-s N	setta chiave identificazione semafori
-m N	setta chiave identificazione memoria condivisa
-r N	massimo numero di utenti readers
-n N	massimo numero di messaggi della bacheca
-b file	backup memoria condivisa su file
-l file	carica memoria condivisa da file backup
-h	Visualizza questo help e termina il programma

N: rappresenta un intero positivo

Non usando nessuno dei parametri -a, -d, -b, -l viene eseguito il programma di gestione della bacheca in modo interattivo.
Non usando i parametri -s, -m, -r, -n vengono usati i valori di default.
I parametri -r e -n vengono usati solamente insieme al parametro -a nella creazione bacheca, altrimenti vengono ignorati.

Programma "VolatileBulletinBoardUsers":

Lanciando il programma con il parametro –h passato da linea di comando si visualizza il seguente manuale d'uso dei parametri per questo programma:

VolatileBulletinBoard [OPZIONI]
OPZIONI:

-s N	setta chiave identificazione semafori
-m N	setta chiave identificazione memoria condivisa
-h	Visualizza questo help e termina il programma

N: rappresenta un intero positivo

Non usando i parametri -s, -m vengono usati i valori di default.

Il programma esegue la gestione interattiva della bacheca elettronica cosi come il programma VolatileBulletinBoard lanciato senza i parametri -a, -d, -b, -l.

Alla partenza del programma in modalità gestione interattiva, viene presentato un menù per la scelta dell'azione da compiere, in ogni momento comunque richiamabile digitando la lettera "h".

I comandi presentati nel menù utente sono i seguenti:

- l: Lista messaggi presenti in bacheca
- r: Lettura messaggio in bacheca
- a: Spedizione nuovo messaggio in bacheca
- d: Cancellazione messaggio in bacheca
- h: Visualizza questo aiuto comandi
- q: Uscita dalla gestione bacheca

Ad ogni selezione di un comando, l'utente viene seguito in modo intuitivo nella scelta degli eventuali dati in input da inserire o nelle azioni da compiere, attraverso messaggi che spiegano cosa fare.

File: VolatileBulletinBoard.h

```
 1 /*
 2  ============================================================================
 3  Name        : VolatileBulletinBoard.h
 4  Author      : Pier Paolo Ciarravano
 5  Version     : 1.0 - 15/09/2008
 6  Copyright   : GPL - General Public License
 7  Description : Bacheca Elettronica Volatile in C, Ansi-style
 8                Corso di Sistemi Operativi I - Prof. A.Santoro 2007-2008
 9                Ingegneria Informatica - Univ. "La Sapienza" Roma
10  ============================================================================
11  */
12
13 #ifndef VOLATILEBULLETINBOARD_H_
14 #define VOLATILEBULLETINBOARD_H_
15
16 #include <stdio.h>
17 #include <stdlib.h>
18 #include <string.h>
19 #include <unistd.h>
20 #include <time.h>
21 #include <sys/types.h>
22 #include <sys/ipc.h>
23 #include <sys/shm.h>
24 #include <sys/sem.h>
25 #include <pwd.h>
26 #include <signal.h>
27 #include <fcntl.h>
28
29 //Struttura dei messaggi
30 typedef struct s_message_bb {
31     int id;
32     int isFree;
33     char title[100];
34     char text[2048];
35     char userLogin[20];
36     char email[100];
37     struct tm insertDate; //time inserimento
38   } message_bb;
39
40 //Struttura formata da l'array di strutture dei messaggi e numero di elementi
   dell'array
41 typedef struct s_bulettinboard {
42       int max_messages;
43       int idSequence;
44       message_bb messages[];
45     } bulettinboard;
46
47 //puntatore alla struttura bulettinboard, puntata dalla memoria condivisa
48 bulettinboard *MBULLETTINBOARD;
49
50 //variabili per valori di default
51 long int SEMAPHORE_KEY;
52 long int SMEMORY_KEY;
53 int MAX_READERS;
54 int MAX_MESSAGES;
55
56 //Setta valori di default
57 void setDefaultValue();
58
59 //Identificatore array di semafori: 0 semaforo READERS (init:MAX_READERS), 1
   semaforo WRITERS (init:1 - mutex)
60 int ID_SEMAPHORE_RW;
61
62 //Identificatore memoria condivisa
63 int ID_SMEMORY;
64
65 //Parser degli eventuali parametri da linea di comando e inizializzazione delle
   variabili
66 int parseMainOptions_bb(int argc, char *argv[]);
67
68 //creazione e inizializzazione memoria condivisa e semafori, utilizzato dal
   programma di attivazione
69 //  ritorna:
70 //  0:ok
```

File: VolatileBulletinBoard.h

```
71  //  -1:errore creazione memoria condivisa
72  //  -2:errore creazione semafori
73  int init_bb_arg(long int sk, int long mk, int mr, int mm);
74  //Utilizza valori di default SEMAPHORE_KEY, SMEMORY_KEY, MAX_READERS, MAX_MESSAGES
75  int init_bb();
76
77  //rimuove memoria condivisa e semafori
78  int destroy_bb_arg(long int sk, int long mk);
79  //utilizza valori di default SEMAPHORE_KEY, SMEMORY_KEY
80  int destroy_bb();
81  //utilizza ID_SMEMORY, ID_SEMAPHORE_RW per rimuovere memoria condivisa e semafori
82  //FUNZIONE NON USATA MA IMPLEMENTATA COMUNQUE
83  int destroyUsingId_bb();
84
85  //apertura memoria condivisa e semafori, utilizzato dal programma operante sulla
    bacheca (lettore/scrittore),
86  // inizializza ID_SMEMORY, ID_SEMAPHORE_RW e MAX_MESSAGES leggendolo dalla memoria
    condivisa,
87  // e collega la memoria condivisa al puntatore della struttura di tipo
    bulettinboard: MBULLETTINBOARD
88  int open_bb_arg(long int sk, int long mk);
89  //utilizza i valori di default SEMAPHORE_KEY, SMEMORY_KEY
90  int open_bb();
91
92  //Scollega la memoria condivisa puntata da *MBULLETTINBOARD dallo spazio di
    indirizzamento del processo
93  int detachSharedMemory();
94
95  //Lista i messaggi nella bacheca ritornando un puntatore alla struttura di tipo
    bulettinboard,
96  // che contiene l'array messages clonato con i messaggi presenti nella memoria
    condivisa,
97  // per prevenire eventuali effetti collaterali sulla modifica dei messaggi, in una
    parte di codice non gestita dai semafori
98  bulettinboard *list_bb(); //OPERAZIONE DI LETTURA
99
100 //legge un messaggio prendendo in input l'id del messaggio
101 // ritorna una copia (clonata) della struttura sulla memoria condivisa,
102 // la struttura ha id = -1 se il messaggio non e' stato trovato
103 message_bb readMessage_bb(int idMessage); //OPERAZIONE DI LETTURA
104
105 //scrive nell'array dei messaggi della memoria condivisa il messaggio nella prima
    posizione disponibile
106 // ritorna l'id univoco assegnato al messaggio, oppure -1 se non ci sono posizioni
    disponibili
107 int writeMessage_bb(message_bb message); //OPERAZIONE DI SCRITTURA
108
109 //rimuove dall'array dei messaggi della memoria condivisa il messaggio
    specificando l'id del messaggio,
110 // inoltre scala tutti i messaggi seguenti in modo da non lasciare buchi
    nell'array stesso,
111 // ritorna 0 se buon esito, -1 in caso di errore o se non e' stato trovato il
    messaggio con id richiesto
112 int removeMessage_bb(int idMessage); //OPERAZIONE DI SCRITTURA
113
114 //Popola la variabile passata per riferimento con la stringa contenente la user
    name dell'utente che ha lanciato il processo, riempiendo non piu' di size
    caratteri
115 void setUserLogin(char *userLogin, int size);
116
117 //Popola la variabile passata per riferimento di size 20 con la stringa contenente
    la stringa formattata della struttura tm passata come parametro, formattata con
    "GG/MM/AAAA hh:mm:ss"
118 void formatTime(char *formatDateTime, struct tm *dateTime);
119
120 //Popola la variabile tempo passata per riferimento, con l'orario attuale di
    sistema
121 void setDateTime(struct tm *dateTime);
122
123 //Crea un nuovo messaggio message_bb, popolando i suoi campi e inserendolo nella
    memoria condivisa,
124 // utilizzando la funzione writeMessage_bb: ritorna il messaggio stesso creato,
125 // con id popolato con -1 se non e' stato possibile inserire il messaggio, oppure
    con l'id univoco assegnato al messaggio
```

File: VolatileBulletinBoard.h

```
126 message_bb insertNewMessage(char *title, char *text, char *email);
127
128 //printf di tutti i campi del messaggio passato come parametro in un'unica riga
129 void printMessage(message_bb message);
130
131 //modifica nell'array dei messaggi della memoria condivisa, il messaggio passato
    come paramentro con id popolato,
132 // ritorna -1 se non ha trovato il messaggio nella memoria condivisa, oppure l'id
    del messaggio da modificare
133 //NON RICHISTO DA SPECIFICHE MA IMPLEMENTATO COMUNQUE
134 //FUNZIONE NON USATA MA IMPLEMENTATA COMUNQUE
135 int updateMessage_bb(message_bb message); //OPERAZIONE DI SCRITTURA
136
137 //Modifica il messaggio con id passato come parametro, controllando se l'utente e'
    proprietario del messaggio stesso,
138 // aggiorna anche insertDate,
139 // ritorna -1 se non ha trovato il messaggio nella memoria condivisa, -2 se
    l'utente non e' owner del messaggio, oppure l'id del messaggio da modificare
140 //NON RICHISTO DA SPECIFICHE MA IMPLEMENTATO COMUNQUE
141 //FUNZIONE NON USATA MA IMPLEMENTATA COMUNQUE
142 int updateMessage(int idMessage, char *title, char *text, char *email);
143
144 //Funzioni per Backup/Restore memoria condivisa su file
145 //NON RICHISTO DA SPECIFICHE MA IMPLEMENTATO COMUNQUE
146 //Ritornano -1 in caso di errore o 0 in caso di successo
147 int fileBackupMessage_bb(char *filename); //OPERAZIONE DI LETTURA
148 int fileRestoreMessage_bb(char *filename); //OPERAZIONE DI SCRITTURA
149 char *FILE_MBULLETTINBOARD;
150
151 //Funzioni gestione bacheca con menu testuali e funzioni standard per input-output
152 void runSimpleBulettinboardManager();
153 void printHelp();
154 void listBulettinboardManager();
155 void readBulettinboardManager();
156 void insertBulettinboardManager();
157 void removeBulettinboardManager();
158
159 //Legge in input da stdin, maxLengthMessage caratteri, fino ad incontrare
    (invio).(invio) (in stile DATA SMTP)
160 void inputTextData(char *textMessage, int maxLengthMessage);
161 //Rimuove da \n in poi, in una stringa
162 void removeCR(char *text);
163
164 //Funzioni di utilita'
165 //Stampa l'help di utilizzo
166 void usage();
167 //Stampa i valori di default utilizzati
168 void dumpDefaultValues();
169 void dumpDefaultKeysValues();
170 //Stampa intestazione iniziale
171 void printHeader();
172
173 //Setta la gestione di default dei segnali SIGINT, SIGTERM, SIGHUP, SIGQUIT
174 void defaultTerminationSignal();
175 //Ignora la gestione di default dei segnali SIGINT, SIGTERM, SIGHUP, SIGQUIT
176 void ignoreTerminationSignal();
177
178 //Funzione di test usata per segnali di interruzione
179 //void waitSec(int seconds);
180
181
182 #endif /* VOLATILEBULLETINBOARD_H_ */
183
```

File: VolatileBulletinBoardFunctions.c

```c
/*
 ============================================================================
 Name        : VolatileBulletinBoard.c
 Author      : Pier Paolo Ciarravano
 Version     : 1.0 - 15/09/2008
 Copyright   : GPL - General Public License
 Description : Bacheca Elettronica Volatile in C, Ansi-style
               Corso di Sistemi Operativi I - Prof. A.Santoro 2007-2008
               Ingegneria Informatica - Univ. "La Sapienza" Roma
 ============================================================================
 */

#include "VolatileBulletinBoard.h"

void setDefaultValue() {

  //Valori di default
  SEMAPHORE_KEY = 2730; //0AAA
  SMEMORY_KEY = 3003; //0BBB
  MAX_READERS = 100;
  MAX_MESSAGES = 1000;

}

void runSimpleBulettinboardManager() {

  printHelp();

  char inputCommand[1024];
  char command;

  do
  {
    //Leggo Comando
    printf("Comando (h per aiuto):");
    //gets(inputCommand);
    fgets(inputCommand, 1024, stdin); //Piu' sicuro di gets per buffer overflow
    command = inputCommand[0];
    //printf("char command input: %c\n" , command);

    switch (command) {

        case 'l':
        case 'L':
          //printf("Lista messaggi presenti in bacheca\n");
          listBulettinboardManager();
          break;

        case 'r':
        case 'R':
          //printf("Lettura messaggio in bacheca\n");
          readBulettinboardManager();
          break;

        case 'a':
        case 'A':
          //printf("Spedizione nuovo messaggio in bacheca\n");
          insertBulettinboardManager();
          break;

        case 'd':
        case 'D':
          //printf("Cancellazione messaggio in bacheca\n");
          removeBulettinboardManager();
          break;

        case 'h':
        case 'H':
          printHelp();
          break;

        case 'q':
        case 'Q':
          printf("Uscita dalla gestione bacheca... Bye!!\n\n");
```

File: VolatileBulletinBoardFunctions.c

```c
            detachSharedMemory();
            break;

          default:
            printf("Comando non riconosciuto! Utilizzare h per lista comandi\n\n");
      }

    }
    while((command!='q')&&(command!='Q'));

}

void listBulettinboardManager() {

    bulettinboard *listaMessaggi = list_bb();

    if (((*listaMessaggi).max_messages)==0)
    {
      printf("Nessun messaggio presente in bacheca!\n\n");
      return;
    }

    printf("Lista messaggi presenti in bacheca:\n\n");
    char inputCommand[1024];
    int i;
    for(i=0; i<((*listaMessaggi).max_messages); i++)
    {
      //Stampa riga messaggio sintetica senza testo e email
      char dateFormat[20];
      formatTime(dateFormat, &((*listaMessaggi).messages[i].insertDate));
      printf("Messaggio ID: %5d - Oggetto: \"%-40.40s\" - Username: %-10.10s - Data/ora: %s\n", (*listaMessaggi).messages[i].id, (*listaMessaggi).messages[i].title, (*listaMessaggi).messages[i].userLogin, dateFormat);

      //Gestione paginazione di 10 messaggi
      if ((((i+1) % 10) == 0) && (i<(((*listaMessaggi).max_messages))-1) )
      {
        printf("\nPremere invio per prossima pagina, oppure Q e invio per terminare...");
        fgets(inputCommand, 1024, stdin);
        if ((inputCommand[0]=='q')||(inputCommand[0]=='Q'))
        {
          break;
        }
        printf("\n");
      }
    }
    printf("\n");
    free(listaMessaggi);

}

void readBulettinboardManager() {

    int idRequest = -1;
    int sscanfResult;
    char inputCommand[1024];

    printf("Inserisci ID identificativo messaggio da leggere:");
    fgets(inputCommand, 1024, stdin);
    //uso sscanf in modo da evitare la memorizzazione di altre stringhe nel buffer di lettura
    sscanfResult = sscanf(inputCommand, "%d", &idRequest);
    if ((sscanfResult == 0)||(idRequest==-1))
    {
      printf("Valore inserito non corretto!!\n\n");
    }
    else
    {
      //printf("Leggo messaggio: %d\n\n", idRequest);
      message_bb readedMessage = readMessage_bb(idRequest);

      if (readedMessage.id == -1)
```

File: VolatileBulletinBoardFunctions.c

```
144     {
145       printf("Il messaggio con ID: %d non e' presente in bacheca!!\n\n",
    idRequest);
146     }
147     else
148     {
149       printf("\n-------------------------------------------------------\n");
150       printf("MESSAGGIO ID: %d\n", readedMessage.id);
151       char dateFormat[20];
152       formatTime(dateFormat, &(readedMessage.insertDate));
153       printf("Oggetto: \"%s\"\n", readedMessage.title);
154       printf("Testo: \"%s\"\n", readedMessage.text);
155       printf("Username: %s\n", readedMessage.userLogin);
156       printf("Email: %s\n", readedMessage.email);
157       printf("Data/Ora inserimento: %s\n", dateFormat);
158       printf("-------------------------------------------------------\n\n");
159     }
160   }
161
162 }
163
164 void insertBulettinboardManager() {
165
166   char inputText[1024];
167   char titleMessage[100];
168   char textMessage[2048];
169   char emailMessage[100];
170
171   //strcpy (titleMessage,"");
172   //strcpy (textMessage,"");
173   //strcpy (emailMessage,"");
174
175   printf("Spedizione nuovo messaggio in bacheca:\n");
176
177   //Input titolo
178   printf("Inserisci l'oggetto del messaggio (invio per terminare, max.100
    carat.):");
179   fgets(inputText, 1024, stdin); //leggo piu' di 100 caratteri per evitare che
    questi rimangano nel buffer di lettura
180   strncpy(titleMessage, inputText, 100);
181   titleMessage[99] = '\0';
182   removeCR(titleMessage);
183
184   //Input testo
185   printf("Inserisci il testo del messaggio (per terminare \".\" da solo in una
    riga, max.2048 carat.):\n");
186   inputTextData(textMessage, 2048);
187
188   //Input email
189   printf("Inserisci indirizzo email (invio per terminare, max.100 carat.):");
190   fgets(inputText, 1024, stdin); //leggo piu' di 100 caratteri per evitare che
    questi rimangano nel buffer di lettura
191   strncpy(emailMessage, inputText, 100);
192   emailMessage[99] = '\0';
193   removeCR(emailMessage);
194
195   //Stampa campi inseriti
196   printf("\n------------ DATI MESSSAGGIO DA SPEDIRE: ------------\n");
197   printf("Oggetto: \"%s\"\n", titleMessage);
198   printf("Testo: \"%s\"\n", textMessage);
199   printf("Email: %s\n\n", emailMessage);
200
201   char inputCommand[1024];
202   printf("Sei sicuro di voler inviare questo messaggio in bacheca (s/N):");
203   fgets(inputCommand, 1024, stdin);
204   if ((inputCommand[0]=='s')||(inputCommand[0]=='S'))
205   {
206     //Spedisco il messaggio in bacheca
207     message_bb message = insertNewMessage(titleMessage, textMessage,
    emailMessage);
208
209     if (message.id == -1)
210     {
211       printf("Non e' stato possibile inserire in bacheca il messaggio perche' la
```

```
    bacheca e' piena!!\n\n");
212     }
213     else
214     {
215       printf("Messaggio spedito correttamente in bacheca:");
216       printf("\n-----------------------------------------------------\n");
217       printf("MESSAGGIO ID: %d\n", message.id);
218       char dateFormat[20];
219       formatTime(dateFormat, &(message.insertDate));
220       printf("Oggetto: \"%s\"\n", message.title);
221       printf("Testo: \"%s\"\n", message.text);
222       printf("Username: %s\n", message.userLogin);
223       printf("Email: %s\n", message.email);
224       printf("Data/Ora inserimento: %s\n", dateFormat);
225       printf("-----------------------------------------------------\n\n");
226     }
227   }
228   else
229   {
230     printf("Spedizione messaggio annullata\n\n");
231   }
232
233 }
234
235
236 void removeBulettinboardManager() {
237
238   int idRequest = -1;
239   int sscanfResult;
240   char inputCommand[1024];
241
242   printf("Inserisci ID identificativo messaggio da cancellare:");
243   fgets(inputCommand, 1024, stdin);
244   //uso sscanf in modo da evitare la memorizzazione di altre stringhe nel buffer
    di lettura
245   sscanfResult = sscanf(inputCommand, "%d", &idRequest);
246   if ((sscanfResult == 0)||(idRequest==-1))
247   {
248     printf("Valore inserito non corretto!!\n\n");
249   }
250   else
251   {
252     //printf("Cancello messaggio: %d\n\n", idRequest);
253     int removeResult = removeMessage_bb(idRequest);
254
255     if (removeResult == -1)
256     {
257       printf("Il messaggio con ID: %d non e' presente in bacheca, pertanto non e'
    possibile cancellarlo!!\n\n", idRequest);
258     }
259     else if (removeResult == 0)
260     {
261       printf("Il messaggio con ID: %d e' stato cancellato correttamente dalla
    bacheca!!\n\n", idRequest);
262     }
263   }
264
265 }
266
267
268 void inputTextData(char *textMessage, int maxLengthMessage) {
269
270   int lastIndex = maxLengthMessage-1;
271   maxLengthMessage -= 1; //conteggio il terminatore di stringa
272   strcpy (textMessage, "");
273   char inputLine[4096];
274   char *substr;
275
276   //Esce con (invio).(invio) (in stile DATA SMTP)
277   //     Enter text, end with "." on a line by itself
278   do
279   {
280     fgets(inputLine, 4000, stdin);
281     strncat (textMessage, inputLine, maxLengthMessage);
```

File: VolatileBulletinBoardFunctions.c

```
282     maxLengthMessage -= strlen(inputLine);
283   }
284   while ( (strcmp (inputLine , ".\n") != 0) && (maxLengthMessage>0) );
285   //while ( ((substr = strstr(textMessage, "\n.\n"))==(char *)0) &&
    (maxLengthMessage>0) );
286
287   //if (substr!=(char *)0)
288   if ((substr = strstr(textMessage, "\n.\n"))!=(char *)0)
289   {
290     //rimuove ultima sequenza riga \n.\n, spostando semplicemente il terminatore
    di stringa nel puntatore alla sottostringa
291     //strcpy ((substr+1), ""); //Elimina solo .\n
292     strcpy (substr, "");
293   }
294   //Caso stringa vuota con il solo ".\n", nel caso l'utente non abbia inserito
    nulla
295   if((strcmp (textMessage , ".\n") == 0))
296   {
297     strcpy (textMessage, "");
298   }
299
300   //Aggiungo comunque un terminatore di stringa
301   textMessage[lastIndex] = '\0';
302
303 }
304
305 void removeCR(char *text) {
306
307   char *substr;
308   if ((substr = strstr(text, "\n"))!=(char *)0)
309   {
310     strcpy (substr, "");
311   }
312
313 }
314
315 void printHelp() {
316
317   printf("   Comandi:\n");
318   printf("            l: Lista messaggi presenti in bacheca\n");
319   printf("            r: Lettura messaggio in bacheca\n");
320   printf("            a: Spedizione nuovo messaggio in bacheca\n");
321   printf("            d: Cancellazione messaggio in bacheca\n");
322   printf("            h: Visualizza questo aiuto comandi\n");
323   printf("            q: Uscita dalla gestione bacheca\n");
324   printf("\n");
325
326 }
327
328 int init_bb() {
329   return init_bb_arg(SEMAPHORE_KEY, SMEMORY_KEY, MAX_READERS, MAX_MESSAGES);
330 }
331
332 int init_bb_arg(long int sk, int long mk, int mr, int mm) {
333
334   //Ritorna:  0:ok
335   //      -1:errore creazione memoria condivisa
336   //      -2:errore creazione semafori
337
338
339   //AZIONI PER MEMORIA CONDIVISA
340
341   //Creazione memoria condivisa
342   ID_SMEMORY = shmget(mk, 2*sizeof(int) + mm*sizeof(message_bb), IPC_CREAT |
    IPC_EXCL | 0666);
343   if (ID_SMEMORY == -1)
344     return -1;
345
346   //Collegamento memoria condivisa allo spazio di indirizzamento del processo in
    esecuzione
347   MBULLETTINBOARD = shmat(ID_SMEMORY, NULL , SHM_R | SHM_W);
348   if ( MBULLETTINBOARD == (bulettinboard*)-1 )
349     return -1;
350
```

File: VolatileBulletinBoardFunctions.c

```
351   //Inizializzazione struttura puntata dalla memoria condivisa
352   (*MBULLETTINBOARD).max_messages = mm;
353   (*MBULLETTINBOARD).idSequence = 0; //Variabile per assegnare un nuovo id al
    messaggio
354   int i;
355   for(i=0; i<mm; i++)
356   {
357     (*MBULLETTINBOARD).messages[i].id = 0;
358     (*MBULLETTINBOARD).messages[i].isFree = 1;
359     //strcpy((*MBULLETTINBOARD).messages[i].title, "");
360   }
361
362
363   //AZIONI PER SEMAFORI
364
365   //Creazione array semafori: 0 semaforo READERS, 1 semaforo WRITERS
366   ID_SEMAPHORE_RW = semget(sk, 2, IPC_CREAT | IPC_EXCL | 0666);
367   if (ID_SEMAPHORE_RW == -1)
368       return -2;
369
370   //Inizializzazione semafori
371   // Semaforo con indice 0 per READERS: inizializzato con MAX_READERS
372   // Semaforo con indice 1 per WRITERS: semaforo di tipo mutex, inizializzato con
    1
373   ushort semaphoresInitValues[2] = {MAX_READERS, 1};
374   int semctl_result = semctl(ID_SEMAPHORE_RW, 0, SETALL, semaphoresInitValues );
375   if (semctl_result == -1)
376     return -2;
377
378   //Messaggi di output gestiti da chi ha chiamato questa funzione, tramite il
    parametro di ritorno
379   return 0;
380 }
381
382
383 int destroy_bb() {
384   return destroy_bb_arg(SEMAPHORE_KEY, SMEMORY_KEY);
385 }
386
387 int destroy_bb_arg(long int sk, int long mk) {
388
389   //Ritorna maschera bit  1:ok
390   //            2:errore rimozione memoria condivisa
391   //            4:errore rimozione semafori
392
393   int result = 1;
394
395   //Rimozione memoria condivisa
396   ID_SMEMORY = shmget(mk, 0, 0666); //il parametro 0666 viene ignorato
397   if (ID_SMEMORY == -1) { //Specificato errore ENOENT in error.h
398     //La memoria condivisa specificata dalla chiave non esiste
399     //printf("La memoria condivisa specificata dalla chiave: %ld, non esiste!\n",
    mk);
400     result += 2;
401   }
402   else
403   {
404     //Lancio comando rimozione memoria condivisa
405     int shmctl_result = shmctl(ID_SMEMORY, IPC_RMID, NULL);
406     if (shmctl_result == -1)
407       result += 2;
408     //else
409     //  printf("La memoria condivisa specificata dalla chiave: %ld, e' stata
    rimossa correttamente!\n", mk);
410   }
411
412   //Rimozione array semafori
413   ID_SEMAPHORE_RW = semget(sk, 0, 0666); //il parametro 0666 viene ignorato
414   if (ID_SEMAPHORE_RW == -1) {
415     result += 4;
416   }
417   else
418   {
419     //Lancio comando rimozione array semafori
```

File: VolatileBulletinBoardFunctions.c

```
420     int semctl_result = semctl(ID_SEMAPHORE_RW, 0, IPC_RMID, NULL);
421     if (semctl_result == -1)
422       result += 4;
423     //else
424     //  printf("L'array di semafori specificato dalla chiave: %ld, e' stata
    rimosso correttamente!\n", sk);
425   }
426
427   //Messaggi di output gestiti da chi ha chiamato questa funzione, tramite il
    parametro di ritorno
428   return result;
429 }
430
431 //utilizza ID_SMEMORY, ID_SEMAPHORE_RW per rimuovere memoria condivisa e semafori
432 //FUNZIONE NON USATA MA IMPLEMENTATA COMUNQUE
433 int destroyUsingId_bb() {
434
435   //Ritorna maschera bit  1:ok
436   //            2:errore rimozione memoria condivisa
437   //            4:errore rimozione semafori
438
439   int result = 1;
440
441   //Rimozione memoria condivisa
442   int shmctl_result = shmctl(ID_SMEMORY, IPC_RMID, NULL);
443   if (shmctl_result == -1)
444     result += 2;
445
446   //Rimozione array semafori
447   int semctl_result = semctl(ID_SEMAPHORE_RW, 0, IPC_RMID, NULL);
448   if (semctl_result == -1)
449     result += 4;
450
451   //Messaggi di output gestiti da chi ha chiamato questa funzione, tramite il
    parametro di ritorno
452   return result;
453 }
454
455
456 int open_bb() {
457   return open_bb_arg(SEMAPHORE_KEY, SMEMORY_KEY);
458 }
459
460 int open_bb_arg(long int sk, int long mk) {
461
462   //Ritorna:  0:ok
463   //      -1:errore apertura identificatore memoria condivisa
464   //      -2:errore apertura identificatore semaforo
465   //      -3:errore collegamento memoria condivisa
466
467   //apertura identificatore semaforo
468   ID_SEMAPHORE_RW = semget(sk, 0, 0666); //il parametro 0666 viene ignorato
469   if (ID_SEMAPHORE_RW == -1) {
470     return -2;
471   }
472
473   //apertura identificatore memoria condivisa
474   ID_SMEMORY = shmget(mk, 0, 0666); //il parametro 0666 viene ignorato
475   if (ID_SMEMORY == -1) {
476     return -1;
477   }
478
479   //Colegamento memoria condivisa
480   MBULLETTINBOARD = shmat(ID_SMEMORY, NULL , SHM_R | SHM_W);
481   if ( MBULLETTINBOARD == (bulettinboard*)-1 )
482     return -3;
483
484   //Inizializzazione MAX_MESSAGES
485   MAX_MESSAGES = (*MBULLETTINBOARD).max_messages;
486
487   return 0;
488 }
489
490
```

File: VolatileBulletinBoardFunctions.c

```
491 int detachSharedMemory() {
492   return shmdt(MBULLETTINBOARD);
493 }
494
495
496 //utilizzare free() dopo aver usato la struttura ritornata
497 bulettinboard *list_bb() {
498
499   ignoreTerminationSignal();
500
501   //Decremento (-1) semaforo 0 READERS
502   struct sembuf waitReader[1] = { {0, -1, 0} };
503   int retSemop = semop(ID_SEMAPHORE_RW, waitReader, 1);
504   if (retSemop == -1) {
505     printf("Errore semop semaforo 0: -1\n");
506     //exit(-1);
507   }
508
509   //Conto numero di messaggi validi presenti nell'array
    (*MBULLETTINBOARD).messages[]
510   int num_valid_messages = 0;
511   int i;
512   for(i=0; i<MAX_MESSAGES; i++)
513   {
514     if ((*MBULLETTINBOARD).messages[i].isFree == 0)
515     {
516       num_valid_messages++;
517     }
518   }
519   //printf("num_valid_messages: %d\n", num_valid_messages);
520
521   //Alloco struttura result da ritornare
522   bulettinboard *result;
523   result = malloc(2*sizeof(int) + num_valid_messages*sizeof(message_bb));
524   if (result == NULL) {
525     printf("Errore malloc struttura result per funzione list_bb\n");
526     //exit(-1);
527   }
528
529   //Popolo struttura result da ritornare copiando i campi dei messaggi validi
530   (*result).max_messages = num_valid_messages;
531   (*result).idSequence = -1; //Variabile non utilizzata nella struttura ritornata
532   int j = 0;
533   for(i=0; i<MAX_MESSAGES; i++)
534   {
535     if ((*MBULLETTINBOARD).messages[i].isFree == 0)
536     {
537       (*result).messages[j].id = (*MBULLETTINBOARD).messages[i].id;
538       (*result).messages[j].isFree = 0;
539       strcpy((*result).messages[j].title, (*MBULLETTINBOARD).messages[i].title);
540       strcpy((*result).messages[j].text, (*MBULLETTINBOARD).messages[i].text);
541       strcpy((*result).messages[j].userLogin,
    (*MBULLETTINBOARD).messages[i].userLogin);
542       strcpy((*result).messages[j].email, (*MBULLETTINBOARD).messages[i].email);
543       (*result).messages[j].insertDate =
    (*MBULLETTINBOARD).messages[i].insertDate;
544       j++;
545     }
546   }
547
548   //Incremento (+1) semaforo 0 READERS
549   struct sembuf signalReader[1] = { {0, +1, 0} };
550   retSemop = semop(ID_SEMAPHORE_RW, signalReader, 1);
551   if (retSemop == -1) {
552     printf("Errore semop semaforo 0: +1\n");
553     //exit(-1);
554   }
555
556   defaultTerminationSignal();
557
558   return result;
559 }
560
561
```

File: VolatileBulletinBoardFunctions.c

```
562 message_bb readMessage_bb(int idMessage) {
563
564   ignoreTerminationSignal();
565
566   //Decremento (-1) semaforo 0 READERS
567   struct sembuf waitReader[1] = { {0, -1, 0} };
568   int retSemop = semop(ID_SEMAPHORE_RW, waitReader, 1);
569   if (retSemop == -1) {
570     printf("Errore semop semaforo 0: -1\n");
571     //exit(-1);
572   }
573
574   message_bb result;
575   result.id = -1;
576
577   int i;
578   for(i=0; i<MAX_MESSAGES; i++)
579   {
580     //Cerco il messaggio con id==idMessage
581     if ( ( (*MBULLETTINBOARD).messages[i].isFree == 0 ) &&
    ((*MBULLETTINBOARD).messages[i].id == idMessage) )
582     {
583       result.id = (*MBULLETTINBOARD).messages[i].id;
584       result.isFree = 0;
585       strcpy(result.title, (*MBULLETTINBOARD).messages[i].title);
586       strcpy(result.text, (*MBULLETTINBOARD).messages[i].text);
587       strcpy(result.userLogin, (*MBULLETTINBOARD).messages[i].userLogin);
588       strcpy(result.email, (*MBULLETTINBOARD).messages[i].email);
589       result.insertDate = (*MBULLETTINBOARD).messages[i].insertDate;
590       break; //esco dal ciclo
591     }
592   }
593
594   //Incremento (+1) semaforo 0 READERS
595   struct sembuf signalReader[1] = { {0, +1, 0} };
596   retSemop = semop(ID_SEMAPHORE_RW, signalReader, 1);
597   if (retSemop == -1) {
598     printf("Errore semop semaforo 0: +1\n");
599     //exit(-1);
600   }
601
602   defaultTerminationSignal();
603
604   return result;
605 }
606
607
608 int writeMessage_bb(message_bb message) {
609
610   ignoreTerminationSignal();
611
612   //Decremento (-1) semaforo 1 WRITERS (mutex)
613   struct sembuf waitWriter[1] = { {1, -1, 0} };
614   int retSemop = semop(ID_SEMAPHORE_RW, waitWriter, 1);
615   if (retSemop == -1) {
616     printf("Errore semop semaforo 1: -1\n");
617     //exit(-1);
618   }
619
620   //Blocco tutti i lettori
621   int r;
622   for (r = 0; r < MAX_READERS; ++r) {
623     //Decremento (-1) semaforo 0 READERS
624     struct sembuf waitReader[1] = { {0, -1, 0} };
625     int retSemop = semop(ID_SEMAPHORE_RW, waitReader, 1);
626     if (retSemop == -1) {
627       printf("Errore semop semaforo 0: -1\n");
628       //exit(-1);
629     }
630   }
631
632   //Usata per test segnali di interruzione
633   //waitSec(10);
634
```

```
635   //Opero l'inserimento del messaggio nella memoria condivisa
636   int idResult = -1;
637   //Cerco la prima posizione libera (free==1) nell'array messages della struttura
    puntata da *MBULLETTINBOARD
638   int i;
639   for(i=0; i<MAX_MESSAGES; i++)
640   {
641     if (  (*MBULLETTINBOARD).messages[i].isFree == 1 )
642     {
643       //Posizione libera con indice i trovata nell'array:
    (*MBULLETTINBOARD).messages[i]
644       //Copio message in (*MBULLETTINBOARD).messages[i]
645       (*MBULLETTINBOARD).messages[i].id = ++((*MBULLETTINBOARD).idSequence);
    //Assegno un id univoco al messaggio
646       idResult = (*MBULLETTINBOARD).messages[i].id; //popolo il valore da
    ritornare
647       (*MBULLETTINBOARD).messages[i].isFree = 0;
648       strcpy((*MBULLETTINBOARD).messages[i].title, message.title);
649       strcpy((*MBULLETTINBOARD).messages[i].text, message.text);
650       strcpy((*MBULLETTINBOARD).messages[i].userLogin, message.userLogin);
651       strcpy((*MBULLETTINBOARD).messages[i].email, message.email);
652       (*MBULLETTINBOARD).messages[i].insertDate = message.insertDate;
653       break; //esco dal ciclo
654     }
655   }
656
657
658   //Sblocco tutti i lettori
659   for (r = 0; r < MAX_READERS; ++r) {
660     //Incremento (+1) semaforo 0 READERS
661     struct sembuf signalReader[1] = { {0, +1, 0} };
662     retSemop = semop(ID_SEMAPHORE_RW, signalReader, 1);
663     if (retSemop == -1) {
664       printf("Errore semop semaforo 0: +1\n");
665       //exit(-1);
666     }
667   }
668
669   //Incremento (+1) semaforo 1 WRITERS (mutex)
670   struct sembuf signalWriter[1] = { {1, +1, 0} };
671   retSemop = semop(ID_SEMAPHORE_RW, signalWriter, 1);
672   if (retSemop == -1) {
673     printf("Errore semop semaforo 1: +1\n");
674     //exit(-1);
675   }
676
677   defaultTerminationSignal();
678
679   return idResult;
680 }
681
682
683 int removeMessage_bb(int idMessage) {
684
685   ignoreTerminationSignal();
686
687   //Decremento (-1) semaforo 1 WRITERS (mutex)
688   struct sembuf waitWriter[1] = { {1, -1, 0} };
689   int retSemop = semop(ID_SEMAPHORE_RW, waitWriter, 1);
690   if (retSemop == -1) {
691     printf("Errore semop semaforo 1: -1\n");
692     //exit(-1);
693   }
694
695   //Blocco tutti i lettori
696   int r;
697   for (r = 0; r < MAX_READERS; ++r) {
698     //Decremento (-1) semaforo 0 READERS
699     struct sembuf waitReader[1] = { {0, -1, 0} };
700     int retSemop = semop(ID_SEMAPHORE_RW, waitReader, 1);
701     if (retSemop == -1) {
702       printf("Errore semop semaforo 0: -1\n");
703       //exit(-1);
704     }
```

```
705   }
706
707
708   //Opero la rimozione del messaggio nella memoria condivisa
709   int idResult = -1;
710   int i;
711   for(i=0; i<MAX_MESSAGES; i++)
712   {
713     //Cerco il messaggio con id==idMessage
714     if ( ( (*MBULLETTINBOARD).messages[i].isFree == 0 ) &&
    ((*MBULLETTINBOARD).messages[i].id == idMessage) )
715     {
716       (*MBULLETTINBOARD).messages[i].isFree = 1; //Operazione necessaria solo se
    il messaggio e' l'ultimo dell'array o se non si vuole effettuare lo spostamento
    dei messaggi seguenti nell'array
717       //Scalo di una posizione tutti gli elementi dell'array, se i non e' l'ultimo
    elemento
718       if (i<(MAX_MESSAGES-1))
719       {
720         int j;
721         for(j=(i+1); j<MAX_MESSAGES; j++)
722         {
723           //Copia elemento (*MBULLETTINBOARD).messages[j] in elemento
    (*MBULLETTINBOARD).messages[j-1]
724           (*MBULLETTINBOARD).messages[j-1].id = (*MBULLETTINBOARD).messages[j].id;
725           (*MBULLETTINBOARD).messages[j-1].isFree =
    (*MBULLETTINBOARD).messages[j].isFree;
726           strcpy((*MBULLETTINBOARD).messages[j-1].title,
    (*MBULLETTINBOARD).messages[j].title);
727           strcpy((*MBULLETTINBOARD).messages[j-1].text,
    (*MBULLETTINBOARD).messages[j].text);
728           strcpy((*MBULLETTINBOARD).messages[j-1].userLogin,
    (*MBULLETTINBOARD).messages[j].userLogin);
729           strcpy((*MBULLETTINBOARD).messages[j-1].email,
    (*MBULLETTINBOARD).messages[j].email);
730           (*MBULLETTINBOARD).messages[j-1].insertDate =
    (*MBULLETTINBOARD).messages[j].insertDate;
731           //printf("copiato elemento: %d-->%d\n", j, j-1 );
732         }
733       }
734       idResult = 0;
735       break; //esco dal ciclo
736     }
737   }
738
739
740   //Sblocco tutti i lettori
741   for (r = 0; r < MAX_READERS; ++r) {
742     //Incremento (+1) semaforo 0 READERS
743     struct sembuf signalReader[1] = { {0, +1, 0} };
744     retSemop = semop(ID_SEMAPHORE_RW, signalReader, 1);
745     if (retSemop == -1) {
746       printf("Errore semop semaforo 0: +1\n");
747       //exit(-1);
748     }
749   }
750
751   //Incremento (+1) semaforo 1 WRITERS (mutex)
752   struct sembuf signalWriter[1] = { {1, +1, 0} };
753   retSemop = semop(ID_SEMAPHORE_RW, signalWriter, 1);
754   if (retSemop == -1) {
755     printf("Errore semop semaforo 1: +1\n");
756     //exit(-1);
757   }
758
759   defaultTerminationSignal();
760
761   return idResult;
762 }
763
764
765 void setUserLogin(char *userLogin, int size) {
766
767   //non ritorna correttamente l'utente che ha lanciato il processo, ma ritorna i
```

```
    nome utente che aperto la finestra della shell
768   //getlogin_r(userLogin, size);
769
770   //prelevo correttamente il nome utente che ha lanciato il processo
771   uid_t uid = geteuid ();
772   struct passwd *pw = getpwuid (uid);
773   if (pw)
774   {
775     char *user = pw->pw_name;
776     strncpy(userLogin, user, size);
777   }
778   else
779   {
780     strcpy(userLogin, ".");
781   }
782
783   //printf("Username utente che ha lanciato il processo: %s\n", userLogin);
784 }
785
786
787 void formatTime(char *formatDateTime, struct tm *dateTime) {
788
789   strftime (formatDateTime,20,"%d/%m/%Y %H:%M:%S", dateTime);
790
791 }
792
793
794 void setDateTime(struct tm *dateTime) {
795
796   time_t rawtime;
797   time ( &rawtime );
798   *dateTime = *(localtime ( &rawtime ));
799
800 }
801
802
803 message_bb insertNewMessage(char *title, char *text, char *email) {
804
805   //Creo il messaggio e lo popolo
806   message_bb result;
807   result.id = -1;
808   result.isFree = 0;
809   strncpy(result.title, title, 100);
810   strncpy(result.text, text, 2048);
811   setUserLogin(result.userLogin, 20);
812   strncpy(result.email, email, 100);
813   setDateTime(&(result.insertDate));
814
815   //Inserisco il messaggio nella memoria condivisa
816   int idMessage = writeMessage_bb(result);
817   result.id = idMessage;
818
819   return result;
820 }
821
822
823 int updateMessage_bb(message_bb message) {
824
825   ignoreTerminationSignal();
826
827   //Decremento (-1) semaforo 1 WRITERS (mutex)
828   struct sembuf waitWriter[1] = { {1, -1, 0} };
829   int retSemop = semop(ID_SEMAPHORE_RW, waitWriter, 1);
830   if (retSemop == -1) {
831     printf("Errore semop semaforo 1: -1\n");
832     //exit(-1);
833   }
834
835   //Blocco tutti i lettori
836   int r;
837   for (r = 0; r < MAX_READERS; ++r) {
838     //Decremento (-1) semaforo 0 READERS
839     struct sembuf waitReader[1] = { {0, -1, 0} };
840     int retSemop = semop(ID_SEMAPHORE_RW, waitReader, 1);
```

File: VolatileBulletinBoardFunctions.c

```
841     if (retSemop == -1) {
842       printf("Errore semop semaforo 0: -1\n");
843       //exit(-1);
844     }
845   }
846
847
848   //Opero la rimozione del messaggio nella memoria condivisa
849   int idResult = -1;
850   int i;
851   for(i=0; i<MAX_MESSAGES; i++)
852   {
853     //Cerco il messaggio con id==message.id
854     if ( ( (*MBULLETTINBOARD).messages[i].isFree == 0 ) &&
    ((*MBULLETTINBOARD).messages[i].id == message.id) )
855     {
856       //modifico il messaggio
857       strcpy((*MBULLETTINBOARD).messages[i].title, message.title);
858       strcpy((*MBULLETTINBOARD).messages[i].text, message.text);
859       strcpy((*MBULLETTINBOARD).messages[i].email, message.email);
860       (*MBULLETTINBOARD).messages[i].insertDate = message.insertDate;
861
862       idResult = message.id;
863       break; //esco dal ciclo
864     }
865   }
866
867
868   //Sblocco tutti i lettori
869   for (r = 0; r < MAX_READERS; ++r) {
870     //Incremento (+1) semaforo 0 READERS
871     struct sembuf signalReader[1] = { {0, +1, 0} };
872     retSemop = semop(ID_SEMAPHORE_RW, signalReader, 1);
873     if (retSemop == -1) {
874       printf("Errore semop semaforo 0: +1\n");
875       //exit(-1);
876     }
877   }
878
879   //Incremento (+1) semaforo 1 WRITERS (mutex)
880   struct sembuf signalWriter[1] = { {1, +1, 0} };
881   retSemop = semop(ID_SEMAPHORE_RW, signalWriter, 1);
882   if (retSemop == -1) {
883     printf("Errore semop semaforo 1: +1\n");
884     //exit(-1);
885   }
886
887   defaultTerminationSignal();
888
889   return idResult;
890 }
891
892 int updateMessage(int idMessage, char *title, char *text, char *email) {
893
894   //Leggo il messaggio
895   message_bb updateMessage = readMessage_bb(idMessage);
896   if (updateMessage.id == -1)
897   {
898     return -1;
899   }
900
901   //Controllo non richiesto:
902   // Possibile modifica solo se l'utente che modifica il messaggio e' lo stesso di
    quello che lo ha creato
903   char actualUserLogin[20];
904   setUserLogin(actualUserLogin, 20);
905   if (strcmp (updateMessage.userLogin , actualUserLogin) != 0) {
906     printf("Errore modifica: owners messaggio differenti: %s != %s\n",
    updateMessage.userLogin , actualUserLogin);
907     return -2;
908   }
909
910   //setta la data di inserimento differente
911   setDateTime(&(updateMessage.insertDate));
```

```
912
913   //Setto i campi
914   strcpy(updateMessage.title, title);
915   strcpy(updateMessage.text, text);
916   strcpy(updateMessage.email, email);
917
918   //Salvo il messaggio
919   int idResultUpdate = updateMessage_bb(updateMessage);
920
921   return idResultUpdate;
922 }
923
924
925 int fileBackupMessage_bb(char *filename) {
926
927   ignoreTerminationSignal();
928
929   //Decremento (-1) semaforo 0 READERS
930   struct sembuf waitReader[1] = { {0, -1, 0} };
931   int retSemop = semop(ID_SEMAPHORE_RW, waitReader, 1);
932   if (retSemop == -1) {
933     printf("Errore semop semaforo 0: -1\n");
934     //exit(-1);
935   }
936
937
938   int result;
939   result = -1;
940   //Salvataggio su file leggendo la memoria condivisa
941   int dsFileBackup;
942   dsFileBackup = open(filename, O_WRONLY|O_CREAT|O_TRUNC, 0666);
943   if (dsFileBackup == -1) {
944     printf("ERRORE APERTURA FILE IN SCRITTURA: %s!!\n", filename);
945   }
946   else
947   {
948     int nread = write(dsFileBackup, MBULLETTINBOARD, (2*sizeof(int) +
    MAX_MESSAGES*sizeof(message_bb)) );
949     if (nread==(2*sizeof(int) + MAX_MESSAGES*sizeof(message_bb))) {
950       result = 0;
951     }
952   }
953   close(dsFileBackup);
954
955
956   //Incremento (+1) semaforo 0 READERS
957   struct sembuf signalReader[1] = { {0, +1, 0} };
958   retSemop = semop(ID_SEMAPHORE_RW, signalReader, 1);
959   if (retSemop == -1) {
960     printf("Errore semop semaforo 0: +1\n");
961     //exit(-1);
962   }
963
964   defaultTerminationSignal();
965
966   return result;
967 }
968
969 int fileRestoreMessage_bb(char *filename) {
970
971   ignoreTerminationSignal();
972
973   //Decremento (-1) semaforo 1 WRITERS (mutex)
974   struct sembuf waitWriter[1] = { {1, -1, 0} };
975   int retSemop = semop(ID_SEMAPHORE_RW, waitWriter, 1);
976   if (retSemop == -1) {
977     printf("Errore semop semaforo 1: -1\n");
978     //exit(-1);
979   }
980
981   //Blocco tutti i lettori
982   int r;
983   for (r = 0; r < MAX_READERS; ++r) {
984     //Decremento (-1) semaforo 0 READERS
```

```
  985     struct sembuf waitReader[1] = { {0, -1, 0} };
  986     int retSemop = semop(ID_SEMAPHORE_RW, waitReader, 1);
  987     if (retSemop == -1) {
  988       printf("Errore semop semaforo 0: -1\n");
  989       //exit(-1);
  990     }
  991   }
  992
  993
  994   int result;
  995   result = -1;
  996   //Restore del file scrivendo sulla memoria condivisa
  997   int dsFileRestore = open(filename, O_RDONLY);
  998   if (dsFileRestore == -1) {
  999     printf("ERRORE APERTURA FILE IN LETTURA: %s!!\n", filename);
 1000   }
 1001   else
 1002   {
 1003     int nwrite = read(dsFileRestore, MBULLETTINBOARD, (2*sizeof(int) +
      MAX_MESSAGES*sizeof(message_bb)) );
 1004     if (nwrite==(2*sizeof(int) + MAX_MESSAGES*sizeof(message_bb))) {
 1005       result = 0;
 1006     }
 1007   }
 1008   close(dsFileRestore);
 1009
 1010
 1011   //Sblocco tutti i lettori
 1012   for (r = 0; r < MAX_READERS; ++r) {
 1013     //Incremento (+1) semaforo 0 READERS
 1014     struct sembuf signalReader[1] = { {0, +1, 0} };
 1015     retSemop = semop(ID_SEMAPHORE_RW, signalReader, 1);
 1016     if (retSemop == -1) {
 1017       printf("Errore semop semaforo 0: +1\n");
 1018       //exit(-1);
 1019     }
 1020   }
 1021
 1022   //Incremento (+1) semaforo 1 WRITERS (mutex)
 1023   struct sembuf signalWriter[1] = { {1, +1, 0} };
 1024   retSemop = semop(ID_SEMAPHORE_RW, signalWriter, 1);
 1025   if (retSemop == -1) {
 1026     printf("Errore semop semaforo 1: +1\n");
 1027     //exit(-1);
 1028   }
 1029
 1030   defaultTerminationSignal();
 1031
 1032   return result;
 1033 }
 1034
 1035 void printMessage(message_bb message) {
 1036
 1037   //Formato la insertDate
 1038   char dateFormat[20];
 1039   formatTime(dateFormat, &(message.insertDate));
 1040
 1041   printf("Message id: %d --> %d, \"%s\", \"%s\", %s, %s, %s\n", message.id,
      message.isFree, message.title, message.text, message.userLogin, message.email,
      dateFormat);
 1042
 1043 }
 1044
 1045 int parseMainOptions_bb(int argc, char *argv[]) {
 1046
 1047   //Ritorna maschera bit  1: lancia programma di gestione della bacheca
 1048   //            2: Creazione Bacheca
 1049   //            4: Rimozione Bacheca
 1050   //            8: Backup su file Bacheca
 1051   //            16: Restore da file Bacheca
 1052   //            32: Test (NON IMPLEMENTATO)
 1053
 1054   int result = 1;
 1055   int optionsResult = 0;
```

File: VolatileBulletinBoardFunctions.c

```
1056    while ((optionsResult = getopt(argc, argv, "hads:m:r:n:b:l:")) != -1) {
1057      switch (optionsResult) {
1058
1059      case 'h':
1060        usage();
1061        exit(-1);
1062        break;
1063      case 'a':
1064        //printf("Creazione Bacheca: memoria condivisa e semafori...\n");
1065        //Setto azione: creazione
1066        result += 2;
1067        break;
1068      case 'd':
1069        //printf("Rimozione Bacheca: memoria condivisa e semafori...\n");
1070        //Setto azione: rimozione
1071        result += 4;
1072        break;
1073      case 's':
1074        SEMAPHORE_KEY = strtol(optarg, NULL, 10);
1075        if (SEMAPHORE_KEY == 0) {
1076          printf("Errore: Specificare per -s un valore numerico o diverso da 0!\n");
1077          exit(-1);
1078        }
1079        //printf("s: %ld\n", SEMAPHORE_KEY);
1080        break;
1081      case 'm':
1082        SMEMORY_KEY = strtol(optarg, NULL, 10);
1083        if (SMEMORY_KEY == 0) {
1084          printf("Errore: Specificare per -m un valore numerico o diverso da 0!\n");
1085          exit(-1);
1086        }
1087        //printf("m: %ld\n", SMEMORY_KEY);
1088        break;
1089      case 'r':
1090        MAX_READERS = atoi(optarg);
1091        if (MAX_READERS == 0) {
1092          printf("Errore: Specificare per -r un valore numerico o diverso da 0!\n");
1093          exit(-1);
1094        }
1095        //printf("r: %d\n", MAX_READERS);
1096        break;
1097      case 'n':
1098        MAX_MESSAGES = atoi(optarg);
1099        if (MAX_MESSAGES == 0) {
1100          printf("Errore: Specificare per -n un valore numerico o diverso da 0!\n");
1101          exit(-1);
1102        }
1103        //printf("n: %d\n", MAX_MESSAGES);
1104        break;
1105      case 'b':
1106        FILE_MBULLETTINBOARD = optarg;
1107        //printf("b: %s\n", FILE_MBULLETTINBOARD);
1108        result += 8;
1109        break;
1110      case 'l':
1111        FILE_MBULLETTINBOARD = optarg;
1112        //printf("l: %s\n", FILE_MBULLETTINBOARD);
1113        result += 16;
1114        break;
1115      //case 't':
1116      //  //Setto azione: TEST
1117      //  result += 32;
1118      //  break;
1119      case '?':
1120        printf("Digitare `VolatileBulletinBoard -h' per maggiori informazioni\n");
1121        exit(-1);
1122        break;
1123      //default:
1124        //printf("DEFAULT!!!\n");
1125      }
1126
1127    }
1128    return result;
1129  }
```

```
1130
1131  /*
1132  //Implementato in maniera differente nelle versioni VolatileBulletinBoard.c e
      VolatileBulletinBoardUsers.c
1133  void usage() {
1134
1135    printf("Usare: VolatileBulletinBoard [-a|-d|-b|-l|-t] [OPZIONI]\nOPZIONI:\n");
1136    printf("  -a       Creazione Bacheca: memoria condivisa e semafori\n");
1137    printf("  -d       Rimozione Bacheca: memoria condivisa e semafori\n");
1138    printf("  -s N     setta chiave identificazione semafori\n");
1139    printf("  -m N     setta chiave identificazione memoria condivisa\n");
1140    printf("  -r N     massimo numero di utenti readers\n");
1141    printf("  -n N     massimo numero di messaggi della bacheca\n");
1142    printf("  -b file  backup memoria condivisa su file\n");
1143    printf("  -l file  carica memoria condivisa da file backup\n");
1144    //printf("  -t       test (NON IMPLEMENTATO IN QUESTA VERSIONE)\n");
1145    printf("  -h       Visualizza questo help e termina il programma\n");
1146    printf("\n");
1147    printf("Non usando i parametri -a, -d, -b, -l viene eseguito il programma di
      gestione della bacheca.\n");
1148    printf("Non usando i parametri -s, -m, -r, -n vengono usati i valori di
      default.\n");
1149    printf("I parametri -r e -n vengono usati solamente insieme al parametro -a
      nella creazione bacheca, altrimenti vengono ignorati.\n");
1150    printf("\n");
1151
1152  }
1153  */
1154
1155  void dumpDefaultValues() {
1156
1157    printf("Chiave identificazione semafori: %ld (0X%08X)\n", SEMAPHORE_KEY,
      (unsigned int)SEMAPHORE_KEY);
1158    printf("Chiave identificazione memoria condivisa: %ld (0X%08X)\n", SMEMORY_KEY,
      (unsigned int)SMEMORY_KEY);
1159    printf("Massimo numero di utenti readers: %d\n", MAX_READERS);
1160    printf("Massimo numero di messaggi della bacheca: %d\n", MAX_MESSAGES);
1161  }
1162
1163  void dumpDefaultKeysValues() {
1164
1165    printf("Chiave identificazione semafori: %ld (0X%08X)\n", SEMAPHORE_KEY,
      (unsigned int)SEMAPHORE_KEY);
1166    printf("Chiave identificazione memoria condivisa: %ld (0X%08X)\n", SMEMORY_KEY,
      (unsigned int)SMEMORY_KEY);
1167  }
1168
1169  void printHeader() {
1170
1171    printf("Bacheca Elettronica Volatile di Pier Paolo Ciarravano\n");
1172    printf("Corso di Sistemi Operativi I - Prof. A.Santoro 2007-2008\nIngegneria
      Informatica - Univ. \"La Sapienza\" Roma\n");
1173    printf("Versione: 1.0 - 15/09/2008 \n\n");
1174
1175  }
1176
1177
1178  void defaultTerminationSignal() {
1179    //Setta comportamento di default dei segnali di interruzione causati dall'utente
1180    signal(SIGINT, SIG_DFL);
1181    signal(SIGTERM, SIG_DFL);
1182    signal(SIGHUP, SIG_DFL);
1183    signal(SIGQUIT, SIG_DFL);
1184  }
1185
1186  void ignoreTerminationSignal() {
1187    //Ignora comportamento di default dei segnali di interruzione causati
      dall'utente
1188    signal(SIGINT, SIG_IGN);
1189    signal(SIGTERM, SIG_IGN);
1190    signal(SIGHUP, SIG_IGN);
1191    signal(SIGQUIT, SIG_IGN);
1192  }
1193
```

File: VolatileBulletinBoardFunctions.c

```
1194
1195 //Usata per test segnali di interruzione
1196 //void waitSec(int seconds){
1197 //  clock_t endwait;
1198 //  endwait=clock()+seconds*CLOCKS_PER_SEC;
1199 //  while (clock()<endwait);
1200 //}
1201
```

File: VolatileBulletinBoard.c

```c
/*
 ============================================================================
 Name        : VolatileBulletinBoard.c
 Author      : Pier Paolo Ciarravano
 Version     : 1.0 - 15/09/2008
 Copyright   : GPL - General Public License
 Description : Bacheca Elettronica Volatile in C, Ansi-style
               Corso di Sistemi Operativi I - Prof. A.Santoro 2007-2008
               Ingegneria Informatica - Univ. "La Sapienza" Roma
 ============================================================================
 */

#include "VolatileBulletinBoard.h"

int main(int argc, char *argv[]) {

  printHeader();
  setDefaultValue();
  int parseMainResult = parseMainOptions_bb(argc, argv);
  //printf("parseMainResult: %d\n", parseMainResult);

  //Ignora comportamento di default per i segnali di interruzione utente
  ignoreTerminationSignal();

  if ( (( ((parseMainResult & 2)==2) ? 1 : 0 ) +
      ( ((parseMainResult & 4)==4) ? 1 : 0 ) +
      ( ((parseMainResult & 8)==8) ? 1 : 0 ) +
      ( ((parseMainResult & 16)==16) ? 1 : 0 )) >1 ) {
      //( ((parseMainResult & 32)==32) ? 1 : 0 )) >1 ) { //Per test (non implementato)
    //Specificato piu' di un paramentro tra -a, -d, -b, -l
    usage();
    printf("Errore: Specificare un solo parametro tra -a, -d, -b, -l oppure usare senza nessun parametro!\n");
    exit(-1);
  }
  else if ( ((parseMainResult & 2)==2) )  {
    //Creazione Bacheca
    printf("Creazione Bacheca: memoria condivisa e semafori...\n\n");
    dumpDefaultValues();
    int initbbResult = init_bb();
    //printf("initbbResult: %d\n", initbbResult);
    if (initbbResult==0)
    {
      printf("Memoria condivisa e samafori creati correttamente!\n");
    }
    else if (initbbResult==-1)
    {
      printf("ERRORE CREAZIONE MEMORIA CONDIVISA!!!\n");
    }
    else if (initbbResult==-2)
    {
      printf("ERRORE CREAZIONE SEMAFORI!!!\n");
    }

  }
  else if ( ((parseMainResult & 4)==4) )  {
    //Rimozione Bacheca
    printf("Rimozione Bacheca: memoria condivisa e semafori...\n\n");
    dumpDefaultKeysValues();
    int destroybbResult = destroy_bb();
    //printf("destroybbResult: %d\n", destroybbResult);
    if (destroybbResult==1)
    {
      printf("Memoria condivisa e samafori rimossi correttamente!\n");
    }
    if (destroybbResult & 2 )
    {
      printf("ERRORE RIMOZIONE MEMORIA CONDIVISA!!!\n");
    }
    else if (destroybbResult!=1)
    {
      printf("Memoria condivisa rimossa correttamente!\n");
    }
```

File: VolatileBulletinBoard.c

```
    73      if (destroybbResult & 4)
    74      {
    75        printf("ERRORE RIMOZIONE SEMAFORI!!!\n");
    76      }
    77      else if (destroybbResult!=1)
    78      {
    79        printf("Semafori rimossi correttamente!\n");
    80      }
    81
    82    }
    83    //else if ( ((parseMainResult & 32)==32) )  {
    84    //  //Test
    85    //  printf("Test Multithread...NON IMPLEMENTATO IN QUESTA VERSIONE!!\n\n");
    86    //  usage();
    87    //  exit(-1);
    88    //}
    89    else if ( ((parseMainResult & 8)==8) )  {
    90      //Backup File
    91      printf("Backup su file Bacheca...\n\n");
    92      dumpDefaultKeysValues();
    93      int openbbResult = open_bb();
    94      if (openbbResult==0)
    95      {
    96        printf("Memoria condivisa e samafori aperti correttamente!\n\n");
    97      }
    98      else if (openbbResult==-1)
    99      {
   100        printf("ERRORE APERTURA MEMORIA CONDIVISA!!!\n");
   101        exit(-1);
   102      }
   103      else if (openbbResult==-2)
   104      {
   105        printf("ERRORE APERTURA SEMAFORI!!!\n");
   106        exit(-1);
   107      }
   108      else if (openbbResult==-3)
   109      {
   110        printf("ERRORE COLLEGAMENTO MEMORIA CONDIVISA!!!\n");
   111        exit(-1);
   112      }
   113
   114      //Backup su file
   115      int resultBackup = fileBackupMessage_bb(FILE_MBULLETTINBOARD);
   116      if (resultBackup==-1)
   117      {
   118        printf("BACKUP SU FILE \"%s\" FALLITO!!\n\n", FILE_MBULLETTINBOARD);
   119      }
   120      else
   121      {
   122        printf("BACKUP SU FILE \"%s\" AVVENUTO CON SUCCESSO!!\n\n",
    FILE_MBULLETTINBOARD);
   123      }
   124
   125      detachSharedMemory();
   126
   127    }
   128    else if ( ((parseMainResult & 16)==16) )  {
   129      //Caricamento da File
   130      printf("Caricamento da file Bacheca...\n\n");
   131      dumpDefaultKeysValues();
   132      int openbbResult = open_bb();
   133      if (openbbResult==0)
   134      {
   135        printf("Memoria condivisa e samafori aperti correttamente!\n\n");
   136      }
   137      else if (openbbResult==-1)
   138      {
   139        printf("ERRORE APERTURA MEMORIA CONDIVISA!!!\n");
   140        exit(-1);
   141      }
   142      else if (openbbResult==-2)
   143      {
   144        printf("ERRORE APERTURA SEMAFORI!!!\n");
   145        exit(-1);
```

File: VolatileBulletinBoard.c

```
146       }
147       else if (openbbResult==-3)
148       {
149         printf("ERRORE COLLEGAMENTO MEMORIA CONDIVISA!!!\n");
150         exit(-1);
151       }
152
153       //Backup su file
154       int resultRestore = fileRestoreMessage_bb(FILE_MBULLETTINBOARD);
155       if (resultRestore==-1)
156       {
157         printf("CARICAMENTO DA FILE \"%s\" FALLITO!!\n\n", FILE_MBULLETTINBOARD);
158       }
159       else
160       {
161         printf("CARICAMENTO DA \"%s\" AVVENUTO CON SUCCESSO!!\n\n",
    FILE_MBULLETTINBOARD);
162       }
163
164       detachSharedMemory();
165
166     }
167     else if ( ((parseMainResult & 1)==1) )  {
168       //Gestione Bacheca
169       printf("Gestione Bacheca...\n\n");
170       dumpDefaultKeysValues();
171       int openbbResult = open_bb();
172       //printf("openbbResult: %d\n", openbbResult);
173       if (openbbResult==0)
174       {
175         printf("Memoria condivisa e samafori aperti correttamente!\n\n");
176       }
177       else if (openbbResult==-1)
178       {
179         printf("ERRORE APERTURA MEMORIA CONDIVISA!!!\n");
180         exit(-1);
181       }
182       else if (openbbResult==-2)
183       {
184         printf("ERRORE APERTURA SEMAFORI!!!\n");
185         exit(-1);
186       }
187       else if (openbbResult==-3)
188       {
189         printf("ERRORE COLLEGAMENTO MEMORIA CONDIVISA!!!\n");
190         exit(-1);
191       }
192
193       //Setta comportamento di default per i segnali di interruzione utente
194       defaultTerminationSignal();
195
196       //Lancio la gestione della bacheca
197       runSimpleBulettinboardManager();
198
199     }
200
201     return EXIT_SUCCESS;
202   }
203
204   void usage() {
205
206     printf("Usare: VolatileBulletinBoard [-a|-d|-b|-l|-t] [OPZIONI]\nOPZIONI:\n");
207     printf("  -a       Creazione Bacheca: memoria condivisa e semafori\n");
208     printf("  -d       Rimozione Bacheca: memoria condivisa e semafori\n");
209     printf("  -s N     setta chiave identificazione semafori\n");
210     printf("  -m N     setta chiave identificazione memoria condivisa\n");
211     printf("  -r N     massimo numero di utenti readers\n");
212     printf("  -n N     massimo numero di messaggi della bacheca\n");
213     printf("  -b file  backup memoria condivisa su file\n");
214     printf("  -l file  carica memoria condivisa da file backup\n");
215     //printf("  -t       test (NON IMPLEMENTATO IN QUESTA VERSIONE)\n");
216     printf("  -h       Visualizza questo help e termina il programma\n");
217     printf("\n");
218     printf("Non usando i parametri -a, -d, -b, -l viene eseguito il programma di
```

File: VolatileBulletinBoard.c

```
      gestione della bacheca.\n");
  219   printf("Non usando i parametri -s, -m, -r, -n vengono usati i valori di
      default.\n");
  220   printf("I parametri -r e -n vengono usati solamente insieme al parametro -a
      nella creazione bacheca, altrimenti vengono ignorati.\n");
  221   printf("\n");
  222
  223 }
  224
```

File: VolatileBulletinBoardUsers.c

```c
/*
 ============================================================================
 Name        : VolatileBulletinBoardUsers.c
 Author      : Pier Paolo Ciarravano
 Version     : 1.0 - 15/09/2008
 Copyright   : GPL - General Public License
 Description : Bacheca Elettronica Volatile in C, Ansi-style
               Corso di Sistemi Operativi I - Prof. A.Santoro 2007-2008
               Ingegneria Informatica - Univ. "La Sapienza" Roma

               VERSIONE PER LA SOLA LETTURA/SCRITTURA IN BACHECA
               SENZA POSSIBILITA' DI CREARE E RIMUOVERE MEMORIA CONDIVISA
               E SEMAFORI.
 ============================================================================
 */

#include "VolatileBulletinBoard.h"

int main(int argc, char *argv[]) {

  printHeader();
  setDefaultValue();
  int parseMainResult = parseMainOptions_bb(argc, argv);
  //printf("parseMainResult: %d\n", parseMainResult);

  //Ignora comportamento di default per i segnali di interruzione utente
  ignoreTerminationSignal();

  if (  ((parseMainResult & 2)==2) ||
      ((parseMainResult & 4)==4) ||
      ((parseMainResult & 8)==8) ||
      ((parseMainResult & 16)==16) ) {
    //Specificati paramentri -a, -d, -b, -l
    usage();
    printf("Errore: Questa versione non puo' utilizzare i parametri: -a, -d, -b, -l!\n");
    exit(-1);
  }
  else  if ( ((parseMainResult & 1)==1) ) {
    //Gestione Bacheca
    printf("Gestione Bacheca...\n\n");
    dumpDefaultKeysValues();
    int openbbResult = open_bb();
    //printf("openbbResult: %d\n", openbbResult);
    if (openbbResult==0)
    {
      printf("Memoria condivisa e samafori aperti correttamente!\n\n");
    }
    else if (openbbResult==-1)
    {
      printf("ERRORE APERTURA MEMORIA CONDIVISA!!!\n");
      exit(-1);
    }
    else if (openbbResult==-2)
    {
      printf("ERRORE APERTURA SEMAFORI!!!\n");
      exit(-1);
    }
    else if (openbbResult==-3)
    {
      printf("ERRORE COLLEGAMENTO MEMORIA CONDIVISA!!!\n");
      exit(-1);
    }

    //Setta comportamento di default per i segnali di interruzione utente
    defaultTerminationSignal();

    //Lancio la gestione della bacheca
    runSimpleBulettinboardManager();

  }

  return EXIT_SUCCESS;
}
```

File: VolatileBulletinBoardUsers.c

```
74
75 void usage() {
76
77   printf("Usare: VolatileBulletinBoard [OPZIONI]\nOPZIONI:\n");
78   printf("  -s N     setta chiave identificazione semafori\n");
79   printf("  -m N     setta chiave identificazione memoria condivisa\n");
80   printf("  -h       Visualizza questo help e termina il programma\n");
81   printf("\n");
82   printf("Non usando i parametri -s, -m vengono usati i valori di default.\n");
83   printf("\n");
84
85 }
86
```