Simple Memory Manager (smm): specifiche per il progetto d'esame (include esonero n.2)

Laboratorio di Sistemi Operativi I
Anno Accademico 2008-2009

Angelo Spognardi
(Tecnologie Informatiche)

Massimo Verola
(Informatica)

Obiettivo

- Questo modulo descrive le **specifiche del progetto** da svolgere per l'esame di Laboratorio di Sistemi Operativi I - AA 2008-2009.
- Tali specifiche si applicano anche al **compito di esonero n.2**, il quale pero' richiede l'implementazione di un numero inferiore di funzionalita'.

Il progetto richiede lo sviluppo di un semplice gestore di memoria, denominato `smm` (Simple Memory Manager)

NOTA BENE:

- Il progetto deve costituire **una creazione originale**, quindi non è possibile condividere parti di codice o di relazione con altri studenti/gruppi, o copiare contenuti derivanti da altre fonti.
- Il progetto puo' essere svolto individualmente o in gruppo (**max 3 persone**): i gruppi composti da 2 oppure da 1 persona otterranno un **bonus** crescente da aggiungere alla valutazione del compito (quindi un compito perfetto di un gruppo da 3 persone vale 30 punti, mentre per gruppi meno numerosi vale > 30 punti).
- Per gli appelli di Gen-Feb 2009 si richiede il soddisfacimento delle sole **specifiche di base** (Esonero n.2), mentre per quelli successivi si applicano anche le **specifiche estese** (Progetto).

Specifiche di base

valide

per l'Esonero n.2

Specifiche di base - I

Il software package Simple Memory Manager `smm`:

- dovra' emulare il comportamento di un **gestore di memoria:**
  - **a partizionamento dinamico** (*dynamic partitioning*)
  - **con algoritmo di selezione dell'area** (*placement algorithm*) di tipo ***best-fit***
  - **senza swapping out** su file

  (vedasi W. Stallings, "Operating Systems", cap. 7)
- dovra' essere implementato come un'**applicazione di tipo server** di nome **`smmd`,** consistente in un eseguibile Linux (cioe' NON come modulo kernel) **che gestisce uno spazio di memoria contiguo allocato nello heap del processo server**, denominato **`smmMem`**, di dimensione fissa pari a **16 MB**
- dovra' fornire un'**API per l'allocazione/deallocazione di blocchi di memoria all'interno di `smmMem` e relativo accesso ad essi** da parte delle applicazioni client mediante una **libreria di nome `libsmm.a`**
- dovra' implementare un meccanismo basato sullo **scambio di messaggi via FIFO** tra le applicazioni client e il programma server, che supporti le richieste di allocazione/deallocazione/accesso di/a blocchi di memoria in **`smmMem`**

Specifiche di base – II

- dovra' implementare un **meccanismo di protezione** della memoria gestito da **`smmd`** per impedire che una qualsiasi applicazione possa accedere a locazioni di memoria contenute nei blocchi allocati dalle altre applicazioni ovvero a spazi di memoria liberi
- dovra' provvedere al **compattamento** dei blocchi allocati in **`smmMem`** ogni volta che **`smmd`** riceve una richiesta di allocazione che puo' essere soddisfatta soltanto **rendendo contigui** spazi liberi di memoria altrimenti separati
- dovra' fornire una **programma interattivo,** di nome **`smmMon`, per l'avvio/terminazione di `smmd` e per il monitoraggio a *run-time* di `smmMem`**

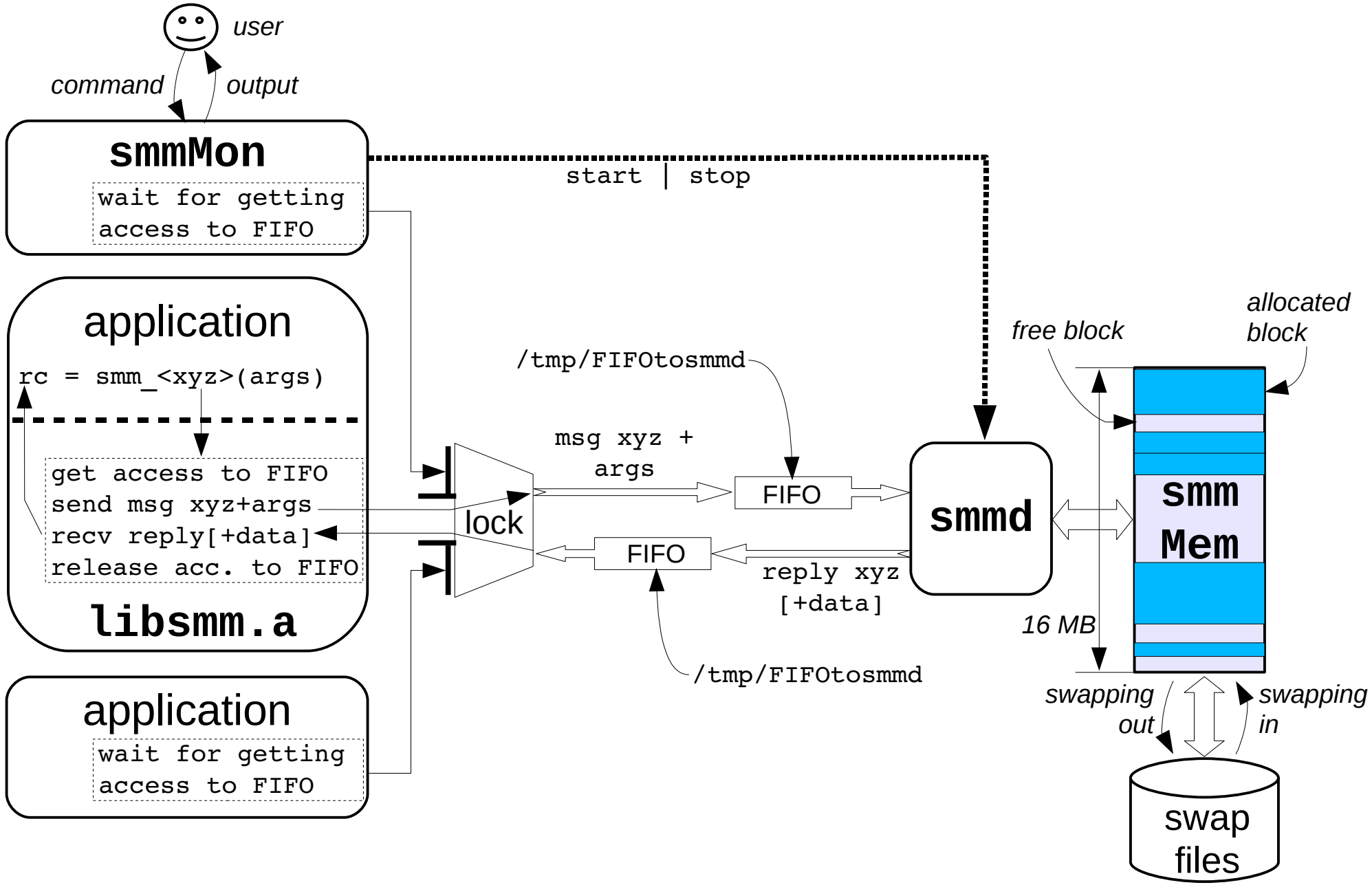
Specifiche implementative

- il server **`smmd`** dovra' **memorizzare, organizzare e mantenere integre** le informazioni relative allo stato di allocazione di **`smmMem`** mediante opportune strutture di dati
- lo scambio di messaggi tra applicazioni e server **`smmd`** deve avvenire tramite **una coppia di FIFO** per la comunicazione full-duplex:
  `/tmp/FIFOtosmmd` : richieste da applicazione a **`smmd`**
  `/tmp/FIFOfromsmmd` : risposte da **`smmd`** ad applicazione
- il server **`smmd`** alla partenza deve creare l'area di memoria **`smmMem`**, deve creare e inizializzare le strutture dati necessarie a mantenere il controllo di **`smmMem`**, e infine attendere le richieste da parte delle applicazioni
- il server **`smmd,`** appena giunge una richiesta, la analizza e, se valida, esegue l'operazione richiesta, ritornando SEMPRE un messaggio al richiedente, che comunica il codice di ritorno (successo/errore) ed eventuali dati di output

Specifiche implementative

- la libreria **`libsmm.a`** dovra' implementare all'interno delle sue funzioni un **meccanismo di lock esclusivo** delle FIFO per garantire l'atomicita' della procedura <*invio richiesta-ricezione risposta*>
- tutti i puntatori gestiti dalla libreria **`libsmm.a`** alle locazioni di memoria in **`smmMem`** (cioe' quelli che compaiono come argomenti o valori di ritorno delle varie funzioni) devono essere indirizzi logici che non coincidono necessariamente con i corrispondenti indirizzi utilizzati da **`smmd`**; cio' e' richiesto per permettere la rilocazione dei blocchi a seguito di un compattamento o ad una operazione di *swapping out* + *swapping in* (vedi dopo Progetto)
- la terminazione del server **`smmd`** deve avvenire a seguito dell'**invio del segnale `SIGUSR1`**, che attiva il cleanup delle strutture dati e procedure in corso e poi l'exit

Schema architetturale



Funzioni da implementare in `libsmm.a` – I

Nota:

- tutti i tipi NON definiti dall'ANSI C (typedef tipo `size_t, pid_t...`) devono essere identici a quelli definiti per Linux, quindi ci si deve riferire agli opportuni include file.

Funzioni di allocazione

- `void *smm_malloc(size_t size);`
  – alloca un blocco di memoria in `smmMem` di dimensione pari a `size` byte e ritorna un puntatore all'inizio del blocco. La memoria allocata NON viene azzerata. `size` deve essere > 0. Ritorna `NULL` in caso di errore.
- `void *smm_calloc(size_t nmemb, size_t size);`
  – alloca un blocco di memoria in `smmMem` per un array di `nmemb` elementi di dimensione pari a `size` byte ognuno, e ritorna un puntatore all'inizio del blocco. La memoria allocata e' inizializzata a 0. `nmemb` e `size` devono essere > 0. Ritorna `NULL` in caso di errore.
- `void *smm_realloc(void *ptr, size_t size);`
  – cambia a `size` byte la dimensione del blocco di memoria gia' allocato in `smmMem` e puntato da `ptr` e ritorna un puntatore all'inizio del blocco. Il contenuto rimane invariato per la porzione di memoria di dimensione pari al minimo tra la vecchia e la nuova dimensione; eventuale memoria allocata in aggiunta risultera' non inizializzata. Se il blocco puntato richiede uno spostamento (puntatore ritornato diverso da `ptr`), la funzione garantisce la liberazione del vecchio puntatore, cioe' una `free(ptr)` viene effettuata. Ritorna `NULL` in caso di errore.

Funzioni da implementare in `libsmm.a` - II

Funzione di deallocazione

- `int smm_free(void *ptr);`
  – libera il blocco di memoria in `smmMem` precedentemente allocato e puntato dal puntatore `ptr`. Nel caso `ptr` sia un puntatore non valido (inesistente o gia' liberato da una `free(ptr)`, la chiamata deve ritornare un codice di errore (<0), o altrimenti di successo (=0).

Funzioni generali di scrittura/lettura a/da memoria

- `int smm_put(void *buf, void *ptr, size_t size);`
  – inserisce a partire dal puntatore `ptr` appartenente a un blocco di memoria precedentemente allocato in `smmMem` un numero di byte pari a `size` copiandoli dal buffer interno all'applicazione puntato da `buf`. `size` deve essere > 0. Ritorna un codice di successo (=0) o errore (<0).
- `int smm_get(void *ptr, void *buf, size_t size);`
  – preleva a partire dal puntatore `ptr` appartenente a un blocco di memoria precedentemente allocato in `smmMem` un numero di byte pari a `size` e li inserisce nel buffer interno all'applicazione puntato da `buf`. `size` deve essere > 0. Ritorna un codice di successo (=0) o errore (<0).

Funzioni da implementare in `libsmm.a` - III

Funzioni di verifica dei puntatori alla memoria

- `int smm_chk_ptr(void *ptr, struct smm_blk *blk);`

```
struct smm_blk {
  void *start; /* pointer to the first byte of the block */
  void *last;  /* pointer to the last byte of the block */
  char  swap;  /* swapped in=0, swapped out=1 */
};
```

– rileva le informazioni relative al blocco a cui appartiene `ptr`, riempiendo opportunamente i campi della struttura pre-allocata dal chiamante. Il campo `swap` deve essere aggiornato solo se si e' implementato lo *swapping out* (vedi dopo Progetto) altrimenti va ignorato. Nel caso `ptr` sia un puntatore non valido, la chiamata deve ritornare un codice di errore (<0), o altrimenti di successo (=0) .

Programma di monitoraggio della memoria `smmMon` - I

- Il programma **`smmMon`** presenta un prompt all'utente e, iterativamente, rimane in attesa dell'inserimento di un comando, il quale viene validato e poi eseguito.
- L'output risultante viene mostrato a video.
- Tutti i puntatori devono essere espressi (sia input che output) in formato esadecimale (0x seguito da 8 caratteri, ad es. `0x1234ABCD`).
- Un comando non valido deve generare un messaggio di errore.

Comandi da implementare in `smmMon - I`

Nota 1: I puntatori presenti nei seguenti comandi sono indirizzi logici (come spiegato a pag. 7) relativi all'applicazione `smmMon`.

- **`start [-b | -d]`** – avvia il server `smmd` e stampa messaggio di successo/errore.
- **`stop`** – termina il server `smmd` e stampa messaggio di successo/errore.
- **`exit`** – esce da `smmMon`.
- **`alloc`** ***`size`*** – richiede l'allocazione di un blocco di memoria di `size` byte (in decimale) e stampa il valore del puntatore al blocco o messaggio di errore.
- **`free`** ***`ptr`*** – richiede la deallocazione del blocco di memoria puntato dal puntatore di valore `ptr` e stampa messaggio di successo/errore.
- **`check`** ***`ptr`*** – stampa i puntatori al primo e ultimo byte del blocco che contiene `ptr` o messaggio di errore se `ptr` non e' valido.
  Se e' stato implementato lo *swapping out*, indica anche se il blocco e' *swapped in* o *swapped out* (vedi dopo Progetto).

Comandi da implementare in `smmMon` - II

Nota 2: I puntatori presenti nei seguenti comandi **NON** sono gli indirizzi logici (come spiegato a pag. 7) utilizzati dall'applicazione `smmMon` **MA** sono gli indirizzi utilizzati direttamente da `smmd` per accedere a `smmMem`.

- **`lab [`*`PID`*`] –`** (*list allocated blocks*) stampa i valori dei puntatori al primo e ultimo byte e della dimensione in byte (in decimale) di `smmMem` e di tutti i blocchi allocati dalle varie applicazioni con relativo PID.
  Se viene inserito un numero di PID valido come argomento opzionale, stampa solo le informazioni sui blocchi di memoria allocati dal processo identificato da PID.
  Se il PID non e' valido, stampa un messaggio di errore.
  Se e' stato implementato lo *swapping out*, indica anche se il blocco e' *swapped in* o *swapped out* (vedi dopo Progetto).
- **`lfb –`** (*list free blocks*) stampa i valori dei puntatori al primo e ultimo byte e della dimensione in byte (in decimale) di tutte le aree libere di `smmMem`.

Note sulle specifiche

- L'implementazione **corretta** di funzionalità aggiuntive rispetto a quelle richieste verrà considerato come un **bonus** da aggiungere al punteggio:
  ad es. consegnare il compito come Esonero n.2 avendo anche inserito una delle funzionalita' estese del Progetto (vedi dopo) oppure consegnare il compito come Progetto avendo inserito entrambe le funzionalita' estese.
- Si raccomanda di **sviluppare e validare** prima tutte le funzionalita’ di base, poi la funzionalita' aggiuntiva obbligatoria per il Progetto (vedi dopo) ed infine la seconda eventuale funzionalita' opzionale.

Package da consegnare - I

- La compilazione degli eseguibili e librerie del package **`smm`** deve essere gestita mediante **`Makefile`**
- L'installazione degli eseguibili **`smmd, smmMon`** e libreria **`libsmm.a`** deve avvenire nella directory **`$HOME/smm`** e **NON** deve richiedere il privilegio di **`root`**
- Il progetto deve essere documentato mediante una relazione che descrive:
  - l'organizzazione del package: albero delle directory, identificazione e caratterizzazione dei file sorgenti (.c) e include (.h) (funzionalita' assegnate ai vari sorgenti, definizione e ripartizione dei dati globali, tipi definiti di dati, ...)
  - le considerazioni per l'installazione e il test con eventuali esempi di comandi da effettuare
  - le eventuali funzionalita' aggiuntive opzionali implementate
  - la struttura dei programmi **`smmd`** e **`smmMon`** con il dettaglio delle principali scelte implementative (strutture di dati, algoritmi, flussi di controllo a livello macro, ...)
  - lla struttura delle funzioni della libreria **`libsmm.a`**
  - il protocollo di comunicazione via FIFO (struttura e tipo dei messaggi di richiesta e di risposta)
  - il meccanismo di controllo per l'accesso esclusivo alle FIFO
  - la gestione degli errori con la lista dei codici di errore definiti
  - le eventuali limitazioni dovute all'implementazione

Package da consegnare - II

- La consegna deve consistere in un package, da spedire via mail, in formato tar compresso (tramite comando **`tar czf`**) col nome **`smm.tgz`**, contenente i sorgenti C, gli include file, il **`Makefile`**, la documentazione ed eventuali file per il test
- il file **`smm.tgz`** deve contenere una sola directory (con pathname relativo) di nome **`smm`**
- la directory **`smm`** deve contenere:
  - file **`AUTHORS`** (Cognome Nome Matricola, uno studente per linea)
  - file **`README`** (informazioni molto concise sulla compilazione, installazione e lancio del programma)
  - directory **`doc`** con relazione in formato PDF o OpenOffice
  - directory **`src`** (sorgenti .c), **`inc`** (include **`.h`**), **`test`** (file/programmi di test), **`bin`** (eseguibili generati), **`lib`** (librerie generate), **`obj`** (file compilati **`.o`**), ed eventuali altre directory

Specifiche estese

valide

per il Progetto

Specifiche estese

- Per il progetto, **in aggiunta a tutto quanto richiesto per l'Esonero n.2**, si deve implementare **una funzionalità ulteriore** tra le due sotto proposte.

1) partizionamento della memoria **`smmMem`** di tipo **Buddy System;** in questo caso **`smmd`** dovra' adottare il **partizionamento dinamico** o il **Buddy System** a seconda del valore dell'opzione passata al lancio del programma:
`smmd -b` (Buddy System) oppure **`smmd -d`** (partizionamento dinamico)

2) **Swapping out** su file di partizioni intere per soddisfare tutte le richieste di allocazione, anche nei casi in cui non ci siano aree libere di dimensione adeguata (anche dopo eventuale compattamento) in **`smmMem.`**
I file necessari per lo *swapping out* devono risiedere nella stessa directory in cui e' presente l'eseguibile **`smmd.`**
L'organizzazione dei file utilizzati per lo *swapping out* e il criterio con il quale selezionare la partizione da sottoporre a *swapping out* deve essere definito dallo studente e documentato nella relazione.
Una partizione in stato di *swapping out* deve essere ricaricata (*swapping in*) in **`smmMem`** (procedendo eventualmente ad uno *swapping out* di un'altra partizione) appena **`smmd`** riceve una richiesta di accesso (**`put`** o **`get`** di dati, vedasi funzioni di **`libsmm.a`**) a un quasiasi dato di tale partizione.

Funzioni ulteriori da implementare in `libsmm.a -l`

Funzioni specifiche di scrittura/lettura a/da memoria di scalari

- `int smm_int_put(void *ptr, int value);`
  – inserisce il valore intero `value` nel puntatore `ptr` appartenente a un blocco di memoria precedentemente allocato in `smmMem`. Ritorna un codice di successo (=0) o errore (<0).
- `int smm_int_get(void *ptr, int *valuep);`
  – preleva dal puntatore `ptr` appartenente a un blocco di memoria precedentemente allocato in `smmMem` un valore intero e lo memorizza nell'indirizzo `valuep`. Ritorna un codice di successo (=0) o errore (<0).
- `int smm_float_put(void *ptr, float value);`
  – inserisce il valore float `value` nel puntatore `ptr` appartenente a un blocco di memoria precedentemente allocato in `smmMem`. Ritorna un codice di successo (=0) o errore (<0).
- `int smm_float_get(void *ptr, float *valuep);`
  – preleva dal puntatore `ptr` appartenente a un blocco di memoria precedentemente allocato in `smmMem` un valore float e lo memorizza nell'indirizzo `valuep`. Ritorna un codice di successo (=0) o errore (<0).

Funzioni ulteriori da implementare in `libsmm.a` - II

Funzioni specifiche di scrittura/lettura a/da memoria di vettori

- `int smm_intarr_put(void *ptr, int *arr, size_t nmemb);`
  – inserisce `nmemb` interi dell'array puntato dal puntatore interno all'applicazione `arr` a partire dal puntatore `ptr` appartenente a un blocco di memoria precedentemente allocato in `smmMem`. `nmemb` deve essere > 0. Ritorna un codice di successo (=0) o errore (<0).
- `int smm_intarr_get(void *ptr, int *arr, size_t nmemb);`
  – preleva a partire dal puntatore `ptr` appartenente a un blocco di memoria precedentemente allocato in `smmMem` un numero di interi pari a `nmemb` e li inserisce nel buffer interno all'applicazione puntato da `arr`. `nmemb` deve essere > 0. Ritorna un codice di successo (=0) o errore (<0).
- `int smm_floatarr_put(void *ptr, float *arr, size_t nmemb);`
  – inserisce `nmemb` float dell'array puntato dal puntatore interno all'applicazione `arr` a partire dal puntatore `ptr` appartenente a un blocco di memoria precedentemente allocato in `smmMem`. Ritorna un codice di successo (=0) o errore (<0).
- `int smm_floatarr_get(void *ptr, float *arr, size_t nmemb);`
  – preleva a partire dal puntatore `ptr` appartenente a un blocco di memoria precedentemente allocato in `smmMem` un numero di float pari a `nmemb` e li inserisce nel buffer interno all'applicazione puntato da `arr`. `nmemb` deve essere > 0. Ritorna un codice di successo (=0) o errore (<0).

Nota sul programma di monitoraggio della memoria `smmMon`

La specifica del comando **`start`** viene ampliata come segue:

- **`start [-b | -d] -`** avvia il server `smmd` e stampa messaggio di successo/errore. Se e' stata implementata anche la gestione di tipo Buddy System, l'argomento opzionale indica quale tipo di gestione adottare: **`-b`**=Buddy System, **`-d`**=partizionamento dinamico.

Nota sul package da consegnare

- Nella relazione indicare chiaramente quale funzionalita' aggiuntiva obbligatoria per il progetto e' stata scelta e descriverne la logica interna dell'implementazione