SMM Simple Memory Manager

Lista dei test case

MAK - Makefile

Tag	Sequenza di comandi	Risultato atteso
MAK.ALL	***Compilazione***	
MAK.ALL.01	make *oppure* make all	Compila senza errori e genera lib/libsmm.a, bin/smmMon, bin/smmd.
MAK.ALL.02	make *oppure* make all	Non ricompila, poiché i target sono stati già generati dal comando precedente.
MAK.ALL.03	touch src/*.c	Comando necessario per aggiornare la data dei sorgenti e forzare il make a ricompilare di nuovo.
MAK.ALL.04	make *oppure* make all	Compila senza errori e genera lib/libsmm.a, bin/smmMon, bin/smmd.
MAK.ALL.05	touch inc/*.h	Comando necessario per aggiornare la data degli include file e forzare il make a ricompilare di nuovo.
MAK.ALL.06	make *oppure* make all	Compila senza errori e genera lib/libsmm.a, bin/smmMon, bin/smmd.
MAK.INS	***Installazione***	
MAK.INS.01	make install	Copia lib/libsmm.a, bin/smmMon, bin/smmd in $HOME/smm.
MAK.INS.02	make install	Non ripete la copia di lib/libsmm.a, bin/smmMon, bin/smmd poiché già presenti in $HOME/smm.
MAK.CLN	***Cleanup***	
MAK.CLN.01	make clean	Rimuove obj/*.o, lib/libsmm.a, bin/smmMon, bin/smmd.

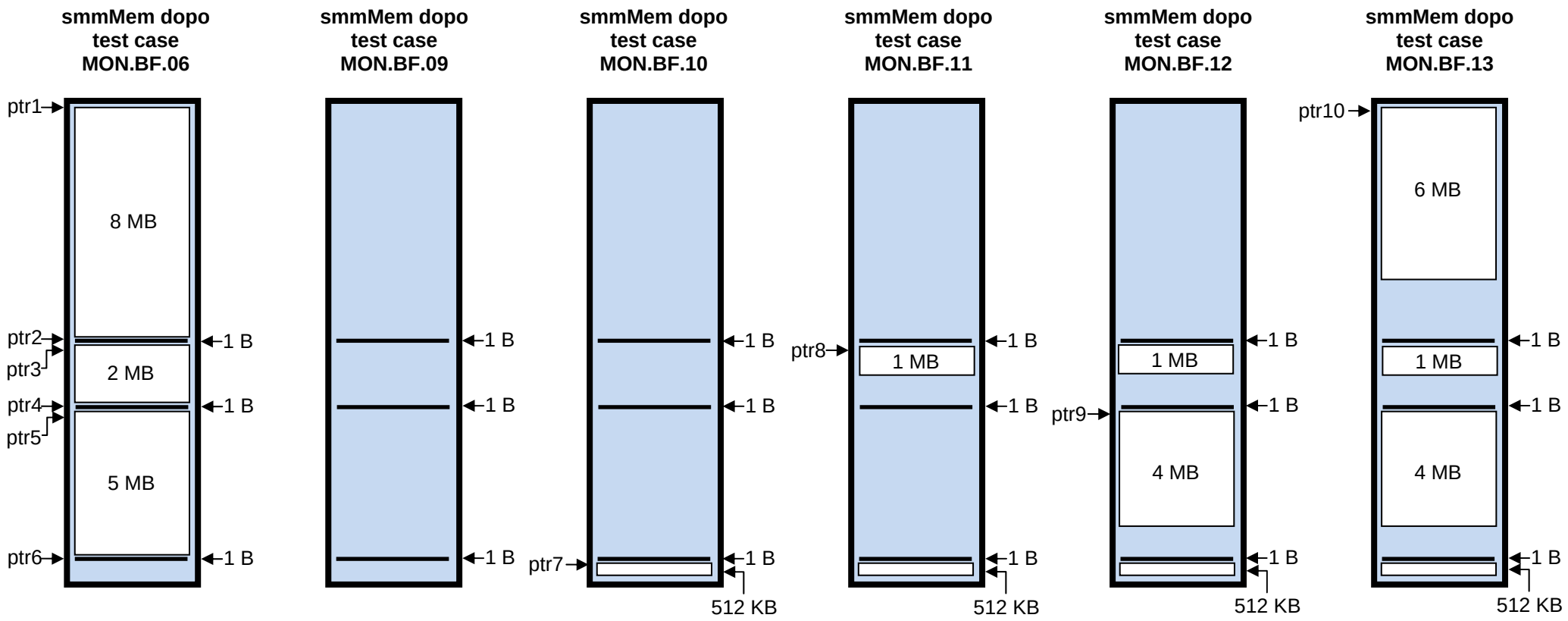
SRV - smmd

Tag	Sequenza di comandi	Risultato atteso
SRV.REX	***Lancio e terminazione (Run+EXit)***	
SRV.REX.01	smmd	smmd va in esecuzione (potrebbero comparire le 2 FIFO in /tmp).
SRV.REX.02	kill -USR1 <PID smmd>	smmd termina (vengono rimosse eventuali FIFO in /tmp).

MON - smmMon		
Tag	**Sequenza di comandi**	**Risultato atteso**
MON.REX	***Lancio e terminazione smmd (start+stop)***	
MON.REX.01	smmMon	smmMon va in esecuzione e presenta il prompt all'utente (potrebbero comparire le 2 FIFO in /tmp).
MON.REX.02	> exit	smmMon termina (vengono rimosse eventuali FIFO in /tmp).
MON.REX.03	smmMon	smmMon va in esecuzione e presenta il prompt all'utente.
MON.REX.04	> start	smmd va in esecuzione; deve ritornare disponibile il prompt di smmMon.
MON.REX.05	> stop	smmd viene terminato.
MON.REX.06	> start	smmd va in esecuzione; deve ritornare disponibile il prompt di smmMon.
MON.REX.07	> exit	Viene visualizzato un messaggio di errore poiché smmd è ancora attivo, *oppure* smmd viene automaticamente terminato (implicita esecuzione di stop) e poi smmMon termina. *oppure* smmd viene lasciato attivo, ma il suo PID salvato su file e poi smmMon termina. In quest'ultimo caso si deve lanciare nuovamente smmMon e verificare che non venga mandato in esecuzione un secondo smmd.
MON.INI	***Fase di inizializzazione dei test case per alloc+free***	
MON.INI.01	smmMon	smmMon va in esecuzione.
MON.INI.02	> start	smmd va in esecuzione.
MON.AFOK	***Test case alloc+free che devono andare a buon fine***	
MON.AFOK.01	> lab	Vengono stampati: • i valori dei puntatori utilizzati direttamente da smmd per indirizzare il primo e l'ultimo byte dell'intera smmMem e la sua dimensione in byte=16777216.
MON.AFOK.02	> lfb	Vengono stampati: • i valori dei puntatori utilizzati direttamente da smmd per indirizzare il primo e l'ultimo byte dell'area libera (che in questo caso coincide con l'intera smmMem) e la sua dimensione in byte=16777216.
MON.AFOK.03	> alloc 1	Viene ritornato un valore di puntatore <ptr> a un blocco di 1 B.
MON.AFOK.04	> lab	Vengono stampati: • i valori dei puntatori utilizzati direttamente da smmd per indirizzare il primo e l'ultimo byte dell'intera smmMem e la sua dimensione in byte=16777216. • valori dei puntatori utilizzati direttamente da smmd per indirizzare il primo e l'ultimo byte (che in questo caso devono coincidere) dell'unico blocco allocato, la sua dimensione in byte=1 e il PID di smmMon.
MON.AFOK.05	> lab <PID di smmMon>	Come comando lab precedente, ma senza le info su smmMem.
MON.AFOK.06	> lfb	Come comando lfb precedente, ma con dimensione area libera=16777215.
MON.AFOK.07	> free <ptr>	Viene stampato un messaggio di successo.
MON.AFOK.08	> alloc 1024	Viene ritornato un valore di puntatore <ptr> a un blocco di 1 KB.
MON.AFOK.09	> lab <PID di smmMon>	Come comando lab precedente, ma con dimensione blocco=1024.
MON.AFOK.10	> lfb	Come comando lfb precedente, ma con dimensione area libera=16776192.
MON.AFOK.11	> free <ptr>	Viene stampato un messaggio di successo.
MON.AFOK.12	> alloc 1048576	Viene ritornato un valore di puntatore <ptr> a un blocco di 1 MB.
MON.AFOK.13	> lab <PID di smmMon>	Come comando lab precedente, ma con dimensione blocco=1048576.
MON.AFOK.14	> lfb	Come comando lfb precedente, ma con dimensione area libera=15728640.
MON.AFOK.15	> free <ptr>	Viene stampato un messaggio di successo.
MON.AFOK.16	> alloc 16777216	Viene ritornato un valore di puntatore <ptr> a un blocco di 16 MB.
MON.AFOK.17	> lab <PID di smmMon>	Come comando lab precedente, ma con dimensione blocco=16777216.
MON.AFOK.18	> lfb	Come comando lfb precedente, ma con dimensione area libera=0.
MON.AFOK.19	> free <ptr>	Viene stampato un messaggio di successo.
MON.AFOK.20	> lab <PID di smmMon>	Non ci deve essere nessun blocco da elencare.
MON.AFOK.21	> lfb	Come comando lfb precedente, ma con dimensione area libera=16777216.

MON - smmMon		
Tag	**Sequenza di comandi**	**Risultato atteso**
MON.AFER	***Test case alloc+free che devono generare messaggi di errore*** *Nota: i valori errati dei parametri di input NON devono essere rilevati dal codice di smmMon, ma dalle funzioni di libsmm.a, che vengono chiamate da smmMon e restituiscono un codice di errore, in base al quale smmMon stamperà a video un messaggio di errore opportuno.*	
MON.AFER.01	> alloc	Viene stampato un messaggio di errore poiché manca il parametro size.
MON.AFER.02	> free	Viene stampato un messaggio di errore poiché manca il parametro ptr.
MON.AFER.03	> alloc 0	Viene stampato un messaggio di errore poiché la dimensione minima di blocco è 1.
MON.AFER.04	> alloc 16777217	Viene stampato un messaggio di errore poiché la dimensione massima di blocco è 16777216.
MON.AFER.05	> free 0x12345678	Viene stampato un messaggio di errore, poiché 0x12345678 è un puntatore non valido (non c'è neppure un blocco allocato).
MON.AFER.06	> lfb	Come comando lfb precedente, ma con dimensione area libera=16777216.
MON.AFER.07	> alloc 8388608	Viene ritornato un valore di puntatore <ptr> a un blocco di 8 MB.
MON.AFER.08	> alloc 8388609	Viene stampato un messaggio di errore poiché la dimensione dell'area libera è 8388608.
MON.AFER.09	> free <ptr>	Viene stampato un messaggio di successo.
MON.AFER.10	> free <ptr>	Viene stampato un messaggio di errore, poiché <ptr> è già stato liberato.
MON.CHK	***Test case che verificano la funzione check ptr***	
MON.CHK.01	> alloc 4194304	Viene ritornato un valore di puntatore <ptr> a un blocco di 4 MB.
MON.CHK.02	> check <ptr>	Vengono stampati i puntatori di inizio e fine blocco, cioè <ptr> e <ptr+0x003FFFFF>.
MON.CHK.03	> check <ptr+0x00000001>	Vengono stampati i puntatori di inizio e fine blocco, cioè <ptr> e <ptr+0x003FFFFF>.
MON.CHK.04	> check <ptr+0x00200000>	Vengono stampati i puntatori di inizio e fine blocco, cioè <ptr> e <ptr+0x003FFFFF>.
MON.CHK.05	> check <ptr+0x003FFFFE>	Vengono stampati i puntatori di inizio e fine blocco, cioè <ptr> e <ptr+0x003FFFFF>.
MON.CHK.06	> check <ptr+0x003FFFFF>	Vengono stampati i puntatori di inizio e fine blocco, cioè <ptr> e <ptr+0x003FFFFF>.
MON.CHK.07	> check <ptr+0x00400000>	Viene stampato un messaggio di errore, poiché <ptr+0x00400000> non appartiene a nessun blocco allocato.
MON.CHK.08	> check <ptr+0x00800000>	Viene stampato un messaggio di errore, poiché <ptr+0x00800000> non appartiene a nessun blocco allocato.
MON.CHK.09	> check <ptr+0x01000000>	Viene stampato un messaggio di errore, poiché <ptr+0x01000000> non appartiene a nessun blocco allocato.
MON.CHK.10	> check	Viene stampato un messaggio di errore poiché manca il parametro ptr.
MON.CHK.11	> free <ptr>	Viene stampato un messaggio di successo.
MON.CPT	***Test case che verificano la procedura di compattamento di smmMem***	
MON.CPT.01	> alloc 4194304	Viene ritornato un valore di puntatore <ptr1> a un blocco di 4 MB.
MON.CPT.02	> alloc 4194304	Viene ritornato un valore di puntatore <ptr2> a un blocco di 4 MB.
MON.CPT.03	> alloc 4194304	Viene ritornato un valore di puntatore <ptr3> a un blocco di 4 MB.
MON.CPT.04	> alloc 4194304	Viene ritornato un valore di puntatore <ptr4> a un blocco di 4 MB.
MON.CPT.05	> lab <PID di smmMon>	Vengono stampate le info sui 4 blocchi allocati, ognuno da 4 MB.
MON.CPT.06	> lfb	Non ci deve essere nessun blocco dell'area libera da elencare.
MON.CPT.07	> free <ptr1>	Viene stampato un messaggio di successo.
MON.CPT.08	> alloc 2097152	Viene ritornato un valore di puntatore <ptr5> a un blocco di 2 MB.
MON.CPT.09	> lab <PID di smmMon>	Vengono stampate le info sui 4 blocchi allocati, 1 da 2 MB e 3 da 4 MB.
MON.CPT.10	> lfb	Vengono stampate le info su un'area libera da 2 MB.
MON.CPT.11	> free <ptr3>	Viene stampato un messaggio di successo.
MON.CPT.12	> lfb	Vengono stampate le info su 2 aree libere da 2 MB e da 4 MB *oppure* su un'unica area da 6MB, se si è già proceduto al compattamento.

MON - smmMon		
Tag	**Sequenza di comandi**	**Risultato atteso**
MON.CPT.13	> alloc 5242880	Viene ritornato un valore di puntatore <ptr6> a un blocco di 5 MB (deve esserci stato un compattamento adesso o in precedenza per rendere contigui il blocco libero da 2 MB a quello da 4 MB).
MON.CPT.14	> lab <PID di smmMon>	Vengono stampate le info sui 4 blocchi allocati, 1 da 2 MB, 2 da 4 MB e 1 da 5 MB.
MON.CPT.15	> lfb	Vengono stampate le info su un'area libera da 1 MB.
MON.CPT.16	> free <ptr2>	Viene stampato un messaggio di successo.
MON.CPT.17	> free <ptr4>	Viene stampato un messaggio di successo.
MON.CPT.18	> free <ptr5>	Viene stampato un messaggio di successo.
MON.CPT.19	> free <ptr6>	Viene stampato un messaggio di successo.
MON.CPT.20	> lab	Vengono stampate solo le usuali info su smmMem, ma non ci devono essere blocchi allocati.
MON.CPT.21	> lfb	Vengono stampate le info su un'area libera da 16 MB.
MON.BF	***Test case che verificano l'algoritmo best-fit***	
MON.BF.01	> alloc 8388608	Viene ritornato un valore di puntatore <ptr1> a un blocco di 8 MB.
MON.BF.02	> alloc 1	Viene ritornato un valore di puntatore <ptr2> a un blocco di 1 B.
MON.BF.03	> alloc 2097152	Viene ritornato un valore di puntatore <ptr3> a un blocco di 2 MB.
MON.BF.04	> alloc 1	Viene ritornato un valore di puntatore <ptr4> a un blocco di 1 B.
MON.BF.05	> alloc 5242880	Viene ritornato un valore di puntatore <ptr5> a un blocco di 5 MB.
MON.BF.06	> alloc 1	Viene ritornato un valore di puntatore <ptr6> a un blocco di 1 B.
MON.BF.07	> free <ptr1>	Viene stampato un messaggio di successo.
MON.BF.08	> free <ptr3>	Viene stampato un messaggio di successo.
MON.BF.09	> free <ptr5>	Viene stampato un messaggio di successo.
MON.BF.10	> alloc 524288	Viene ritornato un valore di puntatore <ptr7> a un blocco di 512 KB.
MON.BF.11	> alloc 1048576	Viene ritornato un valore di puntatore <ptr8> a un blocco di 1 MB.
MON.BF.12	> alloc 4194304	Viene ritornato un valore di puntatore <ptr9> a un blocco di 4 MB.
MON.BF.13	> alloc 6291456	Viene ritornato un valore di puntatore <ptr10> a un blocco di 6 MB.
MON.BF.14	> lab	Verificare che la sequenza dei blocchi allocati rispetti il best-fit, cioè: blocco da 6 MB puntato da ptr10 blocco da 1 B puntato da ptr2 blocco da 1 MB puntato da ptr8 blocco da 1 B puntato da ptr4 blocco da 4 MB puntato da ptr9 blocco da 1 B puntato da ptr6 blocco da 512 KB puntato da ptr7
MON.BF.15	> lfb	Verificare che le aree libere siano complementari ai blocchi allocati
MON.BF.16	> free <ptr2>	Viene stampato un messaggio di successo.
MON.BF.17	> free <ptr4>	Viene stampato un messaggio di successo.
MON.BF.18	> free <ptr6>	Viene stampato un messaggio di successo.
MON.BF.19	> free <ptr7>	Viene stampato un messaggio di successo.
MON.BF.20	> free <ptr8>	Viene stampato un messaggio di successo.
MON.BF.21	> free <ptr9>	Viene stampato un messaggio di successo.
MON.BF.22	> free <ptr10>	Viene stampato un messaggio di successo.
MON.BF.23	> lfb	Verificare che non ci siano blocchi allocati


smmMem dopo
test case
MON.BF.06
ptr1
8 MB
ptr2
ptr3
2 MB
ptr4
ptr5
5 MB
ptr6
1 B
1 B
1 B
smmMem dopo
test case
MON.BF.09
1 B
1 B
1 B
smmMem dopo
test case
MON.BF.10
1 B
1 B
ptr7
1 B
512 KB
smmMem dopo
test case
MON.BF.11
ptr8
1 MB
1 B
1 B
1 B
512 KB
smmMem dopo
test case
MON.BF.12
1 MB
ptr9
4 MB
1 B
1 B
1 B
512 KB
smmMem dopo
test case
MON.BF.13
ptr10
6 MB
1 MB
4 MB
1 B
1 B
1 B
512 KB

LIB - libsmm.a

Tag	Sequenza di comandi	Risultato atteso	Descrizione
Nota: si assume che siano definite le seguenti variabili: void *ptr, *ptr1, *ptr2, *ptr3, *ptr4, *ptr5, *ptr6; int rc; struct smm_blk blk;			
LIB.AFOK	***Test case che verificano le funzioni smm_malloc(), smm_free(), smm_chk_ptr() e che devono andare a buon fine***		
LIB.AFOK.01	ptr=smm_malloc(1);	ptr!=NULL	Alloca 1 B.
LIB.AFOK.02	rc=smm_chk_ptr(ptr,&blk);	rc==0 e blk.start==ptr e blk.last==ptr	Verifica allocazione.
LIB.AFOK.03	rc=smm_free(ptr);	rc==0	Libera blocco.
LIB.AFOK.04	rc=smm_chk_ptr(ptr,&blk);	rc<0	Verifica disallocazione.
LIB.AFOK.05	ptr=smm_malloc(1024);	ptr!=NULL	Alloca 1 KB.
LIB.AFOK.06	rc=smm_chk_ptr(ptr,&blk);	rc==0 e blk.start==ptr e blk.last==ptr+0x000003FF	Verifica allocazione.
LIB.AFOK.07	rc=smm_free(ptr);	rc==0	Libera blocco.
LIB.AFOK.08	rc=smm_chk_ptr(ptr,&blk);	rc<0	Verifica disallocazione.
LIB.AFOK.09	ptr=smm_malloc(1048576);	ptr!=NULL	Alloca 1 MB.
LIB.AFOK.10	rc=smm_chk_ptr(ptr,&blk);	rc==0 e blk.start==ptr e blk.last==ptr+0x000FFFFF	Verifica allocazione.
LIB.AFOK.11	rc=smm_free(ptr);	rc==0	Libera blocco.
LIB.AFOK.12	rc=smm_chk_ptr(ptr,&blk);	rc<0	Verifica disallocazione.
LIB.AFOK.13	ptr=smm_malloc(16777216);	ptr!=NULL	Alloca 16 MB.
LIB.AFOK.14	rc=smm_chk_ptr(ptr,&blk);	rc==0 e blk.start==ptr e blk.last==ptr+ x00FFFFFF	Verifica allocazione.
LIB.AFOK.15	rc=smm_free(ptr);	rc==0	Libera blocco.
LIB.AFOK.16	rc=smm_chk_ptr(ptr,&blk);	rc<0	Verifica disallocazione.
LIB.AFER	***Test case che verificano le funzioni smm_malloc(), smm_free(), smm_chk_ptr() e che devono generare errore***		
LIB.AFER.01	ptr=smm_malloc(0);	ptr==NULL	Alloca 0 B.
LIB.AFER.02	ptr=smm_malloc(16777217);	ptr==NULL	Alloca 16 MB + 1B.
LIB.AFER.03	rc=smm_free(NULL);	rc<0	Libera un puntatore nullo.
LIB.AFER.04	rc=smm_free(0x12345678);	rc<0	Libera un puntatore non valido.
LIB.AFER.05	ptr1=smm_malloc(8388608);	ptr1!=NULL	Alloca 8 MB.
LIB.AFER.06	ptr2=smm_malloc(8388609);	ptr2==NULL	Alloca 8 MB + 1 B.
LIB.AFER.07	rc=smm_free(ptr1);	rc==0	Libera blocco da 8 MB.
LIB.AFER.08	rc=smm_free(ptr1);	rc<0	Libera nuovamente blocco da 8 MB.
LIB.AFER.09	rc=smm_free(ptr2);	rc<0	Libera puntatore non valido.
LIB.AFER.10	ptr=smm_malloc(4194304);	ptr!=NULL	Alloca 4 MB.
LIB.AFER.11	rc=smm_chk_ptr(ptr,&blk);	rc==0 e blk.start==ptr e blk.last==ptr+0x003FFFFF	Verifica puntatore iniziale al blocco da 4 MB.
LIB.AFER.12	rc=smm_chk_ptr(ptr+0x00000001,&blk);	rc==0 e blk.start==ptr e blk.last==ptr+0x003FFFFF	Verifica puntatore con offset 1 B all'interno del blocco da 4 MB.
LIB.AFER.13	rc=smm_chk_ptr(ptr+0x00200000,&blk);	rc==0 e blk.start==ptr e blk.last==ptr+0x003FFFFF	Verifica puntatore con offset 2 MB all'interno del blocco da 4 MB.
LIB.AFER.14	rc=smm_chk_ptr(ptr+0x003FFFFE,&blk);	rc==0 e blk.start==ptr e blk.last==ptr+0x003FFFFF	Verifica puntatore con offset 4 MB - 2 B all'interno del blocco da 4 MB.
LIB.AFER.15	rc=smm_chk_ptr(ptr+0x003FFFFF,&blk);	rc==0 e blk.start==ptr e blk.last==ptr+0x003FFFFF	Verifica puntatore finale (=4 MB - 1 B) del blocco da 4 MB.
LIB.AFER.16	rc=smm_chk_ptr(ptr+0x00400000,&blk);	rc<0	Verifica puntatore con offset non valido di 4 MB rispetto al blocco da 4 MB.
LIB.AFER.17	rc=smm_chk_ptr(ptr+0x00800000,&blk);	rc<0	Verifica puntatore con offset non valido di 8 MB rispetto al blocco da 4 MB.
LIB.AFER.18	rc=smm_chk_ptr(ptr+0x01000000,&blk);	rc<0	Verifica puntatore con offset non valido di 16 MB rispetto al blocco da 4 MB.
LIB.AFER.19	rc=smm_chk_ptr(0x12345678,&blk);	rc<0	Verifica puntatore non valido.
LIB.AFER.20	rc=smm_chk_ptr(NULL,&blk);	rc<0	Verifica puntatore nullo.
LIB.AFER.21	rc=smm_free(ptr);	rc==0	Libera blocco da 4 MB.
LIB.CPT	***Test case che verificano la procedura di compattamento di smmMem***		
LIB.CPT.01	ptr1=smm_malloc(4194304);	ptr1!=NULL	Alloca primo blocco 4 MB.

LIB - libsmm.a

Tag	Sequenza di comandi	Risultato atteso	Descrizione
LIB.CPT.02	ptr2=smm_malloc(4194304);	ptr2!=NULL	Alloca secondo blocco 4 MB.
LIB.CPT.03	ptr3=smm_malloc(4194304);	ptr3!=NULL	Alloca terzo blocco 4 MB.
LIB.CPT.04	ptr4=smm_malloc(4194304);	ptr4!=NULL	Alloca quarto blocco 4 MB.
LIB.CPT.05	rc=smm_free(ptr1);	rc==0	Libera primo blocco da 4 MB.
LIB.CPT.06	ptr5=smm_malloc(2097152);	ptr5!=NULL	Alloca 2 MB.
LIB.CPT.07	rc=smm_free(ptr3);	rc==0	Libera terzo blocco da 4 MB.
LIB.CPT.08	ptr6=smm_malloc(5242880);	ptr6!=NULL	Alloca 5 MB (deve essere stato effettuato compattamento).
LIB.CPT.09	rc=smm_free(ptr2);	rc==0	Libera secondo blocco da 4 MB.
LIB.CPT.10	rc=smm_free(ptr4);	rc==0	Libera quarto blocco da 4 MB.
LIB.CPT.11	rc=smm_free(ptr5);	rc==0	Libera blocco da 2 MB.
LIB.CPT.12	rc=smm_free(ptr6);	rc==0	Libera blocco da 5 MB.
LIB.PG	***Test case che verificano le funzioni smm_put(), smm_get() e che devono andare a buon fine*** Nota: si assume che siano definiti i buffer: char bufw[16777216], bufr[16777216]; e che bufw sia inizializzato con la sequenza di valori da 0 a 255 (0x00, 0x01, 0x02, ..., 0xFE, 0xFF) ripetuta per tutta la lunghezza del buffer:		
LIB.PG.01	ptr=smm_malloc(16777216);	ptr!=NULL	Alloca 16 MB.
LIB.PG.02	rc=smm_put(bufw,ptr,1);	rc==0	Scrivi 1 B.
LIB.PG.03	rc=smm_get(ptr,bufr,1);	rc==0	Leggi 1 B.
LIB.PG.04	rc=memcmp(bufw,bufr,1);	rc==0	Confronta buffer scritto con buffer letto.
LIB.PG.05	rc=smm_put(bufw,ptr,1024);	rc==0	Scrivi 1 KB.
LIB.PG.06	rc=smm_get(ptr,bufr,1024);	rc==0	Leggi 1 KB.
LIB.PG.07	rc=memcmp(bufw,bufr,1024);	rc==0	Confronta buffer scritto con buffer letto.
LIB.PG.08	rc=smm_put(bufw,ptr,1048576);	rc==0	Scrivi 1 MB.
LIB.PG.09	rc=smm_get(ptr,bufr,1048576);	rc==0	Leggi 1 MB.
LIB.PG.10	rc=memcmp(bufw,bufr,1048576);	rc==0	Confronta buffer scritto con buffer letto.
LIB.PG.11	rc=smm_put(bufw,ptr,16777216);	rc==0	Scrivi 16 MB.
LIB.PG.12	rc=smm_get(ptr,bufr,16777216);	rc==0	Leggi 16 MB.
LIB.PG.13	rc=memcmp(bufw,bufr,16777216);	rc==0	Confronta buffer scritto con buffer letto.
LIB.PG.14	rc=smm_put(bufw,ptr+0x00800080,1024);	rc==0	Scrivi 1 KB a partire da puntatore con offset 0x00800080 (8 MB+128 B) all'interno del blocco.
LIB.PG.15	rc=smm_get(ptr+0x00800080,bufr,1024);	rc==0	Leggi 1 KB a partire da puntatore con offset 0x00800080 (8 MB+128 B) all'interno del blocco.
LIB.PG.16	rc=memcmp(bufw,bufr,1024);	rc==0	Confronta buffer scritto con buffer letto.
LIB.PG.17	rc=smm_put(bufw,ptr,0);	rc<0	Scrivi 0 B.
LIB.PG.18	rc=smm_get(ptr,bufr,0);	rc<0	Leggi 0 B.
LIB.PG.19	rc=smm_put(bufw,ptr,16777217);	rc<0	Scrivi 16 MB + 1B.
LIB.PG.20	rc=smm_get(ptr,bufr,16777217);	rc<0	Leggi 16 MB + 1B.
LIB.PG.21	rc=smm_put(bufw,ptr+0x00FFFFF1,16);	rc<0	Scrivi 16 B a partire da puntatore valido interno al blocco, ma che lascia spazio solo per 15 B prima della fine del blocco.
LIB.PG.22	rc=smm_get(ptr+0x00FFFFF1,bufr,16);	rc<0	Leggi 16 B a partire da puntatore valido interno al blocco, ma che lascia spazio solo per 15 B prima della fine del blocco.
LIB.PG.23	rc=smm_put(bufw,ptr+0x01000000,1);	rc<0	Scrivi 1 B a partire da puntatore con offset non valido.
LIB.PG.24	rc=smm_get(ptr+0x01000000,bufr,1);	rc<0	Leggi 1 B a partire da puntatore con offset non valido.
LIB.PG.25	rc=smm_free(ptr);	rc==0	Libera blocco da 16 MB.
LIB.PG.26	rc=smm_put(bufw,0x12345678,1);	rc<0	Scrivi 1 B a partire da puntatore non valido.

LIB - libsmm.a

Tag	Sequenza di comandi	Risultato atteso	Descrizione
LIB.PG.27	rc=smm_get(0x12345678,bufr,1);	rc<0	Leggi 1 B a partire da puntatore non valido.
LIB.PG.28	rc=smm_put(bufw,NULL,1);	rc<0	Scrivi 1 B a partire da puntatore nullo.
LIB.PG.29	rc=smm_get(NULL,bufr,1);	rc<0	Leggi 1 B a partire da puntatore nullo.
LIB.CAL	***Test case che verificano la funzione smm_calloc()*** Nota: si assume che siano definiti i buffer: char bufz[16777216], bufr[16777216]; e che bufz sia inizializzato tutto a zero (0x00).		
LIB.CAL.01	ptr1=smm_calloc(1,1);	ptr1!=NULL	Alloca e azzera blocco da 1x1B=1B.
LIB.CAL.02	rc=smm_get(ptr1,bufr,1);	rc==0	Leggi 1B.
LIB.CAL.03	rc=memcmp(bufz,bufr,1);	rc==0	Verifica che il blocco sia inizializzato a zero.
LIB.CAL.04	ptr2=smm_calloc(1,4194304);	ptr2!=NULL	Alloca e azzera blocco da 1 elemento da 4MB=4MB.
LIB.CAL.05	rc=smm_get(ptr2,bufr,4194304);	rc==0	Leggi 4MB.
LIB.CAL.06	rc=memcmp(bufz,bufr,4194304);	rc==0	Verifica che il blocco sia inizializzato a zero.
LIB.CAL.07	ptr3=smm_calloc(4194304,1);	ptr3!=NULL	Alloca e azzera blocco da 4194304 elementi da 1B=4MB.
LIB.CAL.08	rc=smm_get(ptr3,bufr, 4194304);	rc==0	Leggi 4MB.
LIB.CAL.09	rc=memcmp(bufz,bufr,4194304);	rc==0	Verifica che il blocco sia inizializzato a zero.
LIB.CAL.10	rc=smm_free(ptr1);	rc==0	Libera blocco da 1B.
LIB.CAL.11	rc=smm_free(ptr2);	rc==0	Libera primo blocco da 4MB.
LIB.CAL.12	rc=smm_free(ptr3);	rc==0	Libera secondo blocco da 4MB.
LIB.CAL.13	ptr=smm_calloc(1,16777216);	ptr!=NULL	Alloca e azzera blocco da 1 elemento da 16MB=16MB.
LIB.CAL.14	rc=smm_get(ptr,bufr,16777216);	rc==0	Leggi 16MB.
LIB.CAL.15	rc=memcmp(bufz,bufr,16777216);	rc==0	Verifica che il blocco sia inizializzato a zero.
LIB.CAL.16	rc=smm_free(ptr);	rc==0	Libera blocco da 16MB.
LIB.CAL.17	ptr=smm_calloc(4096,4096);	ptr!=NULL	Alloca e azzera blocco da 4096 elementi da 4KB=16MB.
LIB.CAL.18	rc=smm_get(ptr,bufr,16777216);	rc==0	Leggi 16MB.
LIB.CAL.19	rc=memcmp(bufz,bufr,16777216);	rc==0	Verifica che il blocco sia inizializzato a zero.
LIB.CAL.20	rc=smm_free(ptr);	rc==0	Libera blocco da 16MB.
LIB.CAL.21	ptr=smm_calloc(16777216,1);	ptr!=NULL	Alloca e azzera blocco da 16777216 elementi da 1B=16MB.
LIB.CAL.22	rc=smm_get(ptr,bufr,16777216);	rc==0	Leggi 16MB.
LIB.CAL.23	rc=memcmp(bufz,bufr,16777216);	rc==0	Verifica che il blocco sia inizializzato a zero.
LIB.CAL.24	rc=smm_free(ptr);	rc==0	Libera blocco da 16MB.
LIB.CAL.25	ptr=smm_calloc(1,0);	ptr==NULL	Alloca e azzera blocco da 1 elemento da 0B=0B.
LIB.CAL.26	ptr=smm_calloc(0,1);	ptr==NULL	Alloca e azzera blocco da 0 elementi da 1B=0B.
LIB.CAL.27	ptr=smm_calloc(1,16777217);	ptr==NULL	Alloca e azzera blocco da 1 elemento da 16MB+1B=16MB+1B.
LIB.CAL.28	ptr=smm_calloc(16777217,1);	ptr==NULL	Alloca e azzera blocco da 16777217 elementi da 1B=16MB+1B.
LIB.CAL.29	ptr=smm_calloc(4097,4096);	ptr==NULL	Alloca e azzera blocco da 4097 elementi da 4KB=16MB+4KB.
LIB.CAL.30	ptr=smm_calloc(4096,4097);	ptr==NULL	Alloca e azzera blocco da 4096 elementi da 4KB+1B=16MB+4KB.
LIB.REA	***Test case che verificano la funzione smm_realloc()*** Nota: si assume che siano definiti i buffer: char bufw1[4096], bufw2[4096], bufr[4096]; e che bufw1 sia inizializzato con la sequenza di valori da 0 a 127 (0x00, 0x01, ... , 0x7F) ripetuta per tutta la lunghezza del buffer e che bufw2 sia inizializzato con la sequenza di valori da 128 a 255 (0x80, 0x81, ... , 0xFF) ripetuta per tutta la lunghezza del buffer		
LIB.REA.01	ptr1=smm_malloc(4096);	ptr1!=NULL	Alloca primo blocco da 4KB.
LIB.REA.02	rc=smm_put(bufw1,ptr1,4096);	rc==0	Scrivi 4KB utilizzando buf1.
LIB.REA.03	ptr2=smm_malloc(4096);	ptr2!=NULL	Alloca secondo blocco da 4KB.

LIB - libsmm.a

Tag	Sequenza di comandi	Risultato atteso	Descrizione
LIB.REA.04	rc=smm_put(bufw2,ptr2,4096);	rc==0	Scrivi 4KB utilizzando buf2.
LIB.REA.05	ptr3=smm_realloc(ptr1,2048);	ptr3!=NULL	Restringi il primo blocco a 2KB.
LIB.REA.06	rc=smm_get(ptr3,bufr,2048);	rc==0	Leggi 2KB dal blocco reallocato.
LIB.REA.07	rc=memcmp(bufw1,bufr,2048);	rc==0	Verifica che il contenuto del blocco riallocato sia inalterato.
LIB.REA.08	rc=smm_get(ptr2,bufr,4096);	rc==0	Leggi 4KB dal blocco non riallocato.
LIB.REA.09	rc=memcmp(bufw2,bufr,4096);	rc==0	Verifica che il contenuto del blocco non riallocato sia inalterato.
LIB.REA.10	rc=smm_chk_ptr(ptr3,&blk);	rc==0 e blk.start==ptr3 e blk.last==ptr3+0x000007FF	Verifica il nuovo puntatore al blocco riallocato.
LIB.REA.11	rc=smm_chk_ptr(ptr1,&blk);	se ptr3==ptr1, rc==0 altrimenti se ptr3!=ptr1, rc<0	Verifica il vecchio puntatore al blocco riallocato.
LIB.REA.12	ptr4=smm_realloc(ptr3,8192);	ptr4!=NULL	Espandi il primo blocco a 8KB.
LIB.REA.13	rc=smm_get(ptr4,bufr,2048);	rc==0	Leggi 2KB dal primo blocco.
LIB.REA.14	rc=memcmp(bufw1,bufr,2048);	rc==0	Verifica che il contenuto dei primi 2KB del blocco riallocato sia inalterato.
LIB.REA.15	rc=smm_get(ptr2,bufr,4096);	rc==0	Leggi 4KB dal secondo blocco.
LIB.REA.16	rc=memcmp(bufw2,bufr,4096);	rc==0	Verifica che il contenuto del blocco non riallocato sia inalterato.
LIB.REA.17	rc=smm_chk_ptr(ptr4,&blk);	rc==0 e blk.start==ptr4 e blk.last==ptr4+0x00001FFF	Verifica il nuovo puntatore al blocco riallocato.
LIB.REA.18	rc=smm_chk_ptr(ptr3,&blk);	se ptr4==ptr3, rc==0 altrimenti se ptr4!=ptr3, rc<0	Verifica il vecchio puntatore al blocco riallocato.
LIB.REA.19	ptr5=smm_realloc(ptr4,0);	ptr5==NULL	Restringi il primo blocco a 0B.
LIB.REA.20	ptr5=smm_realloc(ptr4,16777217);	ptr5==NULL	Espandi il primo blocco a 16MB + 1B.
LIB.REA.21	ptr5=smm_realloc(ptr4+0x01000000,4096);	ptr5==NULL	Rialloca un puntatore non valido.
LIB.REA.22	rc=smm_free(ptr2);	rc==0	Libera il secondo blocco.
LIB.REA.23	rc=smm_free(ptr4);	rc==0	Libera il primo blocco.
LIB.REA.24	ptr=smm_realloc(0x12345678,1);	ptr==NULL	Rialloca a 1 byte un puntatore non valido.
LIB.REA.25	ptr=smm_realloc(NULL,1);	ptr==NULL	Rialloca a 1 byte un puntatore nullo.

SMM - Prestazioni e protezione della memoria

Tag	Sequenza di comandi	Note

SMM.PER	***Misura delle prestazioni***
	Nota: le funzioni *perf_smm_<func>()* per la misura dei tempi di esecuzione delle funzioni di libreria *smm_<func>()* devono essere implementate secondo lo seguente struttura di riferimento (valida nel caso della smm_malloc(); per smm_put() e smm_get() il codice è simile):
	
	size_t size=1; /* dimensione del blocco da allocare, a partite da 1B fino a 16MB */
	int i,j;
	int N=24; /* numero di misure da registrare (raddoppiando ogni volta la dimensione del blocco da 1B a 16MB=2^24) */
	int NITER=1000; /* numero di iterazioni per ogni allocazione in modo da calcolarne la media dei tempi; la costante da inserire va determinata in modo che l'intervallo (te-ts) non sia nullo o troppo piccolo */
	long t[N], ts, te; /* misure di tempi (potrebbero essere nsec, usec, ... a seconda della funzione gettime utilizzata) */
	for (i=0;i<=N;i++) {
	ts=gettime(); /* gettime() e' una funzione scritta ad hoc ed deve basarsi su quelle di sistema, ad es. gettimeofday() */
	for (j=0;j<NITER;j++) {
	ptr=smm_malloc(size);
	free(ptr);
	}
	te=gettime();
	t[k]=(te-ts) / N;
	size=size*2;
	}	

SMM.PER.01	perf_smm_malloc();	Misura il tempo di esecuzione t_k di smm_malloc(size) con size=2^k, k=[0,1,2 ... 23,24] (da 1B a 16MB).
SMM.PER.02	perf_smm_put();	Misura il tempo di esecuzione t_k di smm_put(buf,ptr,size) con size=2^k, k=[0,1,2 ... 23,24] (da 1B a 16MB).
SMM.PER.03	perf_smm_get();	Misura il tempo di esecuzione t_k di smm_get(ptr,buf,size) con size=2^k, k=[0,1,2 ... 23,24] (da 1B a 16MB).

SMM.PRO	***Verifica del meccanismo di protezione della memoria in condizione di multiprocesso***
	Nota: la funzione *prot_smm_<func>()* deve essere implementata secondo la seguente struttura di riferimento(valida nel caso della smm_free(); per le altre il codice è simile):
	
	void *ptr;
	pid_t pid; /* PID del processo figlio */
	int rc;
	
	ptr=smm_malloc(1024);
	pid=fork();
	if (pid==0) { /* figlio */
	rc=smm_free(ptr); /* il figlio ha un PID diverso dal padre, quindi non deve poter accedere al blocco allocato dal padre */
	printf("rc=%d\n",rc);
	}	

SMM.PRO.01	prot_smm_free()	rc<0. Verifica la protezione della memoria allocata da un altro processo rispetto alla funzione smm_free.
SMM.PRO.02	prot_smm_realloc()	rc<0. Verifica la protezione della memoria allocata da un altro processo rispetto alla funzione smm_realloc.
SMM.PRO.03	prot_smm_put()	rc<0. Verifica la protezione della memoria allocata da un altro processo rispetto alla funzione smm_put.
SMM.PRO.04	prot_smm_get()	rc<0. Verifica la protezione della memoria allocata da un altro processo rispetto alla funzione smm_get.
SMM.PRO.05	prot_smm_chk_ptr()	rc<0. Verifica la protezione della memoria allocata da un altro processo rispetto alla funzione smm_chk_ptr.