Virtual Memory Manager (vmem): specifiche per il progetto d'esame valide per l'appello di Gennaio Ver. 2.1

Sistemi Operativi, Modulo 2
Anno Accademico 2009-2010

Alessandro Mei

Luigi V. Mancini - Angelo Spognardi

Obiettivo

- Questo modulo descrive le **specifiche del progetto** da svolgere per l'esame di Sistemi Operativi, modulo 2 - AA 2009-2010.
-

Il progetto richiede lo sviluppo di un semplice gestore di memoria virtuale, denominato `vmem` (Virtual 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 Giu-Lug 2010 si richiede il soddisfacimento delle sole **specifiche di base**, mentre per quelli successivi si applicano anche le **specifiche estese**.

Specifiche di base

valide per il primo appello

(Giugno-Luglio)

Specifiche di base - I

Il software package Virtual Memory Manager `vmem`:

- dovrà emulare il comportamento di un **gestore di memoria virtuale:**
  - **a paginamento semplice** (*paging only*)
  - **con dimensione della frame table fissa e massima memoria allocabile fissa**
  - **con gestione dei page fault tramite First-In-First-Out e algoritmo dell'orologio**

  (vedasi W. Stallings, "Operating Systems", cap. 8)
- dovrà essere implementato come un'**applicazione di tipo server** di nome **`vmemd`,** consistente in un eseguibile Linux (cioè NON come modulo kernel) **che gestisce uno spazio di memoria contiguo allocato nello heap del processo server**, denominato **`frameMem`**, di dimensione fissa pari a **1 MB (2^20)**
- dovrà consentire alle applicazioni client di richiedere l'accesso a **locazioni di memoria indicati con indirizzi virtuali** organizzate in pagine che possono essere già in memoria oppure residenti su **un file di swap** (page fault) di 4 MB (2^22)
- dovrà fornire un'**API per l'allocazione di pagine di memoria all'interno di `frameMem` e relativo accesso agli indirizzi virtuali** da parte delle applicazioni client mediante una **libreria di nome `libvmem.a`**

Specifiche di base – II

- dovrà implementare un meccanismo basato sullo **scambio di messaggi via FIFO** tra le applicazioni client e il programma server, che supporti le richieste di allocazione di pagine e accesso/modifica a/di indirizzi di memoria in **`frameMem`**
- dovrà implementare un **meccanismo di protezione** della memoria gestito da **`vmemd`** per impedire che un qualsiasi processo possa accedere a pagine di memoria di altro processo
- dovrà fornire una **programma interattivo,** di nome **`vmemMon`, per l'avvio/terminazione di `vmemd` e per il monitoraggio a *run-time* di `frameMem`** e l'accesso alle statistiche sui page fault.

Specifiche implementative

- il server **`vmemd`** dovrà **memorizzare, organizzare e mantenere integre** le informazioni relative allo stato di allocazione di **`frameMem`** mediante opportune strutture di dati
- lo scambio di messaggi tra applicazioni e server **`vmemd`** deve avvenire tramite **una coppia di FIFO** per la comunicazione full-duplex:
  `/tmp/FIFOtovmemd` : richieste da applicazione a **`vmemd`**
  `/tmp/FIFOfromvmemd` : risposte da **`vmemd`** ad applicazione
- il server **`vmemd`** alla partenza deve creare l'area di memoria **`frameMem`**, deve creare e inizializzare le strutture dati necessarie a mantenere il controllo di **`frameMem`**, e infine attendere le richieste da parte delle applicazioni
- il server **`vmemd,`** 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

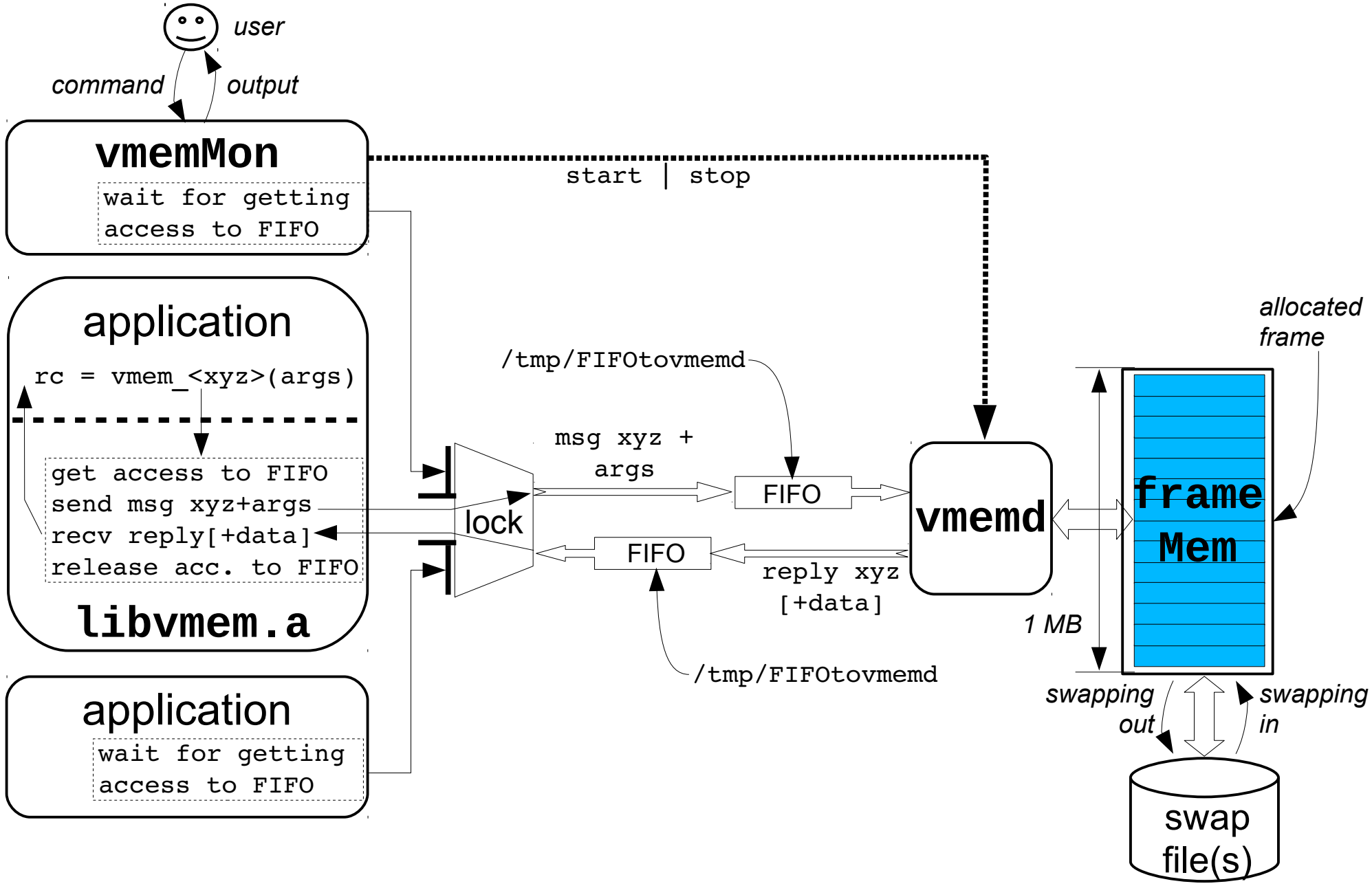
Specifiche implementative

- la libreria **`libvmem.a`** dovrà implementare all'interno delle sue funzioni un **meccanismo di lock esclusivo** delle FIFO per garantire l'atomicità della procedura *<invio richiesta-ricezione risposta>*
- tutti i puntatori gestiti dalla libreria **`libvmem.a`** alle locazioni di memoria in **`frameMem`** (cioè quelli che compaiono come argomenti o valori di ritorno delle varie funzioni) devono essere indirizzi virtuali che non coincidono con i corrispondenti indirizzi utilizzati da **`vmemd`**; ciò è richiesto per permettere la rilocazione delle pagine di memoria virtuale a seguito delle operazioni di *swapping out* + *swapping in*
- la terminazione del server **`vmemd`** deve avvenire a seguito dell'**invio del segnale `SIGTERM`**, che attiva il cleanup delle strutture dati e procedure in corso e poi l'exit

Specifiche implementative

- le applicazioni client come prima operazione devono indicare al server la memoria che utilizzeranno
- la memoria di ogni applicazione dovrà essere organizzata in pagine di dimensioni pari a **16-kbyte** (2^14)
- le richieste di indirizzi nella memoria virtuale possono causare un page fault che dovrà essere gestito tramite l'algoritmo dell'orologio e l'algoritmo First-In-First-Out
- quando l'applicazione notifica al server l'intenzione di terminare, esso provvederà a distruggere le strutture dati non più necessarie e a rilasciare le risorse allocate per tale applicazione
- il server dovrà essere in grado di valutare le proprie performance tenendo traccia del numero di page-fault che si verificano durante il suo periodo di attività

Schema architetturale



Funzioni da implementare in `libvmem.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 e di accesso

- `void *vmem_create(size_t size);`
  – alloca una dimensione di memoria virtuale di dimensione pari a `size` byte e ritorna un puntatore all'inizio della prima pagina. `size` deve essere > 0. Ritorna `NULL` in caso di errore.
- `int vmem_get(void *ptr, void *buf, size_t size);`
  – preleva a partire dal puntatore `ptr` appartenente a un blocco di memoria virtuale 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).
- `int vmem_put(void *buf, void *ptr, size_t size);`
  – inserisce a partire dal puntatore `ptr` appartenente a un blocco di memoria virtuale 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 vmem_close();`
  – rilascia la memoria virtuale richiesta e consente al server di liberare le risorse. Ritorna un codice di successo (=0) o errore (<0).

Funzioni da implementare in `libvmem.a` – II

Funzioni di verifica dei puntatori virtuali alla memoria

- `int vmem_chk_ptr(void *ptr, struct vmem_blk *blk);`

```
struct vmem_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 */
  int nfetch;  /* number of fetching from swap file */
  pid_t pid;   /* PID of the process owner of the block */
};
```

– rileva le informazioni relative al blocco a cui appartiene `ptr`, riempiendo opportunamente i campi della struttura pre-allocata dal chiamante. 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 `vmemMon` - I

- Il programma **`vmemMon`** 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 `vmemMon` - I

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

- **`start [-c | -f]`** – avvia il server `vmemd` e stampa messaggio di successo/errore.
- **`stop`** – termina il server `vmemd` e stampa messaggio di successo/errore.
- **`exit`** – esce da `vmemMon`.
- **`check`** ***`ptr`*** – stampa i puntatori al primo e ultimo byte del blocco che contiene `ptr` o messaggio di errore se `ptr` non e' valido. Stampa, inoltre, le informazioni statisiche sullo stato del relativo blocco (stato di *swapping in/out,* numero di volte che è stato caricato dal file di *swap,* processo proprietario).
- **`get`** ***`ptr`*** – richiede il contenuto dell'indirizzo virtuale puntato da `ptr` o messaggio di errore se `ptr` non e' valido.

Comandi da implementare in `vmemMon` - II

- **`lap [PID]`** – (*list all pages*) stampa i valori dei puntatori al primo e ultimo byte e della dimensione in byte (in decimale) di `frameMem` e di tutte le pagine allocate dalle varie applicazioni con relativo indirizzo virtuale e PID.
  Se viene inserito un numero di PID valido come argomento opzionale, stampa solo le informazioni sulle pagine di memoria allocate dal processo identificato da PID.
  Se il PID non e' valido, stampa un messaggio di errore. Indica anche se le pagine sono presenti attualmente in memoria (*swapped in* o *swapped out*).
- **`lf`** – (*list frames*) stampa lo stato dei frame in memoria: liberi o allocati, in questo caso stampa anche il relativo indirizzo virtuale e PID di appartenenza. Stampa anche la situazione delle pagine (situazione algoritmo clock O situazione First-In-First-Out)

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 **`vmem`** deve essere gestita mediante **`Makefile`**
- L'installazione degli eseguibili **`vmemd, vmemMon`** e libreria **`libvmem.a`** deve avvenire nella directory **`$HOME/vmem`** 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 **`vmemd`** e **`vmemMon`** con il dettaglio delle principali scelte implementative (strutture di dati, algoritmi, flussi di controllo a livello macro, ...)
  - lla struttura delle funzioni della libreria **`libvmem.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 **`vmem.tgz`**, contenente i sorgenti C, gli include file, il **`Makefile`**, la documentazione ed eventuali file per il test
- il file **`vmem.tgz`** deve contenere una sola directory (con pathname relativo) di nome **`vmem`**
- la directory **`vmem`** 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
- La relazione dovrà includere dei test su quanti client sono servibili dal server con un numero di page fault sufficientemente basso (I client e le specifiche saranno forniti dai docenti in seguito)

Specifiche aggiungtive

valide per l'appello di Gennaio 2011

Funzioni da implementare in `libvmem.a` – III

Funzioni per il ridimensionamento della memoria virtuale

- `int vmem_resize(void *ptr, size_t size);`

  - chiede al gestore di ridimensionare il proprio spazio di memoria virtuale. Il nuovo spazio dovrà avere dimensione pari a `size` byte e ritorna un puntatore all'inizio della prima pagina. `size` deve essere > 0. Ritorna `0` in caso di successo, < 0 in caso di errore e di memoria virtuale non più disponibile.
  - nel caso in cui la dimensione richiesta sia minore di quella di partenza, il gestore deve rimuovere le pagine con indirizzi che superano la nuova `size`.
  - nel caso in cui la dimensione richiesta sia maggiore di quella di partenza, il gestore deve aumentare lo spazio a disposizione del processo fino alla nuova size, garantendo che il contenuto della memoria prima della resize resti invariato nelle stesse locazioni di memoria virtuale.
  - nel caso in cui size sia uguale a size usato nella create, la funzione ritorna un successo senza alterare nulla.