Enunt

Sa se implementeze sub forma unei biblioteci un "swapper & demand pager". Memoria virtuala, reprezentata de o zona de memorie din spatiul de adresa al procesului, va avea`virt_pages` pagini. Memoria fizica, simulata de un fisier va avea `phys_pages` pagini. Pentru swapping se va folosi alt fisier. Fisierul de swap va avea `virt_pages` pagini.

Biblioteca trebuie sa creeze mapari intre paginile virtuale si paginile fizice si va oferi urmatoarele functii, prezentate in fisierul header linux tema5.h

```
void *vinit (size_t virt_pages, size_t phys_pages);
int get_ram_fd (void);
int get_swap_fd (void);
void ram_sync (void);
int vend (void);
```

sau windows tema5.h

```
__declspec (dllexport) LPVOID VirtualInit (DWORD virtualPages, DWORD
physicalPages);
__declspec (dllexport) HANDLE GetRamHandle (void);
__declspec (dllexport) HANDLE GetSwapHandle (void);
__declspec (dllexport) void SynchronizeRam (void);
__declspec (dllexport) BOOL VirtualEnd (void);
```

Functia `vinit` va intoarce un pointer catre o zona de memorie din spatiul de adresa al procesului; zona obtinuta reprezinta memoria virtuala, pointer-ul intors fiind, astfel, adresa de inceput a memoriei virtuale.
In cadrul functiei vinit va avea loc, de asemenea, crearea fisierelor ce reprezinta ram-ul si swap-ul si initializarea handler-elor de semnale, respectiv vectorilor de exceptie.

Functia `vend` realizeaza cleanup: demaparea zonei de memorie cumulata cu inchiderea fisierelor ce reprezinta ram-ul si swap-ul si resetarea handler-ului de semnal/vectorului de exceptii. `vend`() returnează 0 în caz de succes și -1 în cazul unei erori. Singura eroare testată este dacă se cheamă `vend`() când librăria de memorie virtuală pe care o scrieți voi nu a fost inițializată în prealabil în mod corect (nu s-a chemat `vinit`() sau cel mai recent `vinit`() a returnat `NULL`).

Functiile `get_ram_fd`, respectiv `get_swap_fd` intorc descriptorii fisierelor asociate memoriei fizice si swap-ului.

Functia `ram_sync` realizeaza sincronizarea intre datele reprezentate in memoria virtuala si fisierul ce simuleaza memorie RAM. Functia are rolul de efectuare a unui flush a datelor din memoria virtuala astfel incat acestea sa se gaseasca in fisierul ce reprezinta memoria RAM. Intrucat se genereaza un page fault doar la un primul acces la o pagina, urmatoarele accese vor modifica memoria virtuala, fara a se reflecta schimbarile in memoria RAM. De

aceea, atunci cand se doreste verificarea memoriei RAM va trebui facuta sincronizarea informatiilor din memoria virtuala.

Actualizarea inversa, de la RAM la memoria virtuala se realizeaza in momentul copierii unei pagini din swap in RAM, in cazul unui page-fault (pagina dorita se afla in swap).

Functiile pe Windows realizeaza aceleasi lucruri, doar ca sunt denumite diferit.

La crearea spatiului de adresa virtual, dupa apelarea functiei `vinit`, paginile nu trebuie sa fie prezente in memorie. Ele vor fi create doar atunci cand programul le acceseaza. Daca nu mai sunt pagini fizice libere, biblioteca trebuie sa evacueze o pagina fizica in swap. Paginile din swap trebuie aduse in memoria fizica, atunci cand ele sunt accesate.

Pentru o pagina care rezida in memori fizica va trebui sa faceti deosebirea intre o pagina care a fost modificata (dirty) si una care nu a fost modificata. Daca o pagina nu a fost modificata atunci, in momentul inlocuirii, ea nu va mai fi inlocuita in swap (nu se va face copiere). Pentru usurinta in testare, acest lucru ramane valabil si la paginile care sunt initial read-only.

Pentru un acces de modificare (write), testarea impune generarea a doua page fault-uri. Dupa primul page fault, pagina este marcata ca read-only, urmand sa se genereze un nou page fault si pagina sa fie marcata read-write.

Nu se impune nici o constrangere legata de algoritmul de evacuare a paginilor in swap. Puteti folosi aceeasi pagina, sau o pagina aleasa aleator, sau un algoritm cunoscut (FIFO, LRU, etc.)

Testare

Pentru simplificarea procesului de corectare a temelor, dar si pentru a reduce greselile temelor trimise, corectarea temelor se va face automat cu ajutorul unor teste publice (linux, windows).

In urma compilarii temei trebuie sa rezulte o biblioteca shared-object (linux) denumita **libvm.so** sau o biblioteca dll (windows) denumita **libvm.dll**. Functiile care sunt exportate de biblioteca se gasesc in fisierul header Linux (Tema5_lin.h) si Windows (Tema5_win.h). Acesta trebuie inclus in arhiva in care se trimite tema.

O tema care trece toate testele poate lua maxim nota 10. Pot exista penalizari in cazul in care apar warning-uri sau intarzieri, sau daca nu se respecta unele criterii cum sunt cele mentionate mai jos:

- -0.1 - Makefile necorespunzator
- -0.1 - README necorespunzator
- -0.4 - cod neingrijit (indentare inexistenta sau necorespunzatoare, surse necomentate)
- -0.4 - neeliberare resurse, neverificare conditii de eroare, folosire necorespunzatoare a functiilor din API

In cazul in care se trec numai anumite teste punctarea se va face propor tional cu numarul total de teste. Cazuri speciale sunt testele de tip check si clean (in numar de 11): acestea au o valoare de 0.5p **fiecare** Un test poate fi depunctat daca acesta reuseste sa fie valid ("Passed") dar implementarea este invalida.

Precizari Generale

Pentru mai multe informatii despre demand paging si swapping, consultati cursul 6 sau cititi capitolul de memorie virtuala din "Modern Operating Systems".

Exista predefinit in fisierul header o valoare maxima a numarului de pagini virtuale, respectiv fizice care pot fi transmise `vinit`. De asemenea, se impune conditia ca paginile virtuale sa fie in numar mai mare ca paginile fizice. Daca aceste conditii nu au loc, functia intoarce NULL. Pentru ca pe Windows, nu se pot crea (usor) pagini virtuale multiple de pagini ale sistemului (ci multiplu de 64 K) s-a definit variabila

```
static DWORD pageSize = 0x10000;
```

care va fi folosita pentru a specifica dimensiunea unei pagini.

Tema se bazeaza pe interceptarea page-fault-urilor in momentul unui acces invalid la o zona de memorie.

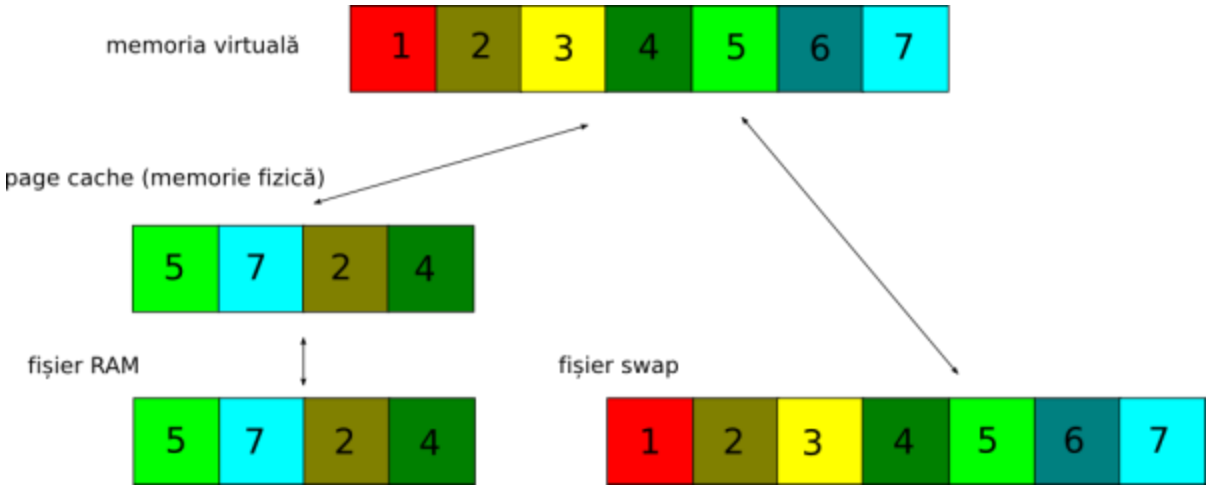
Nu este nevoie sa faceti sincronizarea intre memoria virtuala si fisierul ce simuleaza memoria fizica dupa fiecare scriere. Acest lucru se va realiza in momentul in care testul apeleaza functia `ram_sync` pusa la dispozitie de biblioteca.

Nu se impune nici o restrictie de nume asupra fisierelor ce reprezinta memoria RAM, respectiv swap-ul. Programul de test utilizeaza descriptorul, respectiv handle-ul asociat fisierelor prin intermediul functiilor `get_ram_fd` si `get_swap_fd`.

Imagine

Recomandarea de implementare este să mapați paginile virtuale direct peste fișierul de RAM folosind apelurile `mmap`, `MapViewOfFile`. Nu se va depuncta, însă, pentru altă implementare. În cazul mapării paginilor virtuale direct peste fișierul RAM este funcția `ram_sync` va fi implementată printr-un apel `msync`.

Pentru cazul în care se folosește maparea peste fișierul RAM situația rezultată poate fi exemplificată de imaginea de mai jos:



În imaginea de mai sus este prezentată situația în care fișierul RAM este populat cu pagini, iar fișierul swap conține toate paginile virtuale. În evoluția programului există alte situații: nu toate paginile din fișierul RAM sunt populate, nu s-au copiat pagini în fișierul swap etc.

Precizari Windows

Tema se va rezolva folosind doar functii Win32. Se pot folosi de asemenea si functiile de formatare `printf`, `scanf`, functiile de alocare de memorie `malloc`, `free` si functiile de manipulare a sirurilor de caractere (`strcat`, `strdup`, etc.)

Pentru partea de I/O si procese se vor folosi doar functii Win32. De exemplu, functiile `open`, `read`, `write`, `close` nu trebuie folosite, in locul acestor trebuind sa folositi `CreateFile`,`ReadFile`, `WriteFile`, `CloseHandle`.

Pentru gestiunea memoriei virtuale folositi functiile `VirtualAlloc/VirtualFree/VirtualProtect`.

Pentru interceptarea acceselor invalide la zone de memorie (general protection fault) folositi apelurile `AddVectoredExceptionHandler/RemoveVectoredExceptionHandler`.

Precizari Linux

Tema se va rezolva folosind doar functii POSIX. Se pot folosi de asemenea si functiile de formatare `printf`, `scanf`, functiile de alocare de memorie `malloc`, `free` si functiile de manipulare a sirurilor de caractere (`strcat`, `strdup`, etc.)

Pentru partea de I/O si procese se vor folosi doar functii POSIX. De exemplu, functiile `fopen`, `fread`, `fwrite`, `fclose` nu trebuie folosite, in locul acestor trebuind sa folositi `open`,`read`, `write`, `close`.

Pentru gestiunea memoriei virtuale folositi functiile `mmap/munmap/mprotect`.

Pentru interceptarea accesului invalid la o zona de memorie va trebui sa interceptati semnalul `SIGSEGV` folosind apeluri din familia `sigaction`.

Dimensiunea paginii de memorie virtuala o aflati cu ajutorul functiei `getpagesize`.