Laborator 1

Introducere

Sisteme de Operare

23 Februarie - 1 Martie 2017

Cine sunt?

- email
- experiență
- pasiuni relevante
- de ce SO?

Resurse

- Wiki: http://ocw.cs.pub.ro/so
  - NeedToKnow page: http://ocw.cs.pub.ro/so/meta/need-to-know
  - Folosiți feed-ul RSS
- Lista de discuții
  - so@cursuri.cs.pub.ro
  - Abonați-vă (detalii pe wiki)
- Catalog Google, calendar Google
- Mașini virtuale
- vmchecker (verificare teme)
- Documentație
- cs.curs.pub.ro (rol de portal)
- Pagină de Facebook

Despre laborator

- Subiecte principale
  - Procese
  - Thread-uri
  - Comunicare și sincronizare
  - Memorie
  - Sisteme de fișiere
  - I/O
- POSIX/Win32 API programming (C/C++)
- prezentare + joc interactiv
- Tutorial-like, task-based, learn by doing
- Karma Points ("pentru cei puternici")

Teme

- Tema 1 – hash-table
- Tema 2 – mini-shell
- Tema 3 – demand pager/swapper
- Tema 4 – thread scheduler
- Tema 5 – server de fișiere

- Intense
- Necesare: aprofundare API (laborator) și concepte (curs)
- Estimare de timp: 8-20 ore pe temă
- Teste publice
- Suport de testare la submit - feedback imediat

Notare laborator

- https://ocw.cs.pub.ro/courses/so/meta/notare/reguli-notare-ca-cb-cc

Karma Awards

- Cum se obțin Karma Points?
  - Participare la discuțiile din timpul cursului
  - Participare la discuțiile din timpul laboratorului
  - Răspunsuri pe lista de discuții
  - Editarea wiki-ului
  - Exercițiile bonus din timpul laboratorului
  - Teme elegante
    - Coding style consistent, comentarii punctuale, claritatea codului
    - Soluții simple și corecte
    - Modularitate, cursivitate

Desfășurare laborator

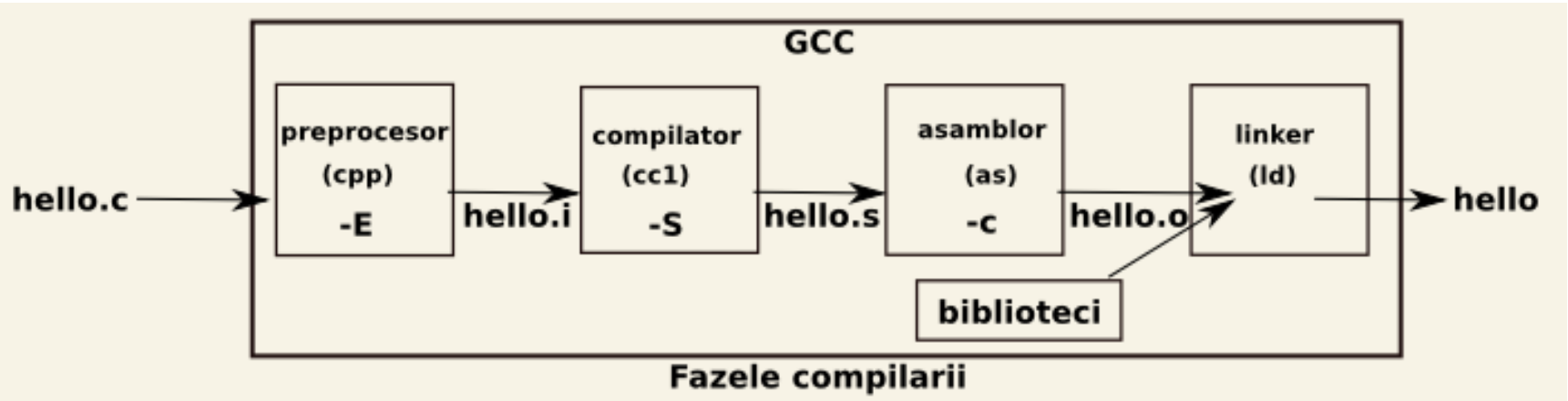
- Parcurgere laborator acasă - 40 de minute
- Prezentare teoretică + joc introductiv - 30 de minute
- Rezolvare exerciții - 80 de minute
  - Punctaj între 0 și 11
  - Bucuria rezolvării unui laborator de SO infinită :)

- Cărți
  - TLPI, The Linux Programming Interface, M. Kerrisk
  - WSP4, Windows System Programming 4th Edition, J. Hart
- Listă de discuții
  - http://cursuri.cs.pub.ro/cgi-bin/mailman/listinfo/so
- Canal IRC, rețea Freenode, #cs_so

Ce vom învăța în laborator

- Compilare, depanare, biblioteci
- Operații I/E simple
- Procese
- Semnale
- Gestiunea memoriei
- Debugging
- Memoria virtuală
- Fire de execuție (2)
- Operații de I/E avansate (2)
- Sisteme de fișiere

- Compilare
  - Traducerea unui program (limbaj sursă, limbaj țintă)
- Makefile
  - Automatizarea procesului de compilare
- Depanare
  - Detectarea erorilor din programe
- Biblioteci
  - Colecție de fișiere precompilate


GCC
hello.c
preprocesor
(cpp)
-E
hello.i
compilator
(cc1)
-S
hello.s
asamblor
(as)
-c
hello.o
linker
(ld)
hello
biblioteci
Fazele compilarii

GCC

- GNU Compiler Collection
- gcc hello.c
  - Compilare simplă, rezultă fișierul executabil a.out
- gcc hello.c -o hello
  - Compilare simplă cu specificarea numelui fișierului de ieșire
- gcc hello.c -c -o hello.o
  - Oprirea compilării după obținerea fișierului obiect
- gcc hello.o -o hello
  - Editarea de legături pentru fișierul obiect hello.o

- cl.exe - Microsoft Compiler
- cl hello.c
  - Compilare simplă, rezultă fișierul executabil hello.exe
- cl /Fehello_win.exe hello.c
  - Compilare simplă cu specificarea numelui executabilului
- cl /c hello.c
  - Obținerea fișierului obiect
- cl /Fehello.exe hello.obj
  - Editarea de legături pentru fișierul obiect
- cl /? - help

Fișiere make

- Automatizarea compilării
- Fișier Makefile
  - Reguli
  - Comenzi
  - Variabile
- Compilare 'deșteaptă'
- make vs. nmake

gdb

- Fișierele sunt compilate cu opțiunea -g
- Execuție
  - gdb ./a.out
- Comenzi utile
  - p - print
  - bt - backtrace
  - step, next
  - set args

Biblioteci

- Statice
  - Rezolvare simboluri în momentul editării de legături
  - Funcțiile utilizate sunt incluse în executabil
  - Dimensiune executabil mai mare, rulare mai rapidă
- Dinamice
  - Rezolvare simbolurilor se poate face
    - La încărcare (load-time)
    - La rulare (run-time) (dlopen and friends)
  - Executabil de dimensiune redusă

Lucrul cu biblioteci în Linux

- Crearea unei biblioteci statice (.a)
  - ar rc libxyz.a f1.o f2.o
- Crearea unei biblioteci partajate (.so)
  - gcc -fPIC -c f1.c
  - gcc -shared f1.o -o libxyz.so
- Legarea cu o bibliotecă
  - -lxyz
  - -Lpath
  - LD_LIBRARY_PATH

- Crearea unei biblioteci statice (.lib)
  - lib /out:<nume.lib> <lista fisiere obiect>
- Crearea unei biblioteci dinamice (.dll)
  - __declspec(dllimport), __declspec(dllexport)
  - link (/dll) sau cl /LD

Laborator 2

Operații I/O simple

Sisteme de Operare

28 februarie - 6 Martie 2019

Fișierul

- unitate logică de stocare
- abstractizează proprietățile fizice ale mediului de stocare
- colecție de date + nume asociat
- organizare ierarhică
  - `/home/student/lab/lab02/slides/lab02.tex`
  - `D:\so\lab02\1-cat\cat.c`

Tipuri de fișiere

- fișiere obișnuite
- directoare
- link-uri simbolice
- character device
- block device
- pipe-uri
- socketi UNIX

Operații pe fișier

- creare/deschidere
- citire
- scriere
- deplasare în cadrul fișierului
- trunchiere
- ștergere/închidere

- file descriptor vs. file handle
- Linux
  - **open**
    - mod de acces(`flags`): `O_RDONLY`, `O_WRONLY`, `O_RDWR`
    - acțiuni la creare(`flags`): `O_CREAT`, `O_EXCL`, `O_TRUNC`
    - `mode` - permisiuni (ex: 0644)
- Windows
  - **CreateFile**
  - nu „creează un fișier”, ci un handle către un fișier
  - `dwDesiredAccess` - `GENERIC_READ`, `GENERIC_WRITE`
  - `dwShareMode` - `FILE_SHARE_READ`, `FILE_SHARE_WRITE`
  - `dwCreationDisposition` - `CREATE_NEW`, `OPEN_EXISTING`, `TRUNCATE_EXISTING`

Închidere/Ștergere

Linux

- `close`
- `unlink`

Windows

- `CloseHandle`
- `DeleteFile`

- Linux
  - ssize_t read(int fd, void *buf, size_t count);
  - ssize_t write(int fd, const void *buf, size_t count);
    - întoarce numărul total de octeți citiți/scriși **efectiv**
- Windows

```
bRet = ReadFile(                    bRet = WriteFile(
    hFile,                              hFile,                  open file handle
    lpBuffer,                           lpBuffer,               start of data
    dwBytesToRead,                      dwBytesToWrite,         number of bytes
    &dwBytesRead,                       &dwBytesWritten,        return number
    NULL );                             NULL );                 no overlapped
```

Linux	Windows	
lseek	**SetFilePointer**	
`whence`	`dwMoveMethod`	poziția relativă de la care se face deplasare
SEEK_SET	FILE_BEGIN	față de începutul fișierului
SEEK_CUR	FILE_CURRENT	față de poziția curentă
SEEK_END	FILE_END	față de sfârșitul fișierului

- Cum putem determina dimensiunea unui fișier?

Redirectări (Linux)

- int dup(int oldfd)
- int dup2(int oldfd, int newfd)
  - `STDIN_FILENO`
  - `STDOUT_FILENO`
  - `STDERR_FILENO`

Utilitare (Linux)

- lsof(1) – listează informații despre fișierele deschise
- stat(1) – listează informații despre un fișier/sistem de fișiere
- strace(1) – system calls trace
- ltrace(1) – library calls trace

Laborator 3

Procese

Sisteme de Operare

7 - 13 Martie 2019

Ce este un proces?

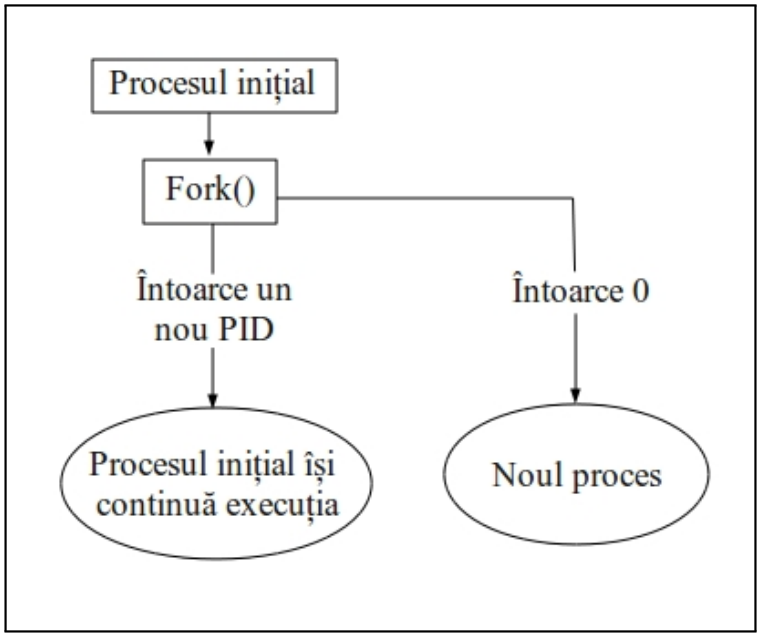
- program în execuție
- unitatea primitivă prin care sistemul de operare alocă resurse utilizatorilor
- caracteristici
  - spațiu de adrese
  - unul sau mai multe fire de execuție
- informațiile asociate procesului (Process Control Block)
  - tabela de fișiere deschise
  - handler-ele pentru semnale
  - directorul curent

Operații cu procese

- creare
- așteptarea terminării
- terminare
- duplicarea descriptorilor de resurse

- Linux - organizare ierarhică

- `fork` - **duplică** procesul curent
  - 0, în copil
  - pid $>$ 0, în părinte
  - -1, în caz de eroare
- `exec` - **înlocuiește** imaginea procesului



- Windows - organizare neierarhică
  - `CreateProcess` - îmbină cele două operații de pe Linux

Așteptarea terminării unui proces

- Linux
  - `waitpid`, `wait`
    - suspendă execuția procesului apelant până când procesul (procesele) specificat în argumente fie s-au terminat, fie au fost oprite (SIGSTOP)
  - `WIFEXITED`, `WEXITSTATUS` ...
    - obțin modul și codul de ieșire ale procesului, examinând status, întors de waitpid
- Windows
  - `WaitForSingleObject`, `WaitForMultipleObjects`
    - suspendă execuția procesului curent până când unul sau mai multe alte procese se termină
  - `GetExitCodeProcess`
    - determină codul de eroare cu care s-a terminat un anumit proces

Terminarea unui proces

- Linux
  - `exit`
    - încheie execuția procesului curent
    - toți descriptorii de fișier ai procesului sunt închiși
    - copiii procesului sunt "înfiați" de `init`
    - părintelui procesului îi e trimis un semnal `SIGCHLD`
    - va scrie bufferele streamurilor deschise și le va închide
- Windows
  - `ExitProcess`
    - încheie execuția procesului curent
  - `TerminateProcess`
    - încheie execuția altui proces
    - **Nu** este recomandată

Redirectare

- Linux
  - dup, dup2
    - descriptorii din părinte se moștenesc, implicit, în copil
- Windows
  - descriptorii ce indică fișierele către care se face redirectarea trebuie să poată fi moșteniți în procesul creat
    - membrul `bInheritHandle` al structurii `SECURITY_ATTRIBUTES` pasate lui `CreateFile` trebuie să fie TRUE
  - pentru moștenirea descriptorilor
    - parametrul `bInheritHandle` din `CreateProcess` trebuie să fie `TRUE`
  - la crearea procesului, trebuie populată structura `STARTUPINFO`
    - setarea membrilor `hStdInput`, `hStdOutput`, `hStdError` la descriptorii corespunzători
    - membrul `dwFlags` trebuie setat la `STARTF_USESTDHANDLES`

Variabile de mediu

- Linux
  - `int main(int argc, char **argv, char **environ)`
    - parametrul `environ` e un vector de șiruri de caractere de forma VARIABILĂ = VALOARE
  - `getenv`, `setenv`
    - obține/setează valoarea unei variabile de mediu
  - `unsetenv`
    - înlătură o variabilă de mediu
- Windows
  - `GetEnvironmentVariable`, `SetEnvironmentVariable`
  - setarea unei variabile cu valoarea `NULL` înlătură acea variabilă

Ce este un pipe?

- mecanisme de comunicare între procese, ce oferă acces de tip FIFO
- sistemele de operare garantează sincronizarea între operațiile de citire și de scriere la cele două capete
- două tipuri
  - **anonime**
    - pot fi folosite doar între procese înrudite
    - există doar în prezența proceselor care dețin descriptori către ele
  - **cu nume**
    - pot fi folosite între oricare două procese
    - există fizic - sunt reprezentate de fișiere speciale

Linux

- `pipe`
- `read`, `write`
- `close`

Windows

- `CreatePipe`
- `ReadFile`, `WriteFile`
- `CloseHandle`

Atenție!

- **Linux**: Când se utilizează `fork`, descriptorii sunt duplicați => numărul necesar de închideri se vor dubla. Închiderea parțială a descriptorilor conduce la blocaje în `read`.
- **Windows**: Valorile descriptorilor nu sunt direct vizibile în procesul copil și trebuie făcute cunoscute printr-o metoda alternativă.

Pipe-uri cu nume

- moduri de deschidere
  - blocant
  - neblocant
- Linux
  - `mkfifo`
- Windows
  - moduri de comunicare
    - flux de octeți
    - flux de mesaje

Server

- `CreateNamedPipe`
- `ConnectNamedPipe`

Client

- `CreateFile`
- `CallNamedPipe`

Laborator 4

Semnale

Sisteme de Operare

14 - 20 martie 2019

Semnale

- 'Întreruperi software'
- Specifice UNIX, diverse forme de echivalență pe Windows
- Generate sincron
  - Acces nevalid la memorie - `SIGSEGV` ('Segmentation fault'), `SIGBUS` ('Bus error')
  - Împărțire la 0 - `SIGFPE`
  - `abort()` - `SIGABRT`
  - Eroare la scrierea în pipe - `SIGPIPE` („Broken pipe")
- Generate asincron
  - Tastatură: `SIGINT` (`CTRL+C`), `SIGQUIT` (`CTRL+\`), `SIGTSTP` (`CTRL+Z`)
  - Sistem sau utilizator: `SIGTERM`, `SIGKILL`, `SIGUSR1`, `SIGUSR2`

Operații cu semnale

- Generare și transmitere
  - `CTRL+C`, `CTRL+\`
  - comanda `kill`, funcțiile `kill(2)`, `raise(3)`, `sigqueue(3)`)
  - direct de SO
- Blocarea unui semnal
  - `sigprocmask(2)`
- Așteptarea unui semnal
  - `pause(2)`, `sigsuspend(2)`
- Tratarea unui semnal
  - mascare, ignorare, acțiune implicită
  - asociere *signal handler*
  - `SIGKILL` si `SIGSTOP` termină procesul întotdeauna

Măști de semnale. Blocarea semnalelor

- mască pe biți reprezentând semnalele
- per proces
- `kill -l` (32 de semnale obișnuite + 32 real-time)
- sigprocmask

Tratarea unui semnal

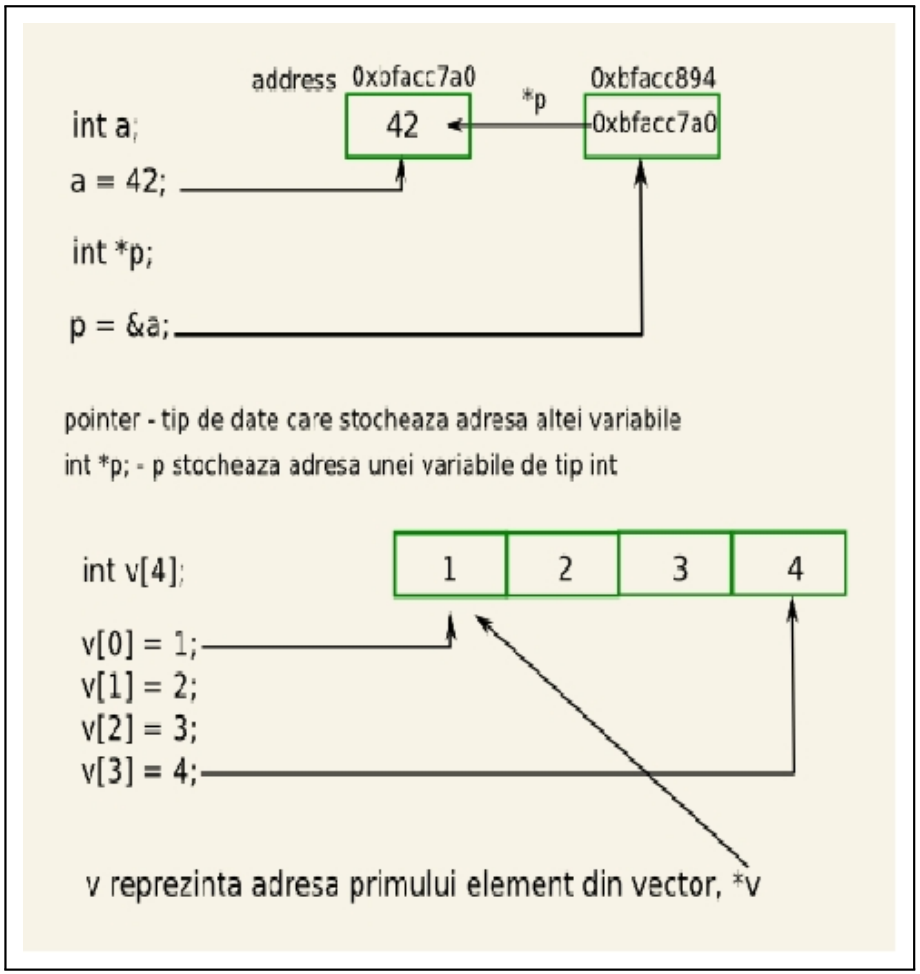
- `signal`
- `int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact)`
  - sa_handler
  - sa_sigaction

Laborator 5

Gestiunea Memoriei

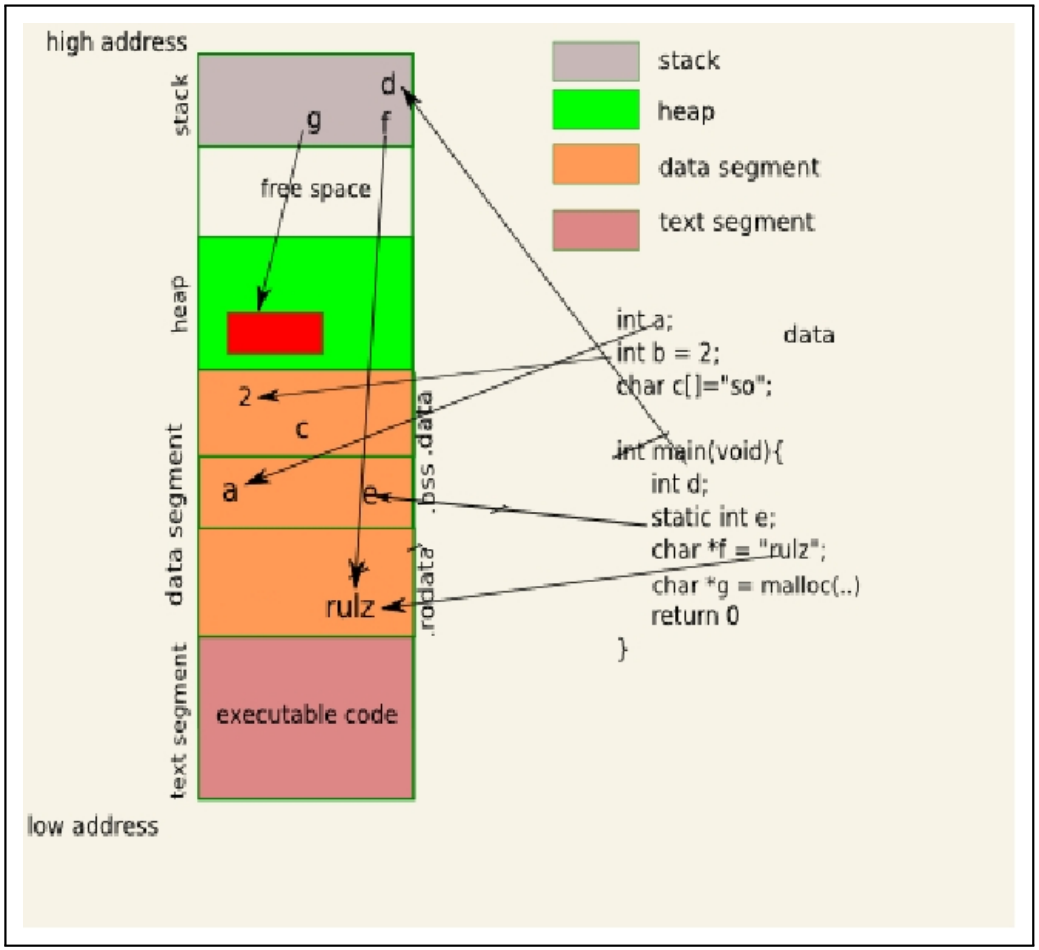
Sisteme de Operare

21 - 27 Martie 2019

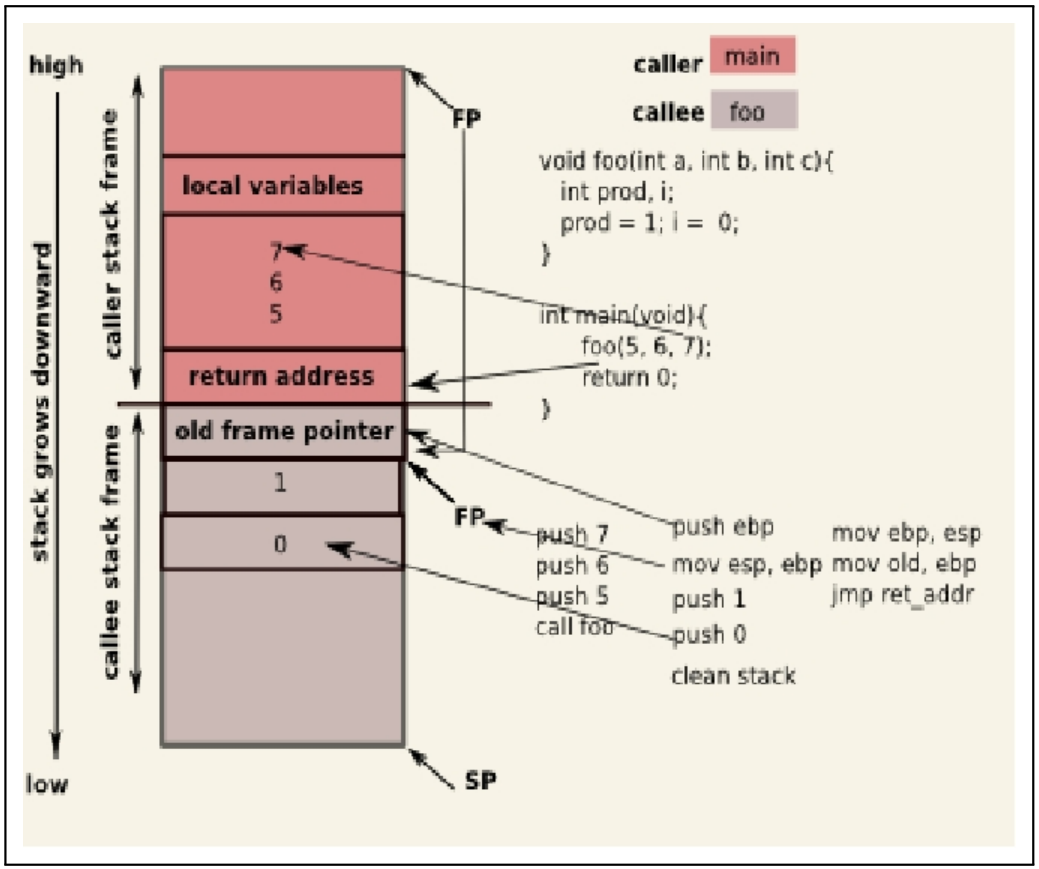


- Primitive (char, int)
- Pointer
- Array, Struct

Spațiul de adresă al unui proces



Pasarea parametrilor pe stivă



Alocarea memoriei

- Linux
  - void *malloc(size_t size);
  - void *calloc(size_t nmemb, size_t size);
  - void *realloc(void *ptr, size_t size);
  - void free(void *ptr);
- Windows
  - HANDLE HeapCreate(flOptions , dwInitialSize, dwMaximumSize);
  - BOOL HeapDestroy(hHeap);
  - LPVOID HeapAlloc(hHeap, dwFlags, dwBytes);
  - HeapReAlloc(hHeap, dwFlags, lpMem, dwBytes);
  - HeapFree(hHeap, dwFlags, lpMem);

- acces nevalid

```
char s[4]; sprintf(s,"%s","so_rulz");
```

- memory leak
  - pierderea referintei la zona de memorie

```
for(i = 0; i < 10; i++)
        a = malloc(16*sizeof(int));
free(a);
```

- dangling reference
  - accesul la o zona de memorie care a fost anterior eliberata

```
a = malloc(16*sizeof(int));
b = a; free(b);
printf("%d", a[i]);
```

  - memoria alocata pentru a a fost eliberata prin intermediul lui b

GDB

- fișierele trebuie compilate cu opțiunea -g
- se transmite ca argument numele executabilului

```
gdb ./a.out
```

- comenzi GDB utile
  - `bt` - backtrace
  - `run` - rulare
  - `step, next` - următoarea instrucțiune
  - `quit` - părăsirea depanatorului
  - `set args` - stabilirea argumentelor de rulare
  - `disassamble` - afișează codul mașină generat de compilator
  - `info reg` - afișează conținutul registrilor
  - `man gdb` - pentru mai multe detalii

mcheck, mtrace

- mcheck
  - verifică consistența heap-ului.
  - MALLOC_CHECK_=1 ./executabil
- mtrace
  - detectează memory leak-urile
  - mtrace(), muntrace(), pe regiunea inspectată.

valgrind

- suită de utilitare pentru debugging și profiling
- memcheck, callgrind, helgrind
- memcheck
  - valgrind –tool=memcheck ./executabil
  - detectează
    - folosirea de memorie neinițializată
    - citire/scriere din/in memorie după ce regiunea respectivă a fost eliberată
    - memory leak-uri
    - citirea/scriere dincolo de sfârșitul zonei alocate
    - folosirea necorespunzatoare a apelurilor malloc/new și free/delete
    - citirea/scrierea pe stivă în zone necorespunzătoare

Keywords

- Spațiu de adresă
  - .text
  - .data .rodata .bss
  - stivă
  - heap
- Alocarea memoriei
  - malloc / calloc / realloc
  - HeapAlloc / HeapReAlloc
- Dezalocarea memoriei
  - free
  - HeapFree
- accesul nevalid
  - gdb
  - mcheck
- memory leak
  - valgrind
  - mtrace

Laborator 6

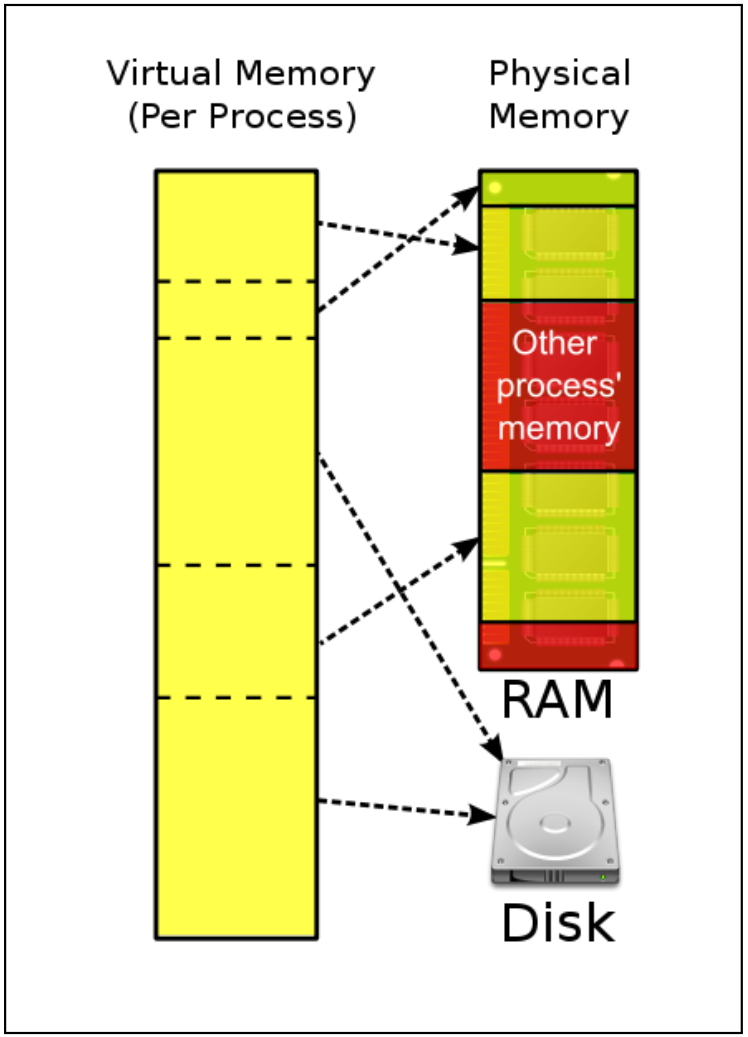
Memoria virtuală

Sisteme de Operare

28 martie - 3 aprilie 2019

Memoria virtuală(1)

- Mecanism folosit implicit..
  - de către nucleul sistemului de operare pentru a implementa o politică eficientă de gestiune a memoriei
  - ce astfel de optimizări cunoașteți?



imagine preluată de pe wikipedia.org

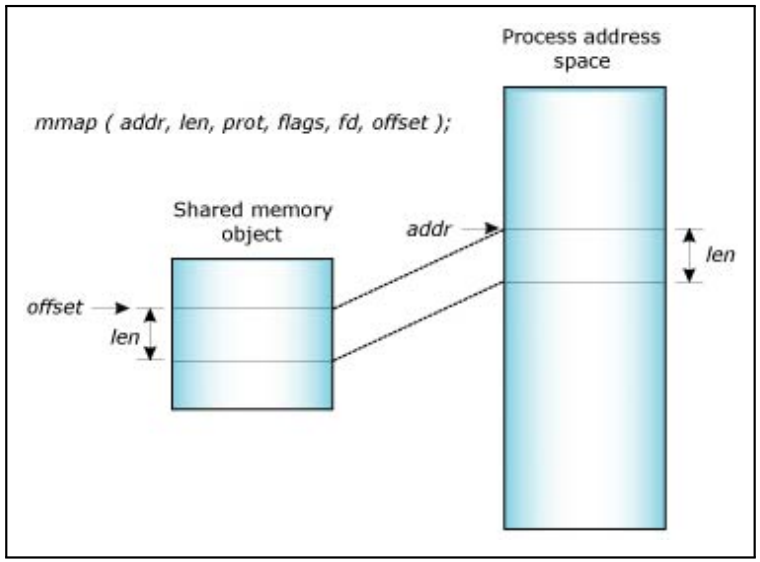
Memoria virtuală(2)

- .. dar și explicit, pentru a mapa în spațiul de adresă al unui proces:
  - fișiere
  - memorie
  - dispozitive
- Mapare fișiere
  - memorie partajată
  - paginare la cerere
  - biblioteci partajate
- Mapare memorie
  - pentru alocarea unei cantități mari de memorie
- Mapare dispozitive
  - acces direct la memoria dispozitivului
  - când ar putea fi necesar?

Mapare fișiere și memorie - Linux(1)

- Accesare fișier similar cu un vector
- Mapările pot depăși dimensiunea memoriei fizice
- Nu pot fi mapate dispozitive cu acces secvențial (socket-uri, pipe-uri)
- Unitate: pagina (număr întreg, alinieri)
- Familia de funcții mmap(2)

- mmap/munmap



- addr poate fi NULL
- prot: PROT_READ, PROT_WRITE, PROT_EXEC, PROT_NONE
- flags: MAP_PRIVATE, MAP_SHARED, MAP_FIXED, MAP_LOCKED, MAP_ANONYMOUS (pt mapare memorie)
- mapare memorie: ignoră fd și offset
- msync - sincronizare explicită fișier cu maparea din memorie

Mapare fișiere - Windows

- CreateFileMapping/OpenFileMapping
  - primește HANDLE fișier
  - tip mapare: PAGE_READONLY, PAGE_READWRITE, PAGE_WRITECOPY
- MapViewOfFile
  - primește HANDLE FileMapping
  - mod acces: FILE_MAP_READ, FILE_MAP_WRITE, FILE_MAP_COPY
- UnmapMapViewOfFile

Mapare memorie - Windows

- VirtualAlloc/VirtualAllocEx
  - tip operație: MEM_RESERVE, MEM_COMMIT, MEM_RESET
  - lpAddress poate fi NULL; multiplu de 4KB pentru alocare și 64KB pentru rezervare
- VirtualFree/VirtualFreeEx
  - tip operație: MEM_DECOMMIT, MEM_RELEASE
- Interogarea zonelor mapate VirtualQuery/VirtualQueryEx
  - adresa de start a zonei, protecție, dimensiune
  - struct _MEMORY_BASIC_INFORMATION

Excepții.Schimbarea protecției unei zone mapate

- Accese la memorie nonconforme cu drepturile
  - Linux - generează semnale SIGBUS, SIGSEGV
    - sigaction, siginfo_t
  - Windows - generează excepții
    - AddVectoredExceptionHandler, VectoredHandler
- Linux - mprotect
  - acces: PROT_READ, PROT_WRITE, PROT_EXEC, PROT_NONE
  - adresa multiplu de dimensiunea unei pagini
- Windows - VirtualProtect/VirtualProtectEx
  - pt regiuni alocate cu VirtualAlloc/VirtualAllocEx folosind MEM_RESERVE

Blocarea paginării

- Utilă pentru procese care trebuie să execute anumite acțiuni la momente de timp bine determinate
- Nu se va mai face swap out - accesele ulterioare nu mai produc page fault
- Linux
  - mlock
  - mlockall
    - flags: MCL_CURRENT, MCL_FUTURE
  - munlock/munlockall
- Windows
  - VirtualLock/VirtualLockEx
    - rezultat: TRUE succes, FALSE altfel
  - VirtualUnlock/VirtualUnlockEx

Laborator 07

Profiling and Debugging

Sisteme de Operare

4 - 10 aprilie 2019

Tehnici de profiling

- Instrumentare
- Eșantionare

- Presupune modificarea codului
- Introduce latențe
- Asigură o precizie sporită
- Nu are nevoie de suport SO

Eșantionare (sampling)

- Nu implică modificarea codului
- Are nevoie de suport SO
- Are nevoie de suport hardware
- Se fac verificări periodice

perf

- performance counters
  - Registre speciale disponibile pe procesoarele moderne
  - Numără anumite evenimente hardware (instrucțiuni, etc)
- perfcounters
  - Subsistem în nucleu de gestiune a performance counters
  - Hardware/software counters, tracepoints
  - Per thread/cpu/whole system
- perf
  - Utilitar userspace (linux/tools/perf).
  - Interfață asemănătoare cu git (subcomenzi).
  - list, stat, record, report, top

Utilizare perf

- perf [–version] [–help] COMMAND [ARGS]
- COMMAND
  - list - listează toate evenimentele disponibile de urmărit cu perf.
  - stat - rulează o comandă și afișează informații statistice despre rulare.
  - top - afișează statistici despre un eveniment în timp real.
  - record - rulează o comandă și salvează profilul în perf.data.
  - report - interpretează un profil salvat în perf.data
  - sched - măsoară proprietăți ale planificatorului (e.g latență).

Debugging

- strace
- gdb
- valgrind
- printf

Concluzii

- "Premature optimization is the root of all evil"
- 80% din timp, rulează 20% din cod
- There are lies, damned lies, and statistics
- Avem nevoie de unelte foarte bune de debugging, mai ales pentru proiectele mari

Laborator 8

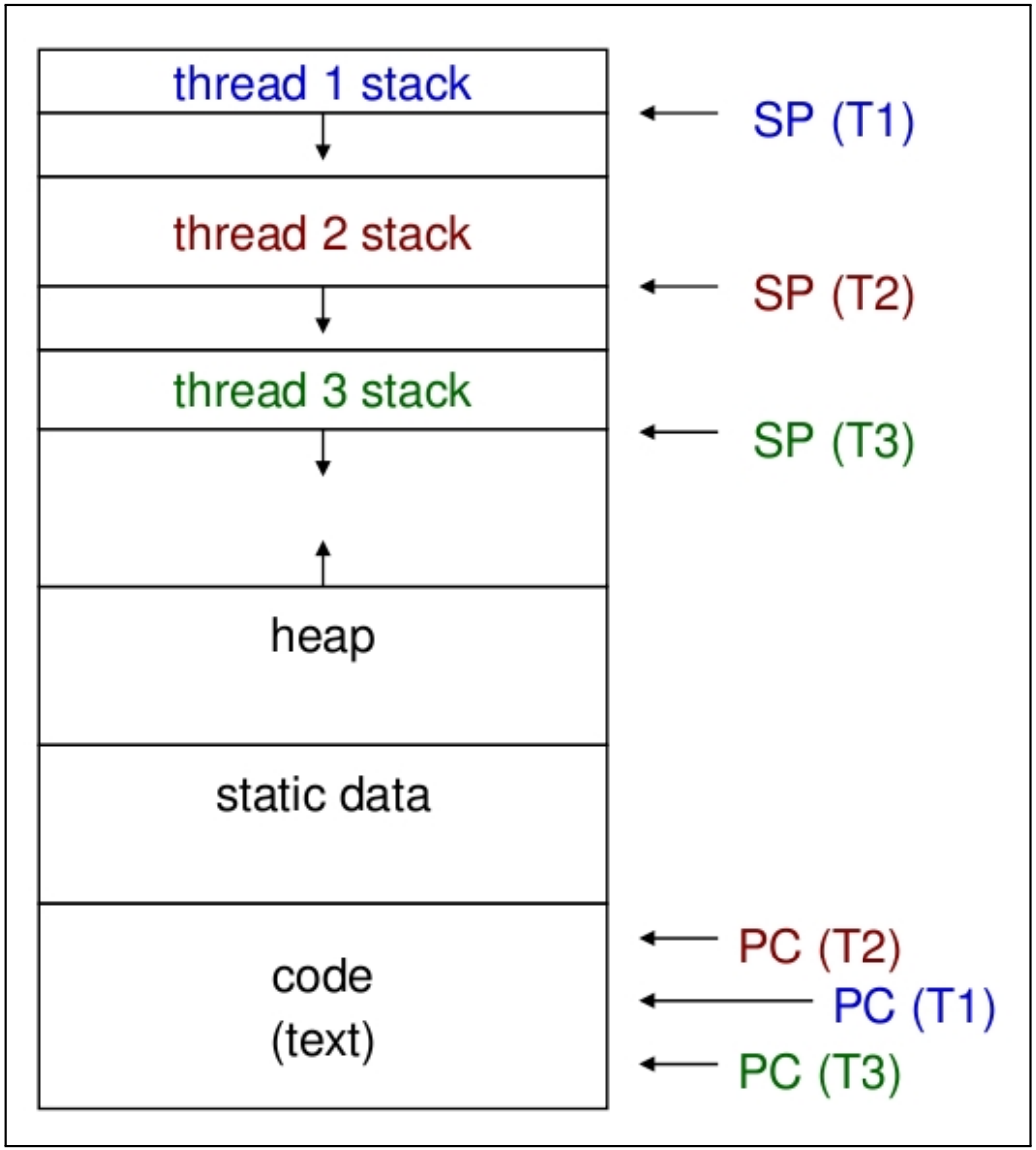
Thread-uri

Sisteme de Operare

11 - 17 aprilie 2019

Thread Control Block

- Informații partajate
  - Spațiu de adresă
  - Heap, Data
  - Semnale și handlere
  - I/O și fișiere
- Informații proprii
  - Starea
  - Regiștrii
  - Program counter
  - Stiva
  - Masca de semnale
  - errno



Avantaje

- Pot comunica între ele fără a implica kernelul
- Asigură o folosire mai eficientă a resurselor calculatorului
- Mai puțin timp pentru crearea/distrugerea unui thread decât a unui proces
- Comutarea între 2 thread-uri mai rapidă decât între procese
- Paralelizarea are sens in 2 situații:
  - Task-uri I/O bound pe același CPU
  - Task-uri CPU bound pe mai multe core-uri
- dezavantaj: sincronizare (overhead + model de programare complex)

- Nu întotdeauna dorim să partajăm totul cu celelalte thread-uri => e nevoie de un storage thread-specific
- O zonă din stiva fiecărui thread este organizată sub forma unui Map cu perechi (cheie, valoare)
- Atenție la crearea de foarte multe thread-uri cu stivă mare (se poate epuiza spațiul de adrese)

- Există 3 tipuri de implementări : ULT, KLT, hibride

ULT vs. KLT

- **User-Level Threads**
  - Kernel-ul nu este conștient de existența lor
  - Schimbarea de context nu implică kernelul => rapidă
  - Planificarea poate fi aleasă de aplicație
  - Aceste thread-uri pot rula pe orice SO
  - Dacă un thread apelează ceva blocant toate thread-urile planificate de aplicație vor fi blocate
  - 2 fire ale unui proces nu pot rula simultan pe 2 procesoare
- **Kernel-Level Threads**
  - Schimbarea de context între thread-uri ale aceluiași proces implică kernel-ul => viteza de comutare este mică
  - Blocarea unui fir nu înseamnă blocarea întregului proces
  - Dacă avem mai multe procesoare putem lansa în execuție simultană mai multe thread-uri ale aceluiași proces

- **Thread-safe** - Operații sigure în context multithreading
  - o funcție este thread-safe dacă și numai dacă va produce mereu rezultatul corect atunci când este apelată concurent, în mod repetat, din mai multe thread-uri
- Tipuri de funcții thread-unsafe
  - Funcții ce nu protejează variabilele partajate
  - Funcții ce întorc pointer la o variabilă statică
  - Funcții ce apelează funcții thread-unsafe
- Funcțiile **reentrante** sunt cele care nu referă date partajate
  - nu lucrează cu variabile globale/statice
  - nu apelează funcții non-reentrante
  - sunt un subset al funcțiilor thread-safe
- un apel reentrant în execuție nu afectează un alt apel simultan

- Mutex (POSIX, Win32)
- Semafor (POSIX, Win32)
- Secțiune critică (Win32)
- Variabilă de condiție (POSIX, Win32)
- Barieră (POSIX, Win32)
- Operații atomice cu variabile partajate (Win32)
- Thread pooling (Win32)

Laborator 9

Thread-uri - Windows

Sisteme de Operare

18 - 24 aprilie 2019

Windows Threads API

- Operații cu thread-uri
  - CreateThread
  - ThreadProc
  - WaitForSingleObject
  - ExitThread
  - GetCurrentThread
- Thread Local Storage
  - TlsAlloc
  - TlsFree
  - TlsGetValue
  - TlsSetValue

Mecanisme de sincronizare

- Mutex (POSIX, Win32)
- Semafor (POSIX, Win32)
- Secțiune critică (Win32)
- Variabilă de condiție (POSIX)
- Barieră (POSIX)
- Eveniment (Win32)
- Operații atomice cu variabile partajate (Win32)
- Thread pooling (Win32)

Mutex (Win32)

- Creare mutex
  - `HANDLE CreateMutex(LPSECURITY_ATTRIBUTES lpMutexAttributes, BOOL bInitialOwner, LPCTSTR lpName)`
- Deschidere mutex deja creat
  - `HANDLE OpenMutex(DWORD dwDesiredAccess, BOOL bInheritHandle, LPCTSTR lpName)`
- Așteptare/acaparare mutex
  - Funcțiile din familia `WaitForSingleObject`
- Eliberare mutex
  - `BOOL ReleaseMutex(HANDLE hMutex)`

Semafor (Win32)

- Creare semafor
  - `HANDLE CreateSemaphore(LPSECURITY_ATTRIBUTES sematttr, LONG initial_count, LONG maximum_count, LPCTSTR name)`
- Deschidere semafor deja existent
  - `HANDLE OpenSemaphore(DWORD dwDesiredAccess, BOOL bInheritHandle, LPCTSTR name)`
- Așteptare/decrementare semafor
  - Funcțiile din familia `WaitForSingleObject`
- Incrementare, cu `lReleaseCount`
  - `BOOL ReleaseSemaphore(HANDLE hSemaphore, LONG lReleaseCount, LPLONG lpPreviousCount)`

Secțiune critică

- `CRITICAL_SECTION`
- Sincronizare DOAR între firele de execuție ale **aceluiași proces**
- Inițializare/Distrugere secțiune critică
  - `void InitializeCriticalSection(LPCRITICAL_SECTION pcrit_sect)`
  - `void DeleteCriticalSection(LPCRITICAL_SECTION pcrit_sect)`
- Intrare în secțiune critică
  - `void EnterCriticalSection(LPCRITICAL_SECTION lpCriticalSection)`
  - `BOOL TryEnterCriticalSection(LPCRITICAL_SECTION lpCriticalSection)`
- Ieșire din secțiune critică
  - `void LeaveCriticalSection(LPCRITICAL_SECTION lpCriticalSection)`

Eveniment

- Două tipuri: manual-reset, auto-reset
- Creare eveniment
  - `HANDLE WINAPI CreateEvent(LPSECURITY_ATTRIBUTES lpEventAttributes, BOOL bManualReset, BOOL bInitialState, LPCTSTR lpName);`
- Semnalizare eveniment
  - `BOOL WINAPI SetEvent(HANDLE hEvent);`
  - `BOOL WINAPI PulseEvent(HANDLE hEvent);`
  - `BOOL WINAPI ResetEvent(HANDLE hEvent);`
- Așteptarea unui eveniment
  - Funcțiile din familia `WaitForSingleObject`

Operații atomice cu variabile partajate

- Incrementare/Decrementare variabilă
  - `LONG InterlockedIncrement(LONG volatile *lpAddend)`
  - `LONG InterlockedDecrement(LONG volatile *lpDecend)`
- Atribuire atomică
  - `LONG InterlockedExchange(LONG volatile *Target, LONG Value)`
  - `LONG InterlockedExchangeAdd(LPLONG volatile Addend, LONG Value)`
  - `PVOID InterlockedExchangePointer(PVOID volatile *Target, PVOID Value)`
- Atribuire atomică condiționată
  - `LONG InterlockedCompareExchange(LONG volatile *dest, LONG exchange, LONG comp)`
  - `PVOID InterlockedCompareExchangePointer(PVOID volatile *dest, PVOID exchange, PVOID comp)`

Thread pooling

- Fiecare task primește un thread din pool
- Eliminare overhead creare/terminare fire de execuție
- Task-urile pot fi:
  - Executate **imediat**
  - Executate **mai târziu** (operații de așteptare + funcție callback asociată)
    - Așteptarea terminării unei operații I/O asincrone
    - Așteptarea expirării unui TimerQueue
    - Funcții de așteptare înregistrate

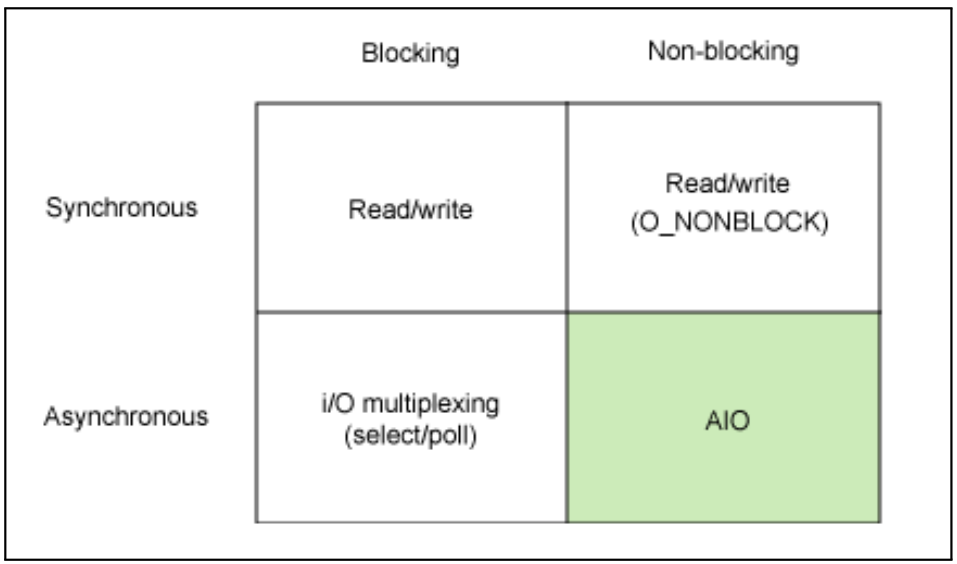
Laborator 10

Operații I/O avansate - Windows

Sisteme de Operare

2 - 8 Mai 2019

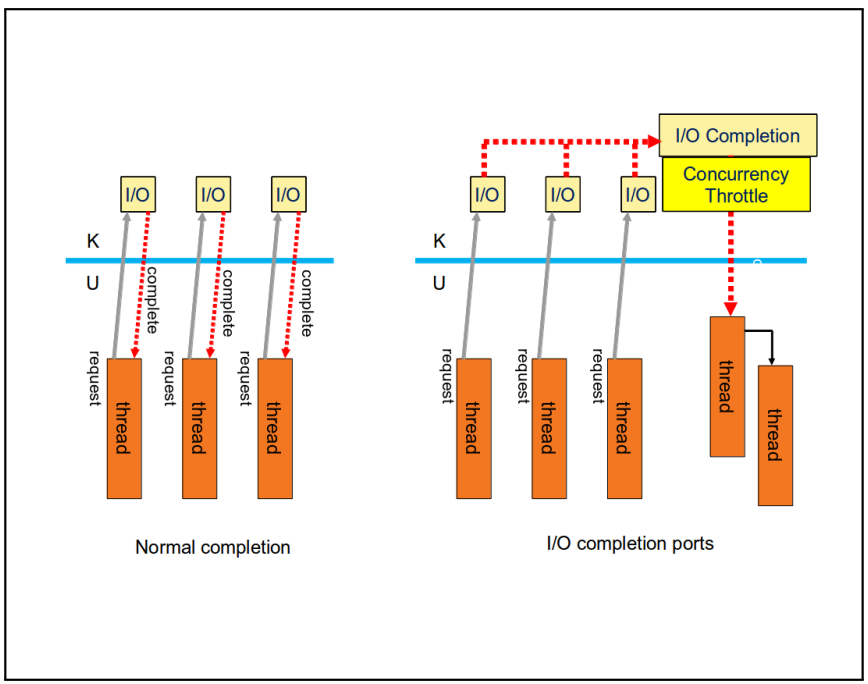
Modele de operații I/O



- Operații blocante
  - Wait
- Operații non-blocante
  - Don't wait
- Notificare
  - Sincron
  - Asincron

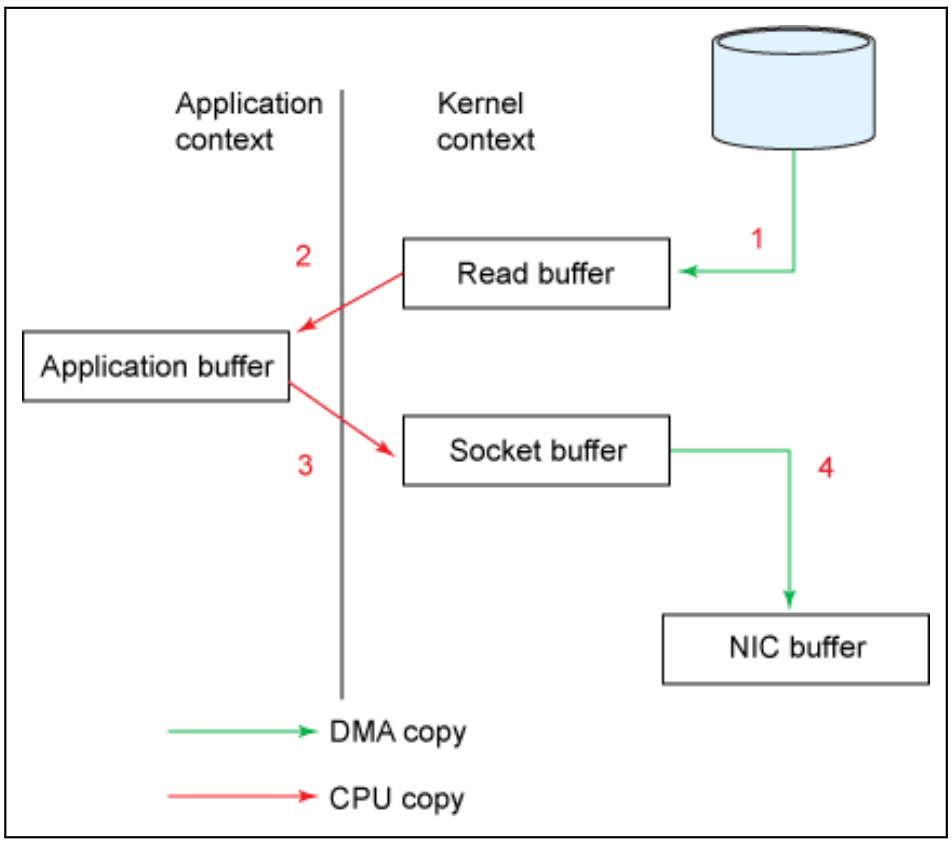
- **Overlapped I/O**
- File handle creat cu flag-ul FILE_FLAG_OVERLAPPED
- Structura OVERLAPPED folosită de ReadFile, WriteFile
  - Codul de eroare pentru cererea I/O
  - Numărul de octeți transferați
  - Poziția în fișier de unde se face operația I/O
  - Un eveniment care va fi semnalizat când operația se termină
- GetOverlappedResult
  - Obține rezultatul unei operații I/O overlapped

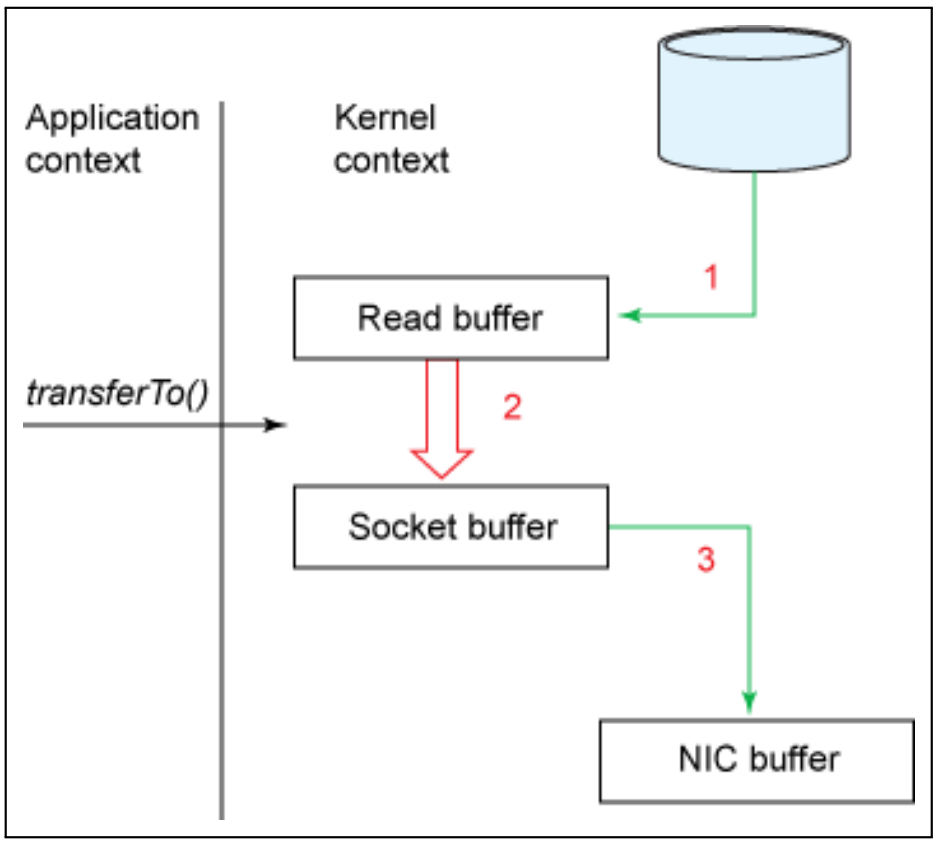
- Obiect în kernel care asociază un set de overlapped handles cu un set de fire de execuție
- Firele de execuție așteaptă ca operațiile de I/O să se încheie



- CreateIoCompletionPort
- GetQueuedCompletionStatus

- Evită copierea datelor dintr-o zonă într-alta
- TransmitFile - transmite un fișier peste un socket





Laborator 11

Operații I/O avansate - Linux

Sisteme de Operare

9 - 15 Mai 2019

Multiplexare

- mai multe canale, un singur fir de execuție
  - canale = set de descriptori
  - evenimente IN/OUT
- apeluri stateless
  - înregistrare interes cuplată cu așteptare
  - select, poll
- apeluri stateful
  - înregistrare interes separată de așteptare
  - epoll

Mecanisme

- select
  - readfds, writefds, exceptfds
  - simplu, ineficient
- poll
  - pollfd (fd, events, revents)
  - simplu, la fel de ineficient
- epoll
  - struct epoll_event (events, data)
  - level-triggered vs edge-triggered
  - simplu, eficient

Generalizarea multiplexării

- eventfd
  - notificări pentru evenimente
- signalfd
  - notificări pentru primire de semnale
- timerfd
  - notificări pentru timere

Operații asincrone

- POSIX AIO
  - -lrt, dacă este suportat
  - aio_read, aio_write, aio_suspend ...
- kernel AIO
  - -laio
  - struct iocb
  - AIO context
  - io_setup, io_submit, io_destroy, io_getevents ...

IO Optimization

- zero-copy
  - splice
- vectored IO
  - struct iovec
  - readv, writev

Laborator 12

Implementarea sistemelor de fișiere

Sisteme de Operare

16 - 22 Mai 2019

Dispozitive

- Reale vs. virtuale
- Caracter vs. bloc

- Gestionarea dispozitivelor de către kernel
- Operații I/O
  - `open, close`
  - `read, write`
  - `ioctl, mmap`

Device nodes

- Fișiere speciale pentru interacțiunea cu device driver-ul
- Major și minor
- `mknod`
- `stat, fstat, lstat`
  - `struct stat`

Sistem de fișiere

- Colecție organizată de fișiere și directoare
- Clasificare
  - Disc: ext2, ext3, ext4, ntfs, fat, reiserfs
  - Rețea: nfs, smbfs, ncp
  - Virtuale: procfs, sysfs, sockfs, pipefs

Operații cu sistemul de fișiere

- `mount, umount`
- `symlink, unlink`
- `mkdir, rmdir, remove`
- `opendir, readdir, struct dirent`
- `getcwd, chdir, chroot`
- `realpath, dirname, basename`