Operativni sistemi 2

Vežbe 5
Upravljanje diskovima i arhitektura operativnih sistema

Lista predmeta:
IR smer - 13e113os2@lists.etf.rs
SI Smer - 13s113os2@lists.etf.rs

Zadatak

Data je sekvenca zahteva (brojevi cilindara): 98, 183, 37, 122, 14, 124, 65, 67. Glava se nalazi na poziciji 53. Prikazati raspored opsluživanja zahteva po svi algoritmima radjenim na predavanjima.

98, 183, 37, 122, 14, 124, 65, 67. Glava se nalazi na poziciji 53.

FCFS

$53 \rightarrow^{45} 98 \rightarrow^{85} 183 \rightarrow^{146} 37 \rightarrow^{85} \underline{122 \rightarrow^{108} 14 \rightarrow^{110} 124} \rightarrow^{59} 65 \rightarrow^{2} 67$

predjeno: 640

SSTF

$53 \rightarrow^{12} 65 \rightarrow^{2} 67 \rightarrow^{30} 37 \rightarrow^{23} 14 \rightarrow^{84} 98 \rightarrow^{24} 122 \rightarrow^{2} 124 \rightarrow^{59} 183$

predjeno: 236

SCAN (ide naniže)

$53 \rightarrow^{16} 37 \rightarrow^{23} 14 \rightarrow^{14} 0 \rightarrow^{65} 65 \rightarrow^{2} 67 \rightarrow^{31} 98 \rightarrow^{24} 122 \rightarrow^{2} 124 \rightarrow^{59} 183$

predjeno: 236

C-SCAN (ide naviše)

$53 \rightarrow 65 \rightarrow 67 \rightarrow 98 \rightarrow 122 \rightarrow 124 \rightarrow 183 \rightarrow \text{Max} \rightarrow 0 \rightarrow 14 \rightarrow 37$

C-LOOK (ide naviše)

$53 \rightarrow 65 \rightarrow 67 \rightarrow 98 \rightarrow 122 \rightarrow 124 \rightarrow 183 \rightarrow 14 \rightarrow 37$

Zadatak – Septembar 2008.

U nekom sistemu za upravljanje diskom implementiran je C-LOOK algoritam raspoređivanja. Jedan zahtev za pristup disku predstavljen je strukturom:

```
struct DiskReq {
  //...
  unsigned long cylinder;
  DiskReq* next;
};
DiskReq* head;
```

Polje `cylinder` ove strukture označava broj cilindra na koji se zahtev odnosi, a zahtevi su uvezani u ulančanu listu (`head`) preko pokazivača `next` ove strukture, uređeni po redosledu pristizanja. Implementirati operaciju:

```
DiskReq* getDiskRequest (DiskReq* prev);
```

koja vraća sledeći zahtev za opsluživanje po C-LOOK algoritmu, pri čemu argument ove operacije označava zahtev koji je prethodno odabran za opsluživanje i koji je već izbačen iz liste. Ova operacija ne treba da izbacuje zahtev iz liste, samo da vrati odabrani.

```
DiskReq* getDiskRequest (DiskReq* prev) {
  // Find the closest higher cylinder, if any:
  unsigned long cyl = prev->cylinder;
  DiskReq* selected = 0;
  unsigned long minDist = ~0;
  for (DiskReq* cur = head; cur!=0; cur=cur->next)
    if (cur->cylinder>=cyl && (cur->cylinder-cyl)<minDist) {
      selected = cur;
      minDist = cur->cylinder-cyl;
    }
  if (selected!=0) return selected;
  // Loop back and find the minimal cylinder:
  selected = head;
  if (selected==0) return 0;  // Empty queue;
  unsigned long minCyl = head->cylinder;
  for (DiskReq* cur = head->next; cur!=0; cur=cur->next)
    if (cur->cylinder<minCyl) {
      selected = cur;
      minCyl = cur->cylinder;
    }
  return selected;
}
```

Zadatak – Januar 2017.

U nekom sistemu klasa `DiskScheduler` data dole realizuje raspoređivač zahteva za pristup disku po *C-SCAN* algoritmu, ali uz sprečavanje izgladnjivanja ograničavanjem vremena čekanja zahteva na opsluživanje, na sličan način kako to radi i Linux (prilikom smeštanja novog zahteva u red, zahtevu se dodeljuje krajnji vremenski rok, engl. *deadline*, do kog se zahtev mora opslužiti). Prilikom smeštanja zahteva tipa `DiskRequest` u red zahteva operacijom `put()`, zahtev se smešta u dve dvostruko ulančane liste: prvu uređenu po broju cilindra na koji se zahtev odnosi, za *C-SCAN* algoritam (kružna lista u kojoj `scanHead` ukazuje na zahtev koji je naredni na redu za opsluživanje), i drugu uređenu hronološki, po vremenu isteka roka do kog se zahtev mora opslužiti (lista čija je glava `edfHead`, EDF – *Earliest Deadline First*). U strukturi `DiskRequest` polja `edfNext` i `edfPrev` predstavljaju pokazivače za (dvostruko) ulančavanje u *EDF* listu, a polja `scanNext` i `scanPrev` pokazivače za (dvostruko) ulančavanje u *C-SCAN* listu; polje `deadline` sadrži razliku između trenutka isteka vremenkog roka datog zahteva i trenutka isteka vremenskog roka zahteva koji mu hronološki prethodi u listi EDF; za prvi zahtev u toj listi, ovo polje predstavlja relativno vreme isteka vremenskog roka tog zahteva u odnosu na sadašnji trenutak (može biti i negativno ako je taj rok istekao).
Implementirati operaciju `get` klase `DiskScheduler` koja iz reda uzima zahtev koji sledeći treba opslužiti.

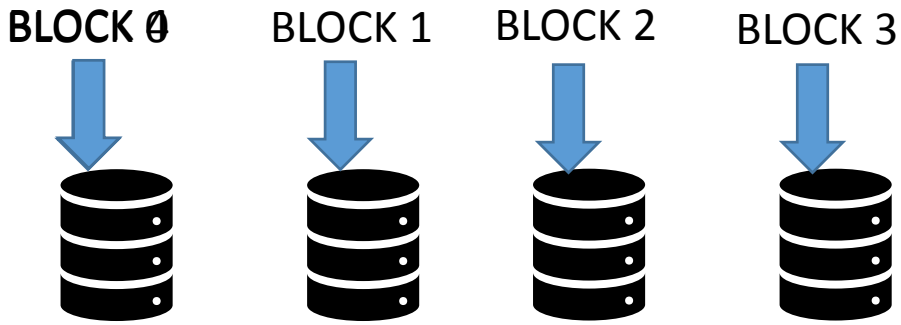
```
struct DiskRequest {
  DiskRequest *edfNext, *edfPrev;
  DiskRequest *scanNext, *scanPrev;
  long deadline;
...
};
class DiskScheduler {
public:
  DiskScheduler ();
  DiskRequest* get ();
  void put (DiskRequest*);
private:
  DiskRequest *edfHead, *scanHead;
};
DiskScheduler::DiskScheduler () : edfHead(0),
scanHead(0) {}
```

```
DiskRequest* DiskScheduler::get () {
  if (!edfHead || !scanHead) return 0;
  DiskRequest* req = 0;
  // Select the request:
  if (edfHead->deadline<=0)
    req = edfHead;
  else
    req = scanHead;
  // Remove it from the EDF queue:
  if (req->edfPrev) req->edfPrev->edfNext = req->edfNext;
  else edfHead = req->edfNext;
  if (req->edfNext) req->edfNext->edfPrev = req->edfPrev;
  req->edfPrev = req->edfNext = 0;

  // Remove it from the SCAN queue
  if (req->scanNext!=req)
    scanHead = req->scanNext;
  else
    scanHead = 0;
  req->scanPrev->scanNext = req->scanNext;
  req->scanNext->scanPrev = req->scanPrev;
  req->scanNext = req->scanPrev = 0;

  return req;
}
```

RAID

Mirroring

Data Striping

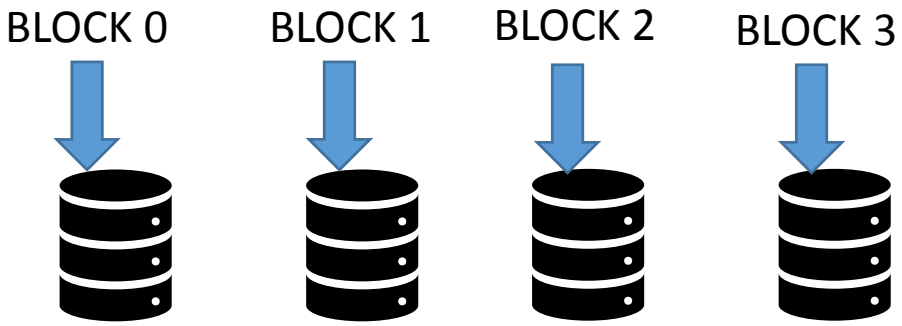


Zadatak – Septembar 2014.

Neki sistem implementira *block-striping* RAID0 funkcionalnost u softveru. Implementirati funkciju:

```
int mapRAIDBlock(unsigned long blk, unsigned long& d, unsigned long& b);
```

koja treba da preslika logički broj bloka `blk` (logički broj za celu RAID0 strukturu kao jedinstven virtuelni disk) u redni broj diska `d` i broj bloka na tom disku `b` u kome se nalazi taj blok. Svi diskovi su identični, imaju isti broj blokova (taj broj blokova na jednom disku vraća funkcija `getNumOfBlocks()`), a broj diskova vraća funkcija `getNumOfDisks()`. U slučaju greške (prekoračenja broja blokova na disku), ova funkcija treba da vrati -1, a u slučaju uspeha treba da vrati 0.




```
int mapRAIDBlock(unsigned long blk, unsigned long& d,
unsigned long& b) {
  static const int dsks = getNumOfDisks(), blks =
getNumOfBlocks();
  d = blk%dsks;
  b = blk/dsks;

  if (b>=blks) return -1;
  else return 0;
}
```

Zadatak – Septembar 2015.

Neki sistem implementira *block-striping* RAID10 funkcionalnost u softveru. Svi diskovi su identični, imaju isti broj blokova (taj broj blokova na jednom disku vraća funkcija `getNumOfBlocks()`). Postoji *n* parova diskova koji se ogledaju, pri čemu broj *n* vraća funkcija `getNumOfDisks()`.
Definisan je globalni niz `disks` od *n* elemenata, gde svaki element niza predstavlja jedan par diskova objektom sledeće klase:

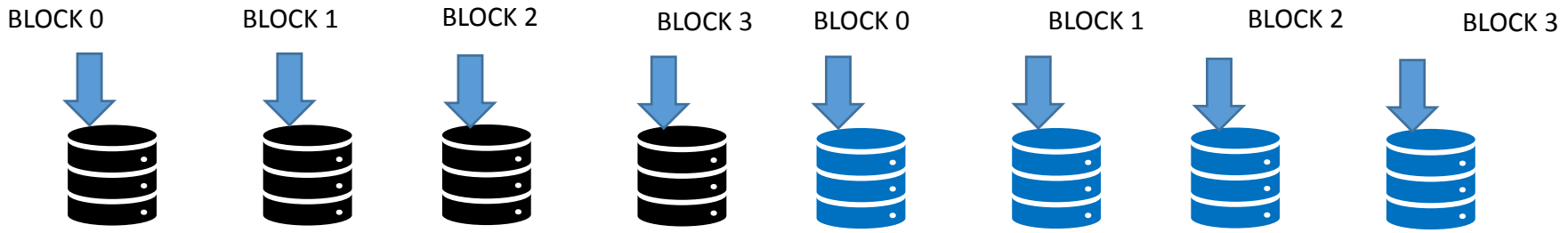
```
class DiskPair {
public:
  int isOK (int diskNo);
  int getLastRead ();
  void putReadRequest (int diskNo, unsigned blockNo, void* buffer);
};
```

U operacijama ove klase celobrojni argument `diskNo` može imati vrednost 0 ili 1 koja ukazuje na jedan ili drugi disk u paru. Funkcija `isOK()` vraća 1 ako je dati disk u paru ispravan i funkcionalan, a 0 ako je otkazao ili nije instaliran. Funkcija `getLastRead()` vraća broj (0 ili 1) diska u paru kome je poslednjem bio izdat zahtev za čitanje. Funkcija `putReadRequest`() zadaje novi zahtev za čitanje na disku `diskNo` u paru sa datim parametrima – redni broj bloka na tom fizičkom disku i adresa bafera u memoriji za čitanje, i odmah vraća kontrolu pozivaocu.

Implementirati funkciju:

```
int readBlock (unsigned blk, void* buffer);
```

koja treba da zada novi zahtev za čitanje sa logičkog bloka broj `blk` (logički broj za celu
RAID10 strukturu kao jedinstven virtuelni disk) u bafer čija je adresa data. Ako su oba diska u paru na koji se preslikava dati logički blok ispravna, zahteve za čitanjem treba zadavati diskovima u paru naizmenično, radi raspodele opterećenja i paralelizacije operacija. U slučaju uspeha, operacija treba da vrati 0. Ukoliko je redni broj bloka prekoračio dozvoljen opseg, operacija treba da vrati -1. Ukoliko su oba diska u paru na koji se preslikava dati blok neispravna, operacija treba da vrati -2.




```
int readBlock (unsigned blk, void* buffer) {
  static const int dsks = getNumOfDisks(), blks =
getNumOfBlocks();
  d = blk%dsks;
  b = blk/dsks;

  if (b>=blks) return -1; // Overflow

  int diskNo = disks[d].getLastRead();
  if (disks[d].isOK(1-diskNo))
    diskNo = 1-diskNo;

  if (disks[d].isOK(diskNo))
    disks[d].putReadRequest(diskNo,b,buffer));
  else
    return -2; // Complete disk[d] pair failure

  return 0;
}
```

Zadatak – Januar 2018.

Neki sistem implementira RAID 4 funkcionalnost u softveru. U celoj RAID 4 strukturi, odnosno u njegovoj pruzi (engl. *stripe*) nalazi se `StripeSize+1` disk, od kojih je prvih `StripeSize` diskova sa blokovima za podatke, a poslednji je disk sa blokovima parnosti. Logički susedni blokovi sa podacima se raspoređuju redom po prvih `StripeSize` diskova (logički blok broj `n` preslikava se u blok broj `n/StripeSize` diska broj `n%StripeSize`), a blok parnosti se formira od `StripeSize` logičkih blokova sa istim celobrojnim količnikom `n/StripeSize` tako što se za svaki odgovarajući bajt sa rednim brojem *b* u tim blokovima i svaki njegov bit u razredu *i* formira odgovarajući bit parnosti u istom bitu *i* istog bajta *b* bloka parnosti, i to po pravilu parne parnosti (ukupan broj jedinica u bitima podataka i bitu parnosti je paran, engl. *parity even*).

Data je funkcija:

```
int readBlock (unsigned dsk, unsigned long blk, byte* buffer);
```

koja učitava blok broj `blk` sa ispravnog diska broj `dsk` u dati bafer. Blok je veličine `BlockSize` bajtova, a definisan je tip `byte` koji predstavlja jedan bajt. Ova funkcija vraća 0 u slučaju uspeha, a negativan kod greške u suprotnom.

Implentirati sledeću funkciju koja se poziva u slučaju da je disk broj `dsk` van funkcije (pokvaren, isključen), kako bi pomoću blokova sa ostalih diskova u RAID strukturi oporavila traženi blok broj `blk`; rezultat oporavka treba da se smesti u dati bafer, a funkcija treba da vrati status poput prethodne:

```
int recoverBlock (unsigned dsk, unsigned long blk, byte* buffer);
```

```
int recoverBlock (unsigned dsk, unsigned long blk,
byte* buffer) {
  byte auxBuf[BlockSize]; // Auxiliary buffer
  for (int i=0; i<BlockSize; i++)
    buffer[i] = 0;
  for (unsigned d=0; d<=StripeSize; d++) {
    if (d==dsk) continue; // Disk dsk is out of order,
skip it
    int status = readBlock(d,blk,auxBuf);
    if (status!=0) return status;
    for (int i=0; i<BlockSize; i++)
      buffer[i] ^= auxBuf[i];
  }
  return 0;
}
```

Zadatak – April 2006.

a)(5) Neki *storage* sistem sa više diskova, visoke pouzdanosti, označen je na sledeći način: RAID-5/12+1, pri čemu je kapacitet svakog diska 100GB. Koliki je efektivni kapacitet (za „korisne“ informacije koje koristi fajl sistem) ove strukture diskova?

Odgovor:

RAID 5 (*block-interleaved distributed parity*): podatke i bite parnosti rasipa po svim diskovima – za svaki paket od *N* blokova, jedan disk (bilo koji) čuva parnost, ostali podatke, ciklično

12 * 100GB = 1200GB

b)(5) Neki *storage* sistem sa više diskova, visoke pouzdanosti, označen je na sledeći način: RAID-1/2x4x100GB, pri čemu je kapacitet svakog diska 100GB. Koliki je efektivni kapacitet (za „korisne“ informacije koje koristi fajl sistem) ove strukture diskova?

Odgovor:

RAID 1: *mirroring* bez ikakve paralelizacije;

4 * 100GB = 400GB

Zadatak – Oktobar 2006.

a)(5) Šta je osnovni nedostatak RAID 1 u odnosu na RAID 5?
Odgovor:
RAID 1 je jednostavniji za implementaciju, ali ima slabije iskorišćenje (ukupan prostor je duplo veći u odnosu na količinu korisnih informacija, odnos je 2:1), dok je kod RAID 5 taj odnos povoljniji i iznosi n:n-1.

b)(5) Objasniti kako SSTF algoritam raspoređivanja zahteva za pristup disku može da dovede do izgladnjivanja?

Odgovor:
Tako što stalno stižu novi zahtevi koji se odnose na cilindre bliske onome na kome se nalazi glava, dok ostali zahtevi za udaljenim cilindrima ostaju da čekaju.

Zadatak – Septembar 2006.

a)(5) Smisao virtuelnih mašina za određene programske jezike, kao što su Java VM ili .Net CLR, jeste da korisničke programe pisane na tim jezicima i prevedene na međukod za te virtuelne mašine učine nezavisnim od platforme (računara i operativnog sistema domaćina) i time prenosive na različite platforme. Ali da li je sama takva virtuelna mašina zavisna od platforme? Zašto?

Odgovor:
Virtuelna mašina jeste zavisna od platforme, jer je to program (u binarnom obliku) koji se izvršava od strane mašine-domaćina (program se sastoji od naredbi baš za taj računar) i na operativnom sistemu domaćinu (koristi sistemske pozive tog OS-a).

b)(5) Virtuelna mašina za neki programski jezik (npr. Java VM) se izvršava kao jedan proces na sistemu-domaćinu, a u okviru svog izvršavanja pokreće niti koje su definisane u programu na izvornom jeziku koji VM izvršava. Da li to obavezno znači da ako se neka nit iz programa blokira na nekom sistemskom pozivu (npr. I/O operaciji), da se i ceo proces VM (cela virtuelna mašina) blokira?
Odgovor:
Ne mora da znači. VM može da preslikava (implementira) niti iz programa u niti operativnog sistema-domaćina, ako takve postoje. U tom slučaju, blokiranje niti iz programa znači samo blokiranje jedne niti OS domaćina, ali ne i ostalih niti u okviru istog VM procesa.

Zadatak – Februar 2010.

U nekom operativnom sistemu za neki računar sa dvoadresnim RISC procesorom i LOAD/STORE arhitekturom sistemski pozivi realizuju se softverskim prekidom koji se izaziva instrukcijom `TRAP`. Adresni deo ove instrukcije nosi redni broj sistemskog poziva koji jedinstveno identifikuje taj poziv. Sistemski poziv u registru R1 očekuje pokazivač na strukturu podataka koja nosi argumente sistemskog poziva, specifične za svaki poziv. Korišćenjem `asm` bloka na jeziku C/C++ napisati bibliotečnu funkciju `create_process()` koja je deo API ovog operativnog sistema i koja obavlja sistemski poziv broj 25h za kreiranje procesa nad zadatim programom. U `asm` bloku treba da budu samo najneophodnije naredbe, ostatak ove funkcije treba da bude realizovan na jeziku C/C++.

Funkcija `create_process()` deklarisana je na sledeći način (rezultat funkcije prenosi se u registru R0):

```
typedef unsigned int PID;
PID create_process (      // Returns: Process ID of the created
process
  char* program_file,     // Null-terminated string with program
file name
  unsigned int priority, // Default (initial) priority
  char** args             // Program arguments (array of strings)
);
```

Navedeni sistemski poziv vraća ID kreiranog procesa u registru R0, a očekuje argumente u sledećoj strukturi na koju treba da ukazuje R1:

```
struct create_process_struct {
  char* program_file;     // Null-terminated string with program
file name
  unsigned int priority; // Default (initial) priority
  char** args;            // Program arguments (array of strings)
};
```

```
PID create_process (char* p_file, unsigned int pri,
char** args) {
  create_process_struct p;
  p.program_file = p_file;
  p.priority = pri;
  p.args = args;
  asm {
    mov r1,sp;
    load r2,#p; // #p is the displacement of p below
the top of the stack
    add r1,r2;
    trap 25h;
  }
}
```

Zadatak – Januara 2015.

Neki operativni sistem ima mikrokernel arhitekturu. U kernelu ovog sistema implementirane su samo osnovne funkcije, uključujući upravljanje procesima i međuprocesnu komunikaciju, ali ne i fajl podsistem. Na raspolaganju su sledeći sistemski pozivi:

```
int getPID(); // Returns the current process ID
int send (int receiver_process_ID, const char* message, size_t msg_size);
int receive (int sender_process_ID, char* buffer, size_t buffer_size);
```

Prvi od ovih poziva vraća identifikator pozivajućeg procesa. Ostala dva poziva služe za međuprocesnu komunikaciju razmenom poruka. Komunikacija je sa direktnim i simetričnim imenovanjem, pri čemu se proces-sagovornik identifikuje njegovim identifikatorom. Poruka je proizvoljan sadržaj zadate dužine. Kod prijema poruke u sistemskom pozivu `receive` drugim i trećim argumentom zadaju se adresa i veličina bafera za prijem poruke. Slanje poruke je asinhrono, dok je prijem blokirajući. U slučaju bilo kakave greške, svi ovi pozivi vraćaju neku negativnu vrednost.

Za sve operacije sa fajl sistemom zadužen je poseban sistemski proces-osluškivač (*listener*) čiji je identifikator dat konstantom `FILE_LISTENER`. Korisnički proces zadaje operaciju sa fajlom slanjem poruke ovom procesu kroz opisani sistem međuprocesne komunikacije. Poruka je niz znakova koji ima format „komandne linije“ u kojoj prva reč označava operaciju, a zatim slede argumenti razdvojeni zarezima. Na primer, za operaciju otvaranja fajla, poruka treba da ima sledeći format:

```
open <pid>,<fname>,<mode>
```

Prvi argument je celobrojna vrednost identifikatora procesa koji zahteva operaciju, drugi je naziv fajla, a treći je decimalna predstava broja čija binarna vrednost sadrži flegove za prava pristupa koja proces zahteva do fajla. Kada izvrši ovu operaciju, proces zadužen za fajl sistem odgovara porukom u kojoj je decimalna predstava identifikatora otvorenog fajla (pozitivna decimalna vrednost) ili negativan kod greške.
Realizovati bibliotečnu sistemsku funkciju `fopen` za sistemski poziv otvaranja fajla. U slučaju uspeha, ona vraća celobrojni identifikator otvorenog fajla. U slučaju neke greške, ova funkcija treba da vrati neki negativan kod greške.

```
int fopen(const char* fname, int mode);
```

```
int fopen(const char* fname, int mode) {
       if(fname==0) return -1;
       int ret = 0;
       int pid = getPID();
       if (pid<0) return -1; // Exception
       // Prepare the ”command line”:
       const int buffersz = 1024;
       char buffer[buffersz];
       ret = snprintf(buffer,buffersz,”open %d,%s,%d”,pid,fname,mode);
       if (ret<0 || ret>=buffersz) return -1; // Exception
       // Send the request and get the reply:
       ret = send(FILE_LISTENER,buffer,strlen(buffer)+1);
       if (ret<0) return -1; // Exception
       ret = receive(FILE_LISTENER,buffer,buffersz);
       if (ret<0) return -1; // Exception
       if (sscanf(buffer,”%d”,&ret)<=0) return -1;
       return ret;
}
```

Zadatak – Januar 2013.

U nastavku je data ilustracija upotrebe objekta *critical section* iz dokumentacije za Win32 API. Na jeziku C++ implementirati klasu `Mutex` koja obezbeđuje objektno orijentisani „omotač“ oko ovih sistemskih poziva i realizuje apstrakciju binarnog semafora za međusobno isključenje niti.

```
// Global variable
CRITICAL_SECTION CriticalSection;

int main( void )
{
  ...
  // Initialize the critical section one time only.
  if (!InitializeCriticalSectionAndSpinCount(&CriticalSection,
      0x00000400) )
    return;
  ...
  // Release resources used by the critical section object.
  DeleteCriticalSection(&CriticalSection);
}
```

```
DWORD WINAPI ThreadProc( LPVOID lpParameter )
{
   ...
   // Request ownership of the critical section.
   EnterCriticalSection(&CriticalSection);
   // Access the shared resource.
   // Release ownership of the critical section.
   LeaveCriticalSection(&CriticalSection);
   ...
   return 1;
}
```

```
class Mutex {
public:
  Mutex ()
  {InitializeCriticalSectionAndSpinCount(&criticalSection,0x00000400);}

  ~Mutex() { DeleteCriticalSection(&criticalSection); }

  void enter () { EnterCriticalSection(&criticalSection); }

  void exit () { LeaveCriticalSection(&criticalSection); }

private:
  CRITICAL_SECTION criticalSection;
};
```