Upravljanje memorijom

Zad 1. Jun 2005.

Date su poruke o greškama (označene sa X i Y) od strane linkera i neki propusti u dizajnu programa (označeni 1-5). Za svaki od propusta 1-5 navesti poruku koju linker može izdati – napisati oznake parova koji odgovaraju (npr. 1-X, 2-Y, itd.). Pretpostavlja se da su fajlovi prevedeni bez grešaka.

X. "Undefined symbol ... referenced in files ..."
Y. "Multiple definitions of symbol ... in files ..."

1. Definicija (telo) funkcije (koja nije `inline`) napisano u fajlu-zaglavlju (.h).
2. U listi fajlova koje treba povezati nije navedena korišćena biblioteka.
3. Program ne sadrži funkciju `main()`.
4. Identifikator promenljive u `extern` deklaraciji navedenoj u fajlu-zaglavlju je pogrešno napisan (drugačije od definicije iste promenljive u .c fajlu).
5. Deklaracija promenljive tipa `int` navedena u fajlu-zaglavlju (.h) bez ključne reči `extern`.

Odgovor: 1 - ____, 2 - ____, 3 - ____, 4 - ____, 5 - ____.

Odgovor: 1 - Y, 2 - X, 3 - X, 4 - X, 5 – Y.

Zad 2. April 2006.

Neki C program sastoji se samo iz dva fajla sa datim sadržajem:

```
// a.c                          // b.c
float base = 2.0;               float log(float);
float log(float);               extern float base;
float ln(float);

float log(float x) {            float f(float x) {
  return ln(x)/ln(base);          return log(x*base)/log(base);
}                               }
```

Odgovoriti na sledeća pitanja:

1) Koliko nerazrešenih adresnih polja instrukcija prevodilac ostavlja u fajlu `a.obj`? _____

2) Koliko nerazrešenih adresnih polja instrukcija prevodilac ostavlja u fajlu `b.obj`? _____

3) Koje simbole izvozi fajl `a.obj`? _____

4) Koje simbole uvozi fajl `a.obj`? _____

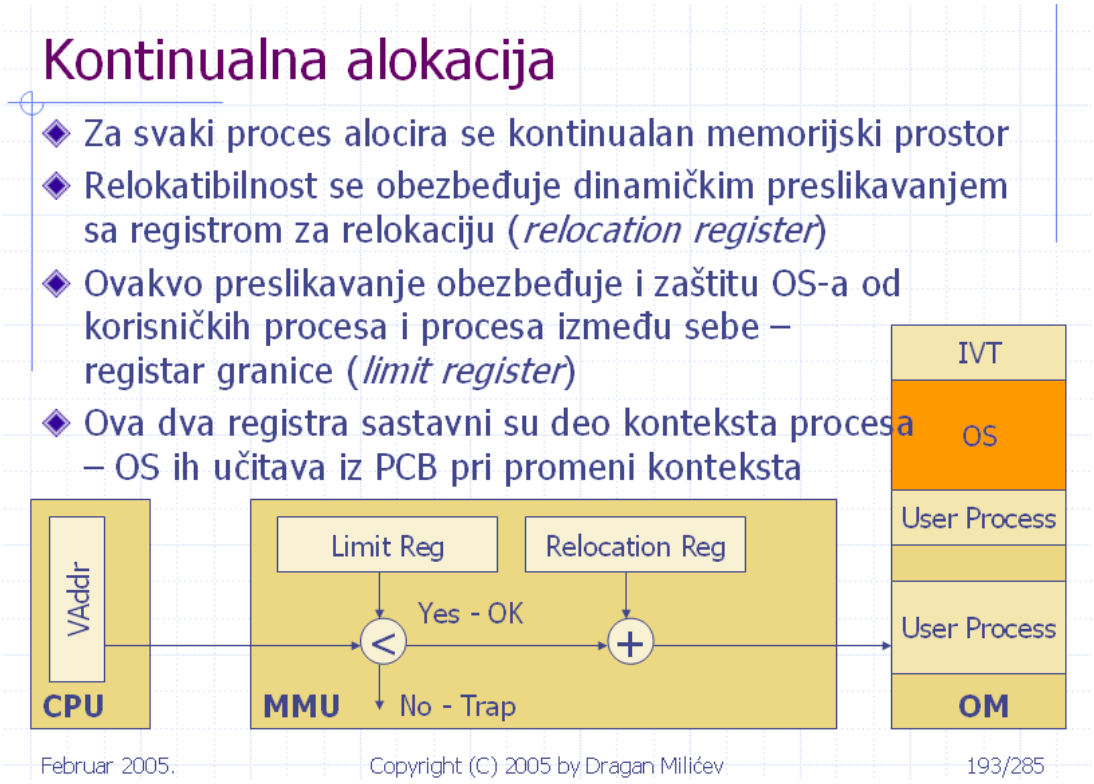
5) Koje simbole izvozi fajl `b.obj`? _____

6) Koje simbole uvozi fajl `b.obj`? _____

7) Koliko grešaka prijavljuje linker prilikom povezivanja `a.obj` i `b.obj` u `p.exe`? _____

Rešenje:

1) 2
2) 4 ili 3, u zavisnosti da li je optimizovao dohvatanje vrednosti `base` u neki registar
3) `base`, `log`
4) `ln`
5) `f`
6) `base, log`
7) 2



Zad 3. Oktobar 2005.

Neki OS koristi kontinualnu alokaciju operativne memorije. Data je deklaracija strukture podataka koja predstavlja zaglavlje svakog slobodnog fragmenta memorije. Ova zaglavlja čine dvostruko ulančanu listu slobodnih fragmenata i upisuju se na sam početak svakog slobodnog fragmenta memorije. Napisati telo funkcije `getFirstFitFragment()` koja treba da pronađe fragment slobodne memorije veličine date argumentom po *first-fit* algoritmu i vrati njegovu adresu. Preostali deo fragmenta treba da postane novi (manji) fragment u listi slobodnih.

```
typedef unsigned int size_t;
struct FreeFragment {
  size_t size; // Veličina fragmenta
  FreeFragment* prev; // Prethodni u listi
  FreeFragment* next; // Sledeći u listi
}
FreeFragment* head; // Glava liste slobodnih fragmenata
void* getFirstFitFragment(size_t);
```

Rešenje:

```
void* getFirstFitFragment (size_t sz) {
  for (FreeFragment* cur = head; cur!=0; cur=cur->next) {
    if (cur->size<sz) continue;

     // set pointer to the first byte of the remaining free
     //fragment
    FreeFragment* newfrgm = (FreeFragment*)((char*)cur+sz);
    if (cur->prev) cur->prev->next = newfrgm;
    if (cur->next) cur->next->prev = newfrgm;
    newfrgm->prev = cur->prev;
    newfrgm->next = cur->next;
    newfrgm->size = cur->size-sz;
    return cur;
  }
  return 0;
}
```

Napomena: Pretpostavka je da se zaglavlje fragmenta uvek može upisati u preostali deo, u suprotnom trebalo bi i to ispitati.

Zad 4. Januar 2006.

Neki OS koristi kontinualnu alokaciju operativne memorije. Data je deklaracija strukture podataka koja predstavlja zaglavlje svakog slobodnog fragmenta memorije. Ova zaglavlja čine dvostruko ulančanu listu slobodnih fragmenata i upisuju se na sam početak svakog slobodnog fragmenta memorije. Napisati telo funkcije `getBestFitFragment()` koja treba da pronađe fragment slobodne memorije veličine date argumentom po *best-fit* algoritmu i vrati njegovu adresu. Preostali deo fragmenta treba da postane novi (manji) fragment u listi slobodnih.

```
typedef unsigned int size_t;
struct FreeFragment {
  size_t size; // Veličina fragmenta
  FreeFragment* prev; // Prethodni u listi
  FreeFragment* next; // Sledeći u listi
}
FreeFragment* head; // Glava liste slobodnih fragmenata
void* getBestFitFragment(size_t);
```

Rešenje:

```
void* getBestFitFragment (size_t sz) {
  FreeFragment* bestFit = 0;
  size_t smallestFragment = MAXSIZE;
  for (FreeFragment* cur = head; cur!=0; cur=cur->next) {
    if (cur->size<sz) continue;
    if (cur->size<smallestFragment) {
      smallestFragment = cur->size;
      bestFit = cur;
    }
  }
  if (bestFit==0) return 0;
  FreeFragment* newfrgm =
          (FreeFragment*)((char*)bestFit+sz);
  if (bestFit->prev) bestFit->prev->next = newfrgm;
  if (bestFit->next) bestFit->next->prev = newfrgm;
  FreeFragment* p = bestFit->prev;
  FreeFragment* n = bestFit->next;
  size_t s = bestFit->size-sz;
  newfrgm->prev = p;
  newfrgm->next = n;
  newfrgm->size = s;
  return bestFit;
}
```

Zadatak 5.

U nekom sistemu primenjuje se kontinualna alokacija operativne memorije. Deo definicije strukture PCB je sledeći:

```
struct PCB {
  ...
  void* memLocation;  // Current place in memory
  size_t memSize;  // Size of memory space
};
```

Implementirati operaciju:

```
void relocate(PCB* process, void* newPlace);
```

kojom sistem premešta memorijski prostor procesa sa tekućeg na novozadato (već alocirano) mesto u memoriji.

U datom sistemu, operacija

```
void free (void* addr, size_t size);
```

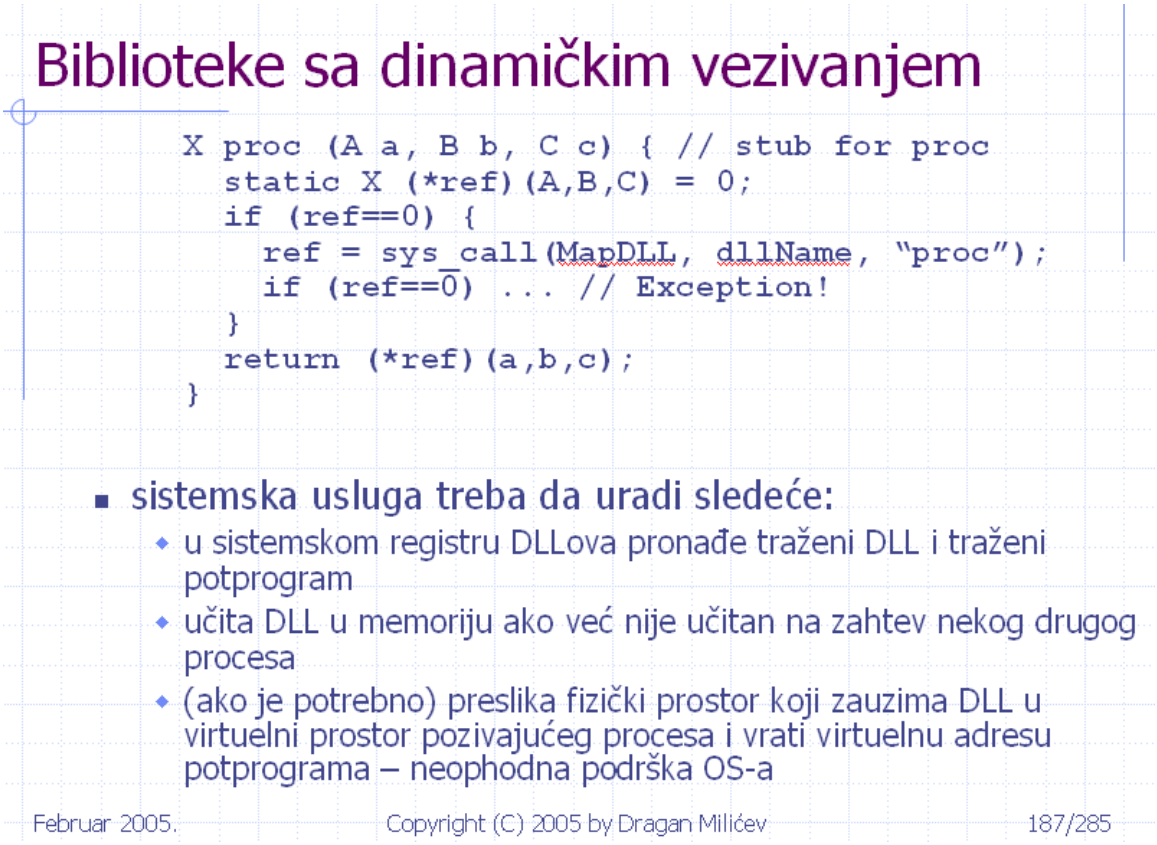
dealocira (proglašava slobodnim) memorijski prostor veličine size počev od adrese addr.

Pomoć: Standardna bibliotečna funkcija memcpy ima sledeću deklaraciju:

```
void* memcpy (void* destination, const void* source, size_t
size);
```

Rešenje:

```
void relocate(PCB* p, void* newPlace) {
  if (p==0) return; // Exception!
  if (newPlace==p->memLocation || p->memSize==0) return; //
Nothing to do
  memcpy(newPlace,p->memLocation,p->memSize); // Move
memory contents
  free(p->memLocation,p->memSize); // Free the old memory
space
  p->memLocation=newPlace;  // Move relocation register
}
```



Zad 6. Oktobar 2005.

Neki OS koristi straničnu organizaciju virtuelne memorije sa mogućnošću deljenja stranica, a podržava i biblioteke sa dinamičkim vezivanjem (DLL). Izvršen je sledeći eksperiment sa jednim malim test-programom koji samo poziva neki potprogram iz datog DLL-a, izmeri vreme tog poziva, ispiše ga na izlaz i završi. Na tek pokrenutom računaru sa podignutim OS-om pokrenut je samo proces nad ovim programom i završio je izvršavanje, ispisavši vreme. Odmah potom je taj program ponovo pokrenut i ispisao je znatno kraće vreme.

(*i*)(5) Zašto je rezultat ovakav? Precizno objasniti.

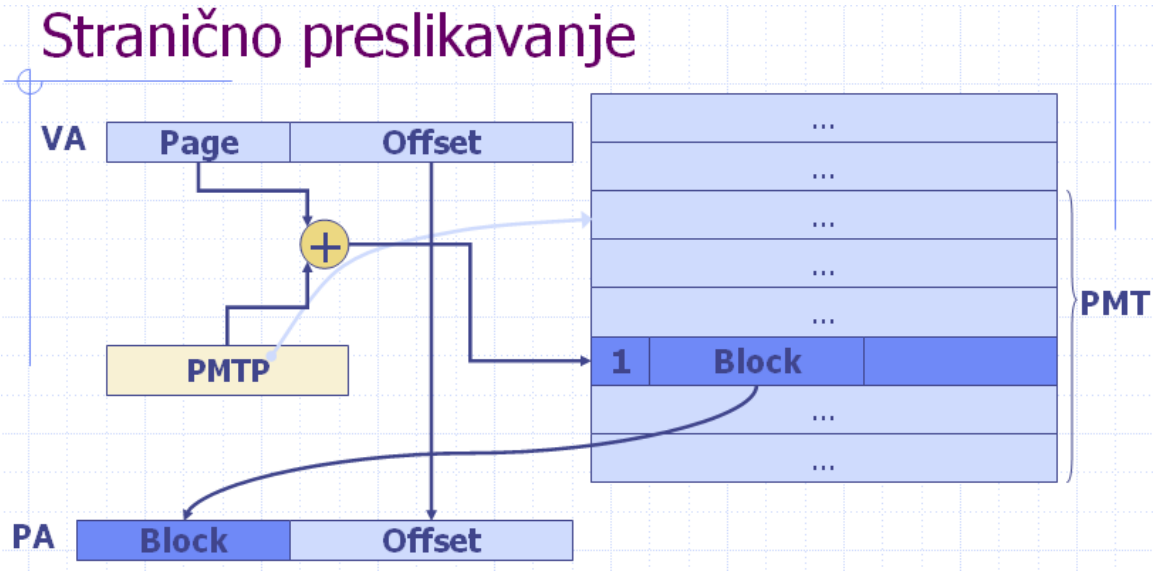
(*ii*)(5) Isti eksperiment je ponovljen na istom računaru i istom OS-u, samo na malo izmenjenoj verziji tog OS-a (izmena se odnosi samo na upotrebu DLL-ova). Ovaj put je program ispisivao vrlo približno isto vreme, svaki put kada je pokrenut. Koja izmena u OS-u je izvršena? Precizno objasniti kako je ova izmena implementirana.

Rešenje:

(i) Kada prvi put neki proces pozove potprogram iz nekog DLL-a, ukoliko taj DLL nije učitan u memoriju jer ga pre toga nije koristio ni jedan drugi proces, OS pokreće učitavanje tog DLL-a. Po završetku procesa, taj DLL može da ostane u memoriji, kako bi drugi procesi mogli brže da mu pristupaju čak i prvi put i preslikaju njegovu fizičku lokaciju u svoj virtuelni memorijski prostor. Na taj način je svaki sledeći pristup do tog DLL-a mnogo brži, jer se on ne učitava sa diska. Upravo to se dogodilo u opisanom eksperimentu.

(ii) Sada očigledno OS izbacuje DLL iz memorije po završetku procesa koji ga je koristio. To se može implementirati ili tako što DLL nije deljen, tj. postoji posebna kopija koja pripada samo virtuelnom adresnom prostoru svakog procesa koji ga je zahtevao, što je manje efikasno, ili je DLL i dalje deljen, ali ga OS izbacuje kada se završi poslednji proces koji je pristupao tom DLL-u, što je nešto teže implementirati, ali je efikasnije. Ovo drugo se može implementirati klasičnom upotrebom brojača referenciranja na deljeni objekat, u ovom slučaju na DLL.

Vitruelna memorija



Zad 7. Jun 2005.

Virtuelni adresni prostor sistema je 1MB, adresibilna jedinica je bajt, a virtuelni adresni prostor je organizovan stranično, sa stranicom veličine 4KB. Fizički adresni prostor iznosi 4GB. Prostor za zamenu (*swap*) na disku može da sadrži najviše 1M blokova koji se koriste za čuvanje zamenjenih stranica i kojima se direktno pristupa (ne kroz fajl-sistem). Pristup stranicama se kontroliše pomoću tri bita zaštite koji se označavaju sa R (*read*), W (*write*) i E (*execute*). Kada se stranica nalazi u memoriji, prostor koji je zauzimala na disku se oslobađa za zamenu drugih stranica. Odgovoriti na sledeća pitanja i kratko, ali precizno obrazložiti odgovor:

- Ako se deskriptori stranica u tabeli preslikavanja stranica (PMT) maksimalno kompaktno smeštaju, kolika je veličina PMT za jedan proces?

  Odgovor: _____

  Račun:

- Ako se evidencija o slobodnim okvirima (blokovima) fizičke memorije čuva u bit-vektoru i maksimalno kompaktno smešta u memoriji, kolika je veličina prostora za smeštanje ovog vektora?

  Odgovor: _____

Rešenje:

- Odgovor: 768B
  Račun:
  Virtuelna adresa (VA), 20 bita: Page(8):Offset(12)
  Fizička adresa (PA), 32 bita: Frame(20):Offset(12)
  Deskriptor stranice: Da li je u memoriji (1) : RWE(3) : Frame(20) ili Block(20)

Odatle sledi da je deskriptor stranice veličine 24 bita, odnosno 3 bajta. PMT ima po jedan ulaz za svaku stranicu, što znači $2^8 = 256$ ulaza po 3 bajta, odnosno 768B.

- Odgovor: 128KB.
  Račun: Broj blokova u fizičkoj memoriji – broj bita u bit-vektoru: 2^{20}. Veličina bit-vektora u bajtovima: $2^{20}/2^3 = 2^{17} = 128K$.

Zad 8. Septembar 2005.

Virtuelna memorija nekog računara organizovana je stranično (*paging*). Veličina logičkog (virtuelnog) adresnog prostora je 2 MB, adresibilna jedinica je 16-bitna reč, a veličina stranice je 128 KB. Veličina fizičkog adresnog prostora je 32 MB. Operativni sistem učitava stranice na zahtev (*demand paging*), tako što se stranica učitava u prvi slobodni blok (okvir, *frame*) fizičke memorije kada joj se pristupi. U početnom trenutku, slobodni blokovi fizičke memorije su blokovi počev od 10h zaključno sa 1Fh. Neki proces generiše sledeću sekvencu logičkih adresa tokom svog izvršavanja (sve vrednosti su heksadecimalne):

20F00, 20F02, 20F04, 822F0, 822F2, 222F0, 222F2, 222F4, 422F0, 422F2, 302F0, 302F2

Prikazati izgled cele tabele preslikavanja stranica (PMT) za ovaj proces posle izvršavanja ove sekvence. Za svaki ulaz u PMT prikazati indikator prisutnosti stranice u fizičkoj memoriji (0 ili 1) i broj bloka u fizičkoj memoriji u koji se stranica preslikava, ukoliko je stranica učitana; ukoliko nije, prikazati samo ovaj indikator.

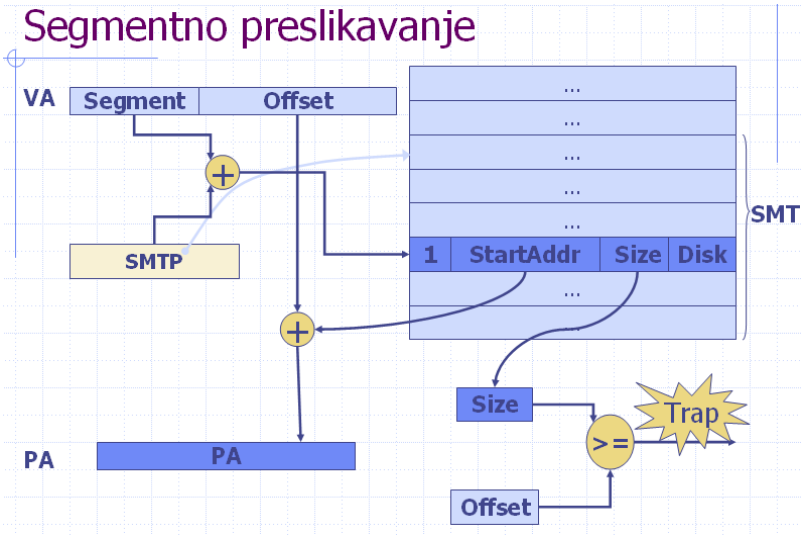
Rešenje:

VA: Page(4):Offset(16)
PA: Block(8):Offset(16)

Sekvenca stranica koje se traže: 2, 2, 2, 8, 8, 2, 2, 2, 4, 4, 3, 3

PMT na kraju sekvence:

Entry	Flag	Block
0	0	
1	0	
2	1	10h
3	1	13h
4	1	12h
5	0	
6	0	
7	0	
8	1	11h
9	0	
A	0	
B	0	
C	0	
D	0	
E	0	
F	0	



Zad 9. Segmentna organizacija memorije

Virtuelni adresni prostor sistema je 1GB, adresibilna jedinica je bajt, a virtuelni adresni prostor je organizovan segmentno. Svaki proces može imati najviše 64 segmenta. Fizički adresni prostor je veličine 512MB. Prostor za zamenu (*swap space*) na disku je duplo veći od fizičke memorije i koristi se za čuvanje zamenjenih segmenata kojima se direktno pristupa na disku (na „presnoj“ particiji, ne kroz fajl-sistem). Pristup segmentima se kontroliše pomoću tri bita zaštite koji se označavaju sa R (*read*), W (*write*) i E (*execute*). Kada se segment nalazi u memoriji, prostor koji je zauzimala na disku se oslobađa za zamenu drugih segmenata. Odgovoriti na sledeće pitanje i kratko, ali precizno obrazložiti odgovor: ako se deskriptori segmenata u tabeli preslikavanja segmenata (SMT) maksimalno kompaktno smeštaju, kolika je veličina SMT za jedan proces?

Rešenje:

Virtuelna adresa (VA), 30 bita: Segment (6) : Offset(24)
Fizička adresa (PA), 29 bita
Swap 1GB => Disk adr. (30)
Deskriptor stranice: Da li je u memoriji (1) : RWE(3) : Size(24) : Segment adr.(29) ili Disk Adr. (30)
Odatle sledi da je deskriptor segmenta veličine 58 bita, SMT ima po jedan ulaz za svaki segment, što znači 64 ulaza. Tako da je veličina SMTa 58*64/8 B = 464B

Zadatak 10. Januar 2006.

Zašto preklopi (*overlays*) ne mogu da se koriste ako program ima više niti koje obezbeđuje operativni sistem? Precizno objasniti.

Rešenje:

Preklopi (*overlays*) se realizuju bez podrške i bilo kakve interakcije sa operativnim sistemom (osim same podrške dinamičkom učitavanju), tako što na ulasku u određeni potprogram koji je alociran u neki preklop, prevodilac generiše kod koji proverava da li je taj kod učitan u preklop i učitava ga ako nije. Ako program koristi niti koje obezbeđuje OS, sistem održavanja preklopa (u odgovornosti programa) i promena konteksta niti (u odgovornosti OS-a) nemaju interakcije i „ne znaju jedan za drugog“. Zato se može dogoditi sledeći konfliktni scenario:

- jedna nit uđe u izvršavanje nekog potprograma P koji pripada jednom preklopu, proveri i po potrebi učita kod za taj potprogram u preklop X
- OS izvrši promenu konteksta niti i pokrene drugu nit; ova druga nit ulazi u potprogram Q koji pripada drugom preklopu, a koji nije učitan
- sistem preklopa učitava preklop kome pripada Q na isto mesto X
- ponovo dođe do preuzimanja i prva nit nastavi izvršavanje od adrese na kojoj očekuje kod za potprogram P, ali se tu sada nalazi kod za Q.

Zad 11. Septembar 2005.

Tabela prikazuje uporedne karakteristike tri tehnike za deljenje operativne memorije: dinamičko učitavanje (*dynamic loading*), preklapanje (*overlaying*) i segmentna organizacija (*segmentation*). U prazna polja tabele upisati odgovor "Da" ili "Ne".

Rešenje:

Karakteristika	**Dinamičko učitavanje**	**Preklapanje**	**Segmentiranje**
Hardver učestvuje u dinamičkom preslikavanju adresa?	Ne	Ne	Da
OS obezbeđuje strukture za dinamičko preslikavanje adresa?	Ne	Ne	Da
Više simbola (npr. potprograma) se može preslikati u istu logičku adresu u adresnom prostoru programa?	Ne	Da	Ne
Više logičkih celina logičkog adresnog prostora se može preslikati u isti deo fizičke memorije tokom izvršavanja programa?	Ne	Da	Da
Da bi se proces izvršavao, ceo njegov logički adresni prostor mora biti učitan i raspoređen po fizičkoj memoriji?	Ne	Ne	Ne

Zad 12. April 2006.

Neki OS koristi straničnu organizaciju memorije sa tehnikom *copy-on-write*. Deskriptor stranice izgleda ovako:

```
struct PageDescr {
  int isInMemory; // Is this page currently in OM?
  int isReadOnly; // Is this page currently read-only?
  int isWriteEnabled; // Is this page allowed for writing?
  int isShared; // Is this page shared with other
                // processes?
  int block; // Block number
};
```

Prilikom preslikavanja adresa i provere dozvoljenosti upisa u stranicu, hardver proverava samo indikator `isReadOnly`. Indikator `isWriteEnabled` ukazuje na to da li je stranica inicijalno označena kao dozvoljena za upis (`isWriteEnabled=1`) ili nije.

Na raspolaganju su i sledeće funkcije iz jezgra ovog operativnog sistema:

```
int findFreeBlock(); // Finds and returns a free block in OM
void copyBlock(int from, int to); // Copies the entire block 'from' to
'to'
void killRunningProc(); // Kills the running process due to an error
```

Napisati kod funkcije `pageWriteTrap(PageDescr*)` koja se izvršava na interni prekid (*trap*) koji generiše hardver kada se pokuša upis u stranicu koja ima `isReadOnly=1`. U slučaju da je ovaj upis neregularan, treba samo ugasiti tekući proces. Inače se radi o upisu u stranicu koja se deli sa drugim procesima i treba primeniti *copy-on-write*. Potrebno je ažurirati samo dati deskriptor adresirane stranice, ne treba pisati kod za ažuriranje deskriptora drugih stranica koje dele isti blok. Pretpostaviti da `findFreeBlock()` uvek pronalazi i vraća alocirani blok.

Rešenje:

```
void pageWriteTrap (PageDecr* pg) {
  if (!pg->isWriteEnabled) { killRunningProcess(); return; }
  // Copy-on-write:
  int newBlock = findFreeBlock();
  copyBlock(pg->block,newBlock);
  pg->isReadOnly = !pg->isWriteEnabled; // should be 0 always
  pg->isShared = 0;
  pg->block = newBlock;
}
```

Napomena: Ukoliko bi trebalo voditi računa i o deskriptorima drugih stranica koje dele isti blok, kako bi to moglo da se implementira pomoću polja `isShared`?