



# Operativni sistemi 2013

## Sistemska programiranje Upravljanje memorijom

**Katedra za računarstvo  
Elektronski fakultet u Nišu**

**Doc. dr Dragan Stojanović  
mr Aleksandar Stanimirović  
mr Bratislav Predić**



# Sadržaj



- Struktura memorije dodeljene procesu
- Deljena memorija



# Sadržaj



- Struktura memorije dodeljene procesu
- Deljena memorija



# Struktura memorije dodeljene procesu



## Pojam

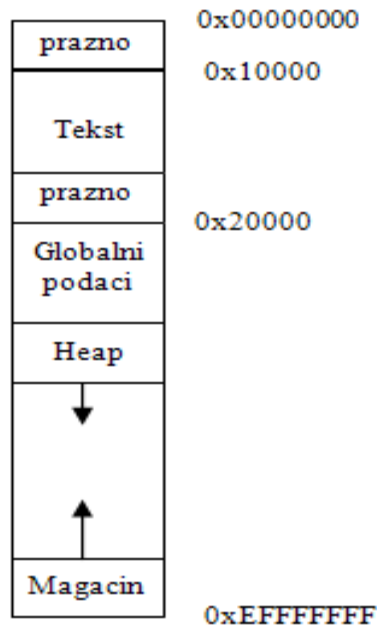
- **Programski prevodilac** vodi računa o zauzimanju memorije za potrebe promenljivih deklariranih u programu.
- Promenljive se mogu koristiti odmah nakon deklaracije bez ikakvih **pripremnih radnji**.
- **Globalne promenljive** se smeštaju u **segment za podatke** a **lokalne promenljive** u **segment magacina**. Ovako deklarirane promenljive zauzimaju memorijski prostor tokom čitavog izvršavanja programa.
- Moguće je vršiti i **dinamičko zauzimanje i oslobađanje** memorije.
- Dinamičko zauzimanje memorije se vrši u segmentu koji se zove **heap**.



# Struktura memorije dodeljene procesu



## Memorijski segmenti





# Struktura memorije dodeljene procesu



## Memorijski segmenti

- Operativnu memoriju računara možemo posmatrati kao niz elemenata čije indeksiranje počinje od nule.
- Pokazivači u C programu predstavljaju indekse ovog niza.
- **Nije moguće pristupiti svakoj memorijskoj lokaciji.** Pokušaj čitanja memorijske lokacije sa adresom 0 bi doveo do greške (**segmentation violation**).
- Postoje četiri segmenta memorije kojima je dozvoljen pristup:
  1. **Tekst (kod) segment** – segment u kome se nalaze instrukcije programa.
  2. **Globalni podaci** - segment u kome se nalaze globalne promenjive koje definiše program.
  3. **Heap** - memorijske lokacije za dinamički kreirane promenljive koje vraća malloc naredba.
  4. **Magacin** - segment u kome se nalaze lokalne promenjive, argumenti poziva funkcija i povratne vrednosti funkcija.



# Struktura memorije dodeljene procesu



## Dinamičko zauzimanje i oslobađanje memorije

```
#include <stdlib.h>
char * malloc (unsigned int size);
void free (char * ptr);
```

## Semantika

- Sistemski poziv **malloc** **zauzima memoriju** koja je dovoljna da se u nju smesti objekat čija je veličina size bajtova.
- U slučaju uspeha sistemski poziv vraća **pokazivač na početak zauzete memorije** a u slučaju greške vraća **NULL**.
- Zauzeta memorija koja više nije neophodna oslobađa se sistemskim pozivom **free**.
- Postoje i brže verzije ovih funkcija i one su deklarirane u zaglavlju **<malloc.h>**.
- Da bi se ove funkcije koristile program se prevodi sa: **gcc -lmalloc**.



# Struktura memorije dodeljene procesu



## Dinamičko zauzimanje i oslobađanje memorije

```
#include <stdlib.h>
char * calloc (unsigned int elem, unsigned int elsize);
char * realloc (char * ptr, unsigned int size);
```

## Semantika

- Sistemski poziv **calloc** **rezerviše memoriju** za niz elemenata čija je dimenzija specificirana.
- Sistemski poziv **realloc** **menja veličinu** prethodno rezervisanog segmenta memorije.





# Struktura memorije dodeljene procesu



```
struct osoba
```

```
{  
  char ime[10];  
  char prezime[15];  
  int godine;  
}
```

Deklaracija strukture.

```
struct osoba * c;
```

```
c = (struct osoba *)malloc(sizeof(struct osoba));
```

```
if (c == 0)
```

```
{  
  printf("Doslo je do greske");  
  exit(1);  
}
```

Dinamičko zauzimanje memorije.

```
c->ime = "Janko";
```

```
c->prezime = "Janković";
```

```
c->godine = 10;
```

Korišćenje zauzete memorije.

```
free((char*)c);
```

Oslobađanje memorije.



# Signali

## Analiza i izmena sadržaja memorijskih lokacija

```
#include <memory.h>
void *memcpy(void *dest, const void *src, size_t n);
void *memccpy(void *dest, const void *src, int c, size_t n);
void *memchr(const void *s, int c, size_t n);
int memcmp(const void *s1, const void *s2, size_t n);
void *memset(void *s, int c, size_t n);
```



# Sadržaj



- Struktura memorije dodeljene procesu
- Deljena memorija



# Deljena memorija

## Pojam

- **Deljena memorija** je jedan od mehanizama koji omogućava komunikaciju između različitih procesa.
- Deljena memorija je oblast operativne memorije koja je **dodeljena adresnim prostorima oba procesa** i oba procesa mogu da čitaju i upisaju u taj deo memorije.
- Prilikom pristupanja deljenoj memoriji od strane većeg broja procesa **ne postoji nikakav mehanizam zaštite**. U takvim situacijama može doći do pojave **problema trke (race conditions)**.
- Prilikom pristupanja deljenoj memoriji od strane većeg broja procesa potrebno je iskoristiti neki mehanizam za međusobno isključenje i sinhronizaciju.



# Deljena memorija

## Kreiranje deljene memorije

```
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/shm.h>
int shmget(key_t key, int size, int flg);
```

## Semantika

- Sistemski poziv koji kreira novi segment deljene memorije.
- Prvi argument **key** predstavlja **jedinstveni identifikator** segmenta deljene memorije na nivou čitavog sistema. **Mora biti poznat svim procesima koji žele da koriste određeni segment deljene memorije.**



# Deljena memorija

## Semantika

- Vrednost drugog argumenta **size** definiše veličinu segmenta deljene memorije koji se kreira.
- Vrednost trećeg argumenta **flag** se definiše kao **rezultat OR operacije** nad različitim vrednostima i određuje :
  - ▶ prava pristupa segmentu deljene memorije (koristićemo vrednost 0666 koja svim korisnicima dodeljuje sve privilegije nad deljenom memorijom)
  - ▶ mod kreiranja segmenta deljene memorije. Najčešća vrednost je:
    - **IPC\_CREAT** – sistemski poziv kreira segment deljene memorije ukoliko on već ne postoji u sistemu.
- Sistemski poziv vraća **celobrojni identifikator (referencu) segmenta deljene memorije** a u slučaju greške vraća (-1). Dobijeni identifikator (referenca) je važeći samo kod procesa koji je izvršio sistemski poziv i kod njegove dece.
- Ukoliko u sistemu ne postoji segment deljene memorije sa zadatim identifikatorom a specificiran je flag **IPC\_CREAT** kreira se novi segment deljene memorije.
- Ukoliko u sistemu već postoji segment deljene memorije sa zadatim identifikatorom ne kreira se novi segment već se samo vraća referenca na postojeći segment deljene memorije.



# Deljena memorija

## Mapiranje segmenta deljene memorije

```
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/shm.h>
void * shmat(int shm_id, char * ptr, int flag);
```

## Semantika

- Sistemski poziv **mapira segment deljene memorije u adresni prostor procesa**.
- Prvi argument **shm\_id** predstavlja identifikator (referencu) segmenta deljene memorije koji je dobijen pozivom funkcije **shmget**.
- Drugi argument **ptr** u našem slučaju uvek ima vrednost **NULL**.
- Treći argument definiše način korišćenja segmenta deljene memorije:
  - ▶ **SHM\_RDONLY** – deljenu memoriju je moguće samo čitati
  - ▶ 0 – moguća je i i zmena sadržaja deljene memorije bajtovima.
- U slučaju uspeha sistemski poziv vraća **pokazivač na početak adresnog prostora** u koji je mapirana deljena memorija. U slučaju greške sistemski poziv vraća NULL.



# Deljena memorija

## Uklanjanje segmenta deljene memorije

```
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/shm.h>
void * shmdt(const void * shmaddr);
```

## Semantika

- Sistemski poziv **uklanja segment deljene memorije iz adresnog prostora procesa**.
- Prvi argument **shmaddr** predstavlja pokazivač na početak adresnog prostora u koji je mapiran segment deljene memorije (adresa koju vraća funkcija shmat).





# Deljena memorija

## Kontrola reda poruka

```
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/msg.h>
int shmctl (int shmid, int cmd, struct shmid_ds buf);
```

## Semantika

- Prvi argument **shmid** predstavlja identifikator (referencu) segmenta deljene memorije koji je dobijen pozivom funkcije **shmget**.
- Drugi argument **cmd** definiše operaciju koju treba izvršiti nad segmentom deljene memorije. Za brisanje deljene memorije se koristi operacija **IPC\_RMID**.



# Deljena memorija

```
//INIT.H
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define MUTEX_KEY 10101
#define EMPTY_KEY 10102
#define FULL_KEY 10103
#define SHARED_MEM_KEY 10103
#define N 10

//INIT.C
#include "init.h"
int main()
{
    int mutexid, emptyid, fullid, shmemid;
    union semun semopts;

    //KREIRANJE SEMAFORA
    mutexid = semget((key_t)MUTEX_KEY, 1, 0666 |
IPC_CREAT);
    emptyid = semget((key_t)EMPTY_KEY, 1, 0666 |
IPC_CREAT);
    fullid = semget((key_t)FULL_KEY, 1, 0666 | IPC_CREAT);
```

```
//INICIJALIZACIJA SEMAFORA
    semopts.val = 1;
    semctl(mutexid, 0, SETVAL, semopts);
    semopts.val = N;
    semctl(emptyid, 0, SETVAL, semopts);
    semopts.val = 0;
    semctl(mutexid, 0, SETVAL, semopts);

//KREIRANJE SEGMENTA DELJENE MEMORIJE
    shmemid = shmget(SHARED_MEM_KEY,
N * sizeof(char), IPC_CREAT | 0666);

//POKRETANJE PROIZVODJACA
    if (fork() == 0)
        execl("proizvodjac", "proizvodjac", NULLL);
//POKRETANJE POTROSACA
    if (fork() == 0)
        execl("potrosac", "potrosac", NULLL);

//BRISANJE SEMAFORA I DELJENE MEMORIJE
    semctl(mutexid, 0, IPC_RMID, 0);
    semctl(emptyid, 0, IPC_RMID, 0);
    semctl(fullid, 0, IPC_RMID, 0);
    shmctl(shmemid, 0, IPC_RMID, 0);
}
```



# Deljena memorija

```
//PROIZVODJAC.C
#include "init.h"
int main()
{
    int mutexid, emptyid, fullid, shmемid;
    int current = 0;
    char * shmptr
    struct sembuf sem_lock = { 0, -1, NULL};
    struct sembuf sem_unlock = { 0, 1, NULL};

    //PRIBAVLJANJE REFERENCE SEMAFORA
    mutexid = semget((key_t)MUTEX_KEY, 1, 0666);
    emptyid = semget((key_t)EMPTY_KEY, 1, 0666);
    fullid = semget((key_t)FULL_KEY, 1, 0666);

    //PRIBAVLJANJE REFERENCE DELJENE MEMORIJE
    shmемid = shmget(SHARED_MEM_KEY,
                     N * sizeof(char), 0666);

    //MAPIRANJE DELJENE MEMORIJE
    shmptr = (char*)shmat(shm_id, NULL, 0);
```

```
//PROIZVOĐAČ GENERIŠE PODATKE
// I UPIŠUJE IH U BAFER
while(TRUE)
{
    semop(emptyid, &sem_lock, 1);
    semop(mutexid, &sem_lock, 1);

    //UPIŠIVANJE U BAFER
    shmptr[current] = rand() % 100;
    current = (current + 1) % N;

    semop(mutexid, &sem_unlock, 1);
    semop(fullid, &sem_unlock, 1);
}

//IZBACIVANJE DELJENE MEMORIJE
//IZ ADRESNOG PROSTORA
shmdt(shmptr);
}
```



# Deljena memorija

```
//POTROSAC.C
#include "init.h"
int main()
{
    int mutexid, emptyid, fullid, shmamid;
    int current = 0;
    char * shmptr;
    struct sembuf sem_lock = { 0, -1, NULL};
    struct sembuf sem_unlock = { 0, 1, NULL};

    //PRIBAVLJANJE REFERENCE SEMAFORA
    mutexid = semget((key_t)MUTEX_KEY, 1, 0666);
    emptyid = semget((key_t)EMPTY_KEY, 1, 0666);
    fullid = semget((key_t)FULL_KEY, 1, 0666);

    //PRIBAVLJANJE REFERENCE DELJIVE MEMORIJE
    shmamid = shmget(SHARED_MEM_KEY,
                    N * sizeof(char), 0666);

    //MAPIRANJE DELJIVE MEMORIJE
    shmptr = (char*)shmat(shm_id, NULL, 0);
```

```
//POTROŠAČ ČITA PODATKE IZ BAFERA
while(TRUE)
{
    semop(fullid, &sem_lock, 1);
    semop(mutexid, &sem_lock, 1);

    //ČITAVANJE IZ BAFERA
    printf("%d\n", shmptr[current]);
    current = (current + 1) % N;

    semop(mutexid, &sem_unlock, 1);
    semop(emptyid, &sem_unlock, 1);
}
//IZBACIVANJE DELJENE MEMORIJE
//IZ ADRESNOG PROSTORA
shmdt(shmptr);
}
```