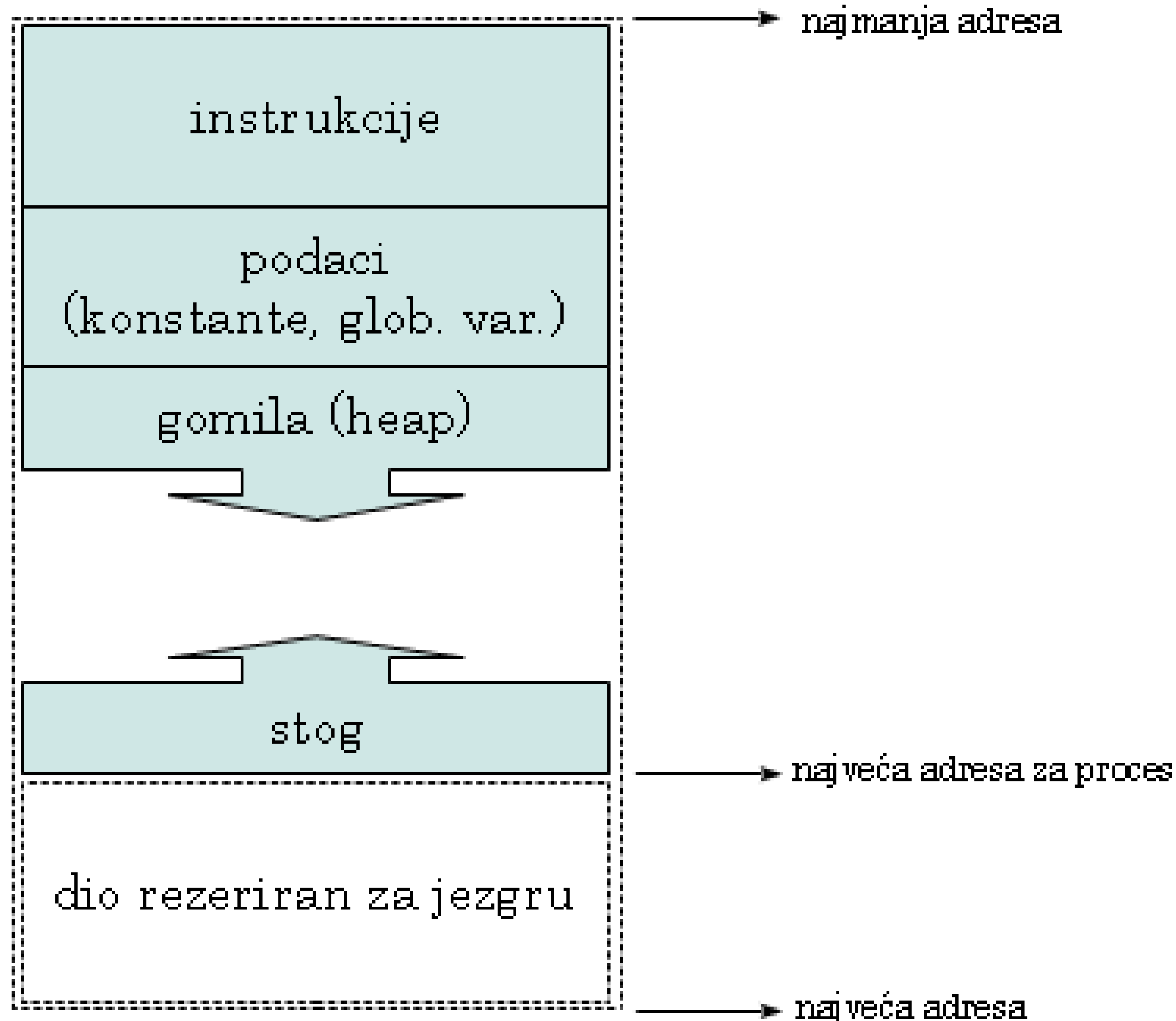
Dinamičko upravljanje spremnikom

dodatni materijali

Dinamičko upravljanje spremnikom u OS-u

- **koristi se na više mjesta/razina u operacijskim sustavima**
  - (već prikazano) za dodjelu adresnog prostora procesima
    - u sustavima koji ne koriste straničenje već "dinamičko upravljanje spremnikom"
  - za upravljanje prostorom na razini OS-a
    - koji su dijelovi za jezgru, za procese, za međuspremnike (buffere) naprava i slično
  - za upravljanje gomilom (heap) unutar jednog procesa
    - u programima: `malloc`/`free`, `new`/`delete` i sl.
    - u nastavku razmatramo samo ovo upravljanje iako se slični postupci mogu koristiti i drugdje

Uobičajena organizacija adresnog prostora procesa



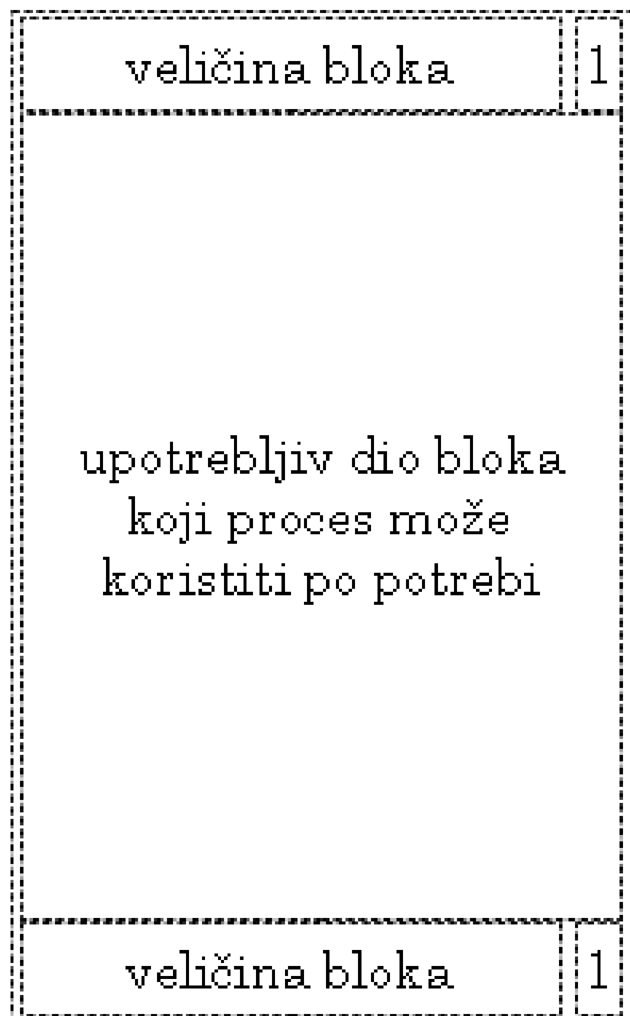
Gomila (heap)

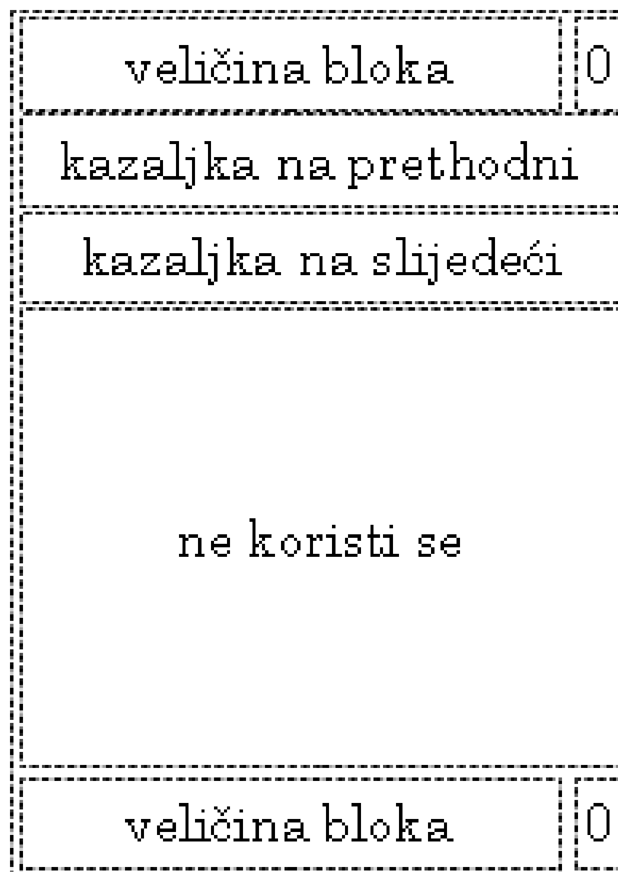
- **služi za posluživanje dinamičkih zahtjeva procesa**
  - pri pokretanju procesa nisu unaprijed poznati zahtjevi
    - veličina problema se zadaje/učitava naknadno
    - primjer: učitavanje i obrada polja/matrice
      - naprije se učitavaju dimenzije, potom radi zauzimanje spremnika, a tek potom učitavanje polja/matrice
  - u C-u:
    - malloc/free (i slični: calloc, realloc)
  - u C++ (Java):
    - new/delete (i slični)
- **veličina gomile raste s novim zahtjevima**
  - adresa do kuda je trenutno gomila narasla naziva se *program break value* (pogledati *sbrk*)

Slobodni i zauzeti blokovi

- **potrebna struktura podataka sastoji se od:**
  - liste slobodnih blokova
  - liste zauzetih blokova
    - nije neophodna; može služiti radi provjere rada algoritma (da se ne oslobađa nepostojeći blok)
- **svaki blok mora imati <u>*zaglavlje*</u> sa:**
  - veličinom bloka
  - oznakom zauzetosti
  - kazaljkama na idući/prethodni blok
    - prema potrebi algoritma, npr. nisu potrebne za zauzeti blok
- **<u>*podnožje*</u> (na kraju bloka) ovisno o algoritmu**
  - uglavnom su poželjna, radi jednostavnosti algoritma
  - zaglavlja što kraća da ne troše uzalud spremnički prostor

Primjer zaglavlja za zauzeti i slobodni blok





Primjer korištenja navedene strukture

- **primjer je u priloženom kodu**

```c
/*! Primjer struktura podataka za dinamičko upravljanje spremnikom (labosi) */

#include <stdio.h>
#include <malloc.h>

/* veličine blokova moraju biti višekratnik od 2 da bi se
   zadnji bit mogao iskoristiti za oznaku zauzetosti */

/*! zaglavlja za zauzete blokove */
struct zag_zau {
	int vel; /* veličina bloka uključuje zaglavlja */
};

/*! podnožje za zauzete blokove jednako je zaglavlju */
#define pod_zau zag_zau

/*! zaglavlje za slobodne blokove */
struct zag_slo {
	int vel; /* veličina bloka uključuje zaglavlja */
	struct zag_slo *preth;
	struct zag_slo *iduci;
};

/* podnožje za slobodne blokove jednako je zaglavlju zauzetih */
#define pod_slo zag_zau

/* B - adresa zaglavlja - početka bloka, A - adresa "korisnog" dijela */
#define ADR_KORISNO(B)     ( (void *) ( ( (char *)(B) ) + sizeof(int) ) )
#define ADR_BLOK(A) ( (void *) ( ( (char *)(A) ) - sizeof(int) ) )

#define ZB(B)       ((struct zag_zau *)(B))    /* adresa u tip zag_zau */
#define SB(B)       ((struct zag_slo *)(B))    /* adresa u tip zag_slo */
#define VEL(B)             ( ZB(B)->vel & (~1) )      /* vrati veličinu bloka
*/
#define ZAUZ(B)            ( ZB(B)->vel & 1 )  /* je li blok zauzet */

#define POSTAVI_VEL(B,V)   do ZB(B)->vel = (V) | ZAUZ(B); while(0)
#define POSTAVI_ZAUZ(B)           do ZB(B)->vel = (ZB(B)->vel) | 1; while(0)
#define POSTAVI_SLOB(B)           do ZB(B)->vel = VEL(B); while(0)

/* idući/prethodni: po adresama */
#define IDUCI_BLOK(B)             ( (void *) ( (char *) (B) + VEL(B) ) )
#define PRETH_BLOK(B)             ( (void *) ( (char *) (B) - VEL(ZB(B)-1) ) )

/* postavi podnožje (kopiraj veličinu i oznaku zauzetosti na kraj bloka) */
#define POSTAVI_PODN(B)    \
do *( (int *) ADR_BLOK ( IDUCI_BLOK(B) ) ) = (B)->vel; while(0)

/* podnožje prethodnog bloka */
#define PRETH_PODN(B)             ( (void *) ( (char *) (B) - sizeof(int) ) )

/* idući/prethodni po listi slobodnih blokova */
#define IDUCI_SLOB(B)             ( SB(B)->iduci )
#define PRETH_SLOB(B)             ( SB(B)->preth )
```

```c
void ispis ( void *prvi_po_adresi, void *prvi_slobodni )
{
	struct zag_zau *iter1;
	struct zag_slo *iter2;

	printf ( "\nPopis svih blokova:\n" );
	for(	iter1 = prvi_po_adresi;
		(long) iter1 < (long) prvi_po_adresi + 10000;
		iter1 = IDUCI_BLOK(iter1) )
	{
		printf ( "Blok na adresi %p velicine %d (zauzet=%d)\n",
			iter1, VEL(iter1), ZAUZ(iter1) );
	}
	printf ( "\nLista slobodnih blokova:\n" );
	for ( iter2 = prvi_slobodni; iter2 != NULL; iter2 = IDUCI_SLOB(iter2) )
	{
		printf ( "Blok na adresi %p velicine %d (zauzet=%d)\n",
			iter2, VEL(iter2), ZAUZ(iter2) );
	}
}

int main(void)
{
	struct zag_zau *a, *c;
	struct zag_slo *b, *d, *x, *y;
	struct zag_slo *lista;

	a = malloc ( 10000 );

	POSTAVI_VEL ( a, 1000 );
	POSTAVI_ZAUZ (a);
	POSTAVI_PODN (a);

	b = IDUCI_BLOK (a);
	POSTAVI_VEL ( b, 1000 );
	POSTAVI_SLOB (b);
	POSTAVI_PODN (b);
	lista = b;
	IDUCI_SLOB(b) = NULL;
	PRETH_SLOB(b) = NULL;

	c = IDUCI_BLOK (b);
	POSTAVI_VEL ( c, 1000 );
	POSTAVI_ZAUZ (c);
	POSTAVI_PODN (c);

	d = IDUCI_BLOK (c);
	POSTAVI_VEL ( d, 7000 );
	POSTAVI_SLOB (d);
	POSTAVI_PODN (d);
	IDUCI_SLOB(lista) = d;
	PRETH_SLOB(d) = lista;
	IDUCI_SLOB(d) = NULL;
```

```
    ispis ( a, lista );

    //oslobodi blok c i spoji ga s b i d
    x = PRETH_BLOK(c);
    if ( !ZAUZ(x) ) {
        printf ("\nPrethodni blok od %p je %p i nije zauzet!\n", c, x);
        //spajanje s prethodnim slobodnim:
        //sve sto treba napraviti je spojiti ga, popraviti veličinu te
        //azurirati zaglavlje
        // | slob || zauz | => |      slob      |
        POSTAVI_VEL ( x, VEL(x) + VEL(c) );
        POSTAVI_PODN (x);
        //obzirom da lista slobodnih nije složena nije potrebno ništa
        //vise raditi (inače bi trebalo prvo 'x' maknuti iz liste,
        //spojiti ga s 'c' te tada staviti u listu slobodnih
        c = (void *) x;
        printf ( "Nakon spajanja:\n" );
        ispis ( a, lista );
    }

    y = IDUCI_BLOK (c);
    if ( !ZAUZ(y) ) {
        printf ("\nSlijedeci blok od %p je %p i nije zauzet!\n", c, y);

        if ( !ZAUZ(c) ) {
            //maknimo prvo 'c' iz liste slobodnih
            //povezivanje prethodnog s blokom iza 'c'
            if ( (void *) lista == (void *) c ) //na prvom mjestu
                lista = IDUCI_SLOB (c);
            else
                IDUCI_SLOB ( PRETH_SLOB (c) ) = IDUCI_SLOB (c);
            //povezivanje bloka iza 'c' s prethodnikom od 'c'
            if ( IDUCI_SLOB (c) != NULL )
                PRETH_SLOB ( IDUCI_SLOB (c) ) = PRETH_SLOB (c);
        }
        //spojimo 'c' sa 'y' u blok koji počinje na mjestu 'c'
        POSTAVI_VEL ( c, VEL(c) + VEL(y) );
        POSTAVI_PODN (c);
        //ažurirati veze u listi
        IDUCI_SLOB (c) = IDUCI_SLOB (y);
        PRETH_SLOB (c) = PRETH_SLOB (y);
        if ( lista == y )
            lista = (void *) c;
        else
            IDUCI_SLOB ( PRETH_SLOB (y) ) = (void *) c;

        if ( IDUCI_SLOB (y) != NULL )
            PRETH_SLOB ( IDUCI_SLOB (y) ) = (void *) c;

        printf ( "Nakon spajanja:\n" );
        ispis ( a, lista );
    }
    return 0;
}
```