Programovací jazyk C(++)

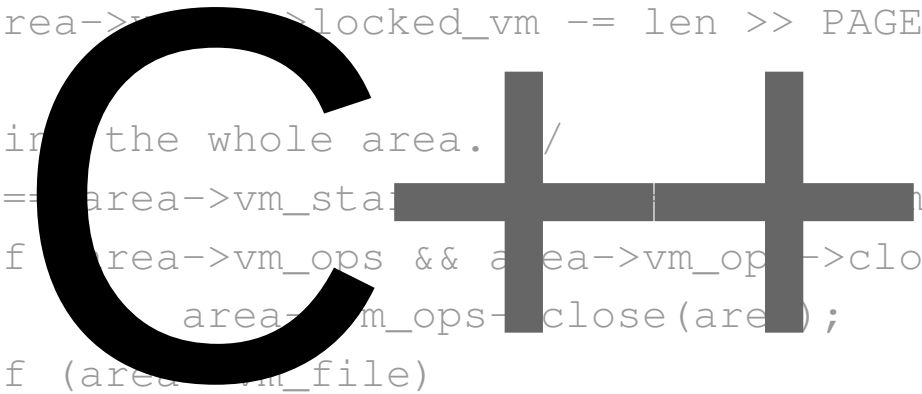
```
static struct vm_area_struct * unmap_fixup(struct mm_struct *mm,
        struct vm_area_struct *area, unsigned long addr, size_t len,
        struct vm_area_struct *extra)
{
        struct vm_area_struct *mpnt;
        unsigned long end = addr + len;

        area->vm_mm->total_vm -= len >> PAGE_SHIFT;
        if (area->vm_flags & VM_LOCKED)
                area->vm_mm->locked_vm -= len >> PAGE_SHIFT;

        /* Unmapping the whole area. */
        if (addr == area->vm_start && end == area->vm_end) {
                if (area->vm_ops && area->vm_ops->close)
                        area->vm_ops->close(area);
                if (area->vm_file)
                        fput(area->vm_file);
                kmem_cache_free(vm_area_cachep, area);
                return extra;
        }
```

4. Struktury, unie

Miroslav Spousta, 2004



Struktury

- datová struktura, která obsahuje položky různých typů

  obdoba record v Pascalu

  pole je podobná struktura, ale má všechny položky stejného typu

```
struct {
     char id[10]; int sirka; int vyska;
} zbozi;
strcpy(zbozi.id, „police“);
zbozi.sirka = 40; zbozi.vyska = 20;
```

- definuje proměnnou **`zbozi`** typu struktura, k jednotlivým položkám se přistupuje pomocí operátoru **`.`** (tečka)

Pojmenované struktury

- anonymní struktury nemůžeme použít pro definici dalších proměnných
- řešení: pojmenované struktury

```
struct zbozi {
     char id[10]; int sirka; int vyska;
};
struct zbozi x, y;  /* definice proměnných typu
                       struct zbozi */
```

- pozor, v definici proměnné je nutné uvádět celý typ (**`struct zbozi`**)
- nejčastější použití: dynamické proměnné, parametry funkcí

Struktury

- nemůžeme definovat strukturu rekurzivně (nemá smysl)
- je ale možné ve struktuře definovat ukazatel na stejnou strukturu

  typické použití: např. strom nebo spojový seznam

```
struct item {
     int data;
     struct item *left, *right;
};
```

- v jedné struktuře je ale možné uvést jinou strukturu (např. ve struktuře osoba může být struktura adresa, přistupuje se k ní opět pomocí operátoru .

Dynamicky alokované struktury

- pomocí funkcí **`malloc()`** a **`free()`**

```
struct st {
      int x, y;
};
struct st *p = (struct st *)malloc(sizeof(struct st));
free(p);
```

- k dynamicky alokované struktuře se často přistupuje pomocí operátoru **`->`**
- např.: **`x->name;`** namísto **`(*x).name`**
- velikost struktury získáme pomocí operátoru **`sizeof()`**
- např.: **`sizeof(struct osoba)`**

Dynamicky alokované struktury

```
struct osoba {
      char jmeno[20]; int vek; struct osoba *next;
};          /* POZOR na středník! */
struct osoba *osoby = NULL; char s[20];
while (fgets(s, 20, stdin)) {
      struct osoba *o =
         (struct osoba *)malloc(sizeof(struct osoba));
      strcpy(o->jmeno, s); o->vek = 0;
      o->next = osoby;          /* (*o).next */
      osoby = o;
}
```

Deklarace typu

- nové pojmenování nějakého typu
- podobně jako deklarace proměnné, ale navíc před ni napíšeme klíčové slovo **`typedef`**:

```
typedef int cele; /* cele je novy typ */
cele x;           /* je ekvivalentni int x; */
```

- **`typedef`** můžeme použít u pro složitější typy:

```
typedef int *ui;  /* novy typ ukazatel na int */
ui x, *y;         /* == int *x, **y; */
```

Deklarace typu

- můžeme definovat nový typ i pro strukturu

```
typedef struct osoba { /* struct osoba */
     char jmeno[20]; int vek;
} osoba_t;                  /* osoba_t je typ */
osoba_t rodic, syn;         /* definice proměnných */
struct osoba dcera;         /* lze i takto */
```

- pokud budeme používat pouze nový typ, můžeme jméno struktury vynechat:

```
typedef struct {            /* anonymní struktura */
     char jmeno[20];
} osoba_t;
osoba_t rodic, syn;         /* pouze takto */
```

Výčtový typ

- definuje symbolické konstanty (které mají nějakou souvislost)
- např. pro barvy můžeme zavést typ:

```
enum tyden {
     pondeli, utery, streda, ctvrtek, patek,
     sobota, nedele
};
enum tyden t = pondeli;
```

- zpřehledňují program, je vhodné je použít vždy, když mají konstanty nějakou souvislost
- vnitřně je reprezentován jako nejmenší typ, který pojme dané hodnoty, pokud nepřiřadíme konstantám hodnoty, číslují se od 0

Výčtový typ

- předdefinované hodnoty:

```
enum barva {
     black,          /* dostane hodnotu 0 */
     red = 6,
     green = 4,
     blue;           /* dostane hodnotu 5 */
};
```

- často je vhodnější použít výčtový typ než makro preprocesoru:
- `#define BLACK 4`

Union

- typ **`union`**, něco jako variantní záznam v Pascalu
- jednotlivé proměnné se v paměti překrývají
- v paměti se vyhradí místo pro nejdelší typ

```
union x {
        char c; int i; float f;
};
union x a[10];
a[2].c = 'c';
a[2].c = 44;
```