Práce se soubory

Úvod do programování 2

Tomáš Kühr

Soubory z pohledu C

- Soubor chápeme jako posloupnost bytů uložených na disku
- Datový proud (anglicky stream)
  - Ještě obecnější přístup
  - Sjednocuje pohled na soubory a vstupní/výstupní zařízení
- Přístup k disku je pomalejší než k paměti → buffering / bufferování / bufrování
- Buffering vstupu
  - Do paměti (buffer) se načte blok dat určité a předem dané velikosti
  - Jednotlivé hodnoty se čtou z této paměti
  - Pokud je třeba, načte se do paměti další blok dat
- Buffering výstupu
  - Data se zapisují do paměti
  - Pokud je paměť plná, zapíše se její obsah na disk

Textové soubory

Začátek práce se souborem

- Všechny zmiňované funkce, proměnné a datové jsou definovány v stdio.h
- Se soubory pracujeme prostřednictvím ukazatelů na strukturovaný typ FILE
```
FILE *identifikator;
```
- Nejprve je potřeba soubor otevřít pomocí funkce fopen
```
FILE *fopen(char *filename, char *mode);
```
- První parametr funkce určuje jméno souboru (případně i cestu)
- Druhý parametr určuje, jak se bude se souborem pracovat (viz dále)
- Funkce vrací ukazatel na „soubor“ nebo NULL, pokud dojde k chybě
- Příklad:
```
FILE *mujSoubor;

mujSoubor = fopen("./data/info.txt", "r+t");
if (!mujSoubor) exit(1);
```

Způsoby otevření souboru

- Jsou určeny hodnotou řetězce mode
- Určují další možnosti práce s otevřeným souborem
- Módy otevření:
  - `"r"` – otevření pro čtení, soubor musí existovat
  - `"w"` – otevření pro zápis, existující soubor se přepíše, neexistující se vytvoří
  - `"a"` – rozšiřování existujícího souboru, neexistující soubor bude vytvořen
  - `"r+"`, `"w+"`, `"a+"` - rozšířené módy, umožňují jak čtení, tak zápis
- Za výše uvedenými řetězci může následovat znak:
  - b – pro označení binárních souborů
  - t (nebo nic) – pro označení textových souborů

Čtení ze souboru

- Soubor musí být otevřen v módu podporujícím čtení
- Čtení jednoho znaku
```
int getc(FILE *f);
```
- Čtení řádku
```
char *fgets(char *str, int length, FILE *f);
```
  - Do str se ze souboru f načte nejvýše jeden řádek (včetně '\n')
  - Maximálně se ale přečte length-1 znaků
- Formátované čtení
```
int fscanf(FILE *f, const char *format, ...);
```
- Po každém čtení se automaticky posouvá pozice v souboru
- Pokud nelze z datového proudu číst, vrací funkce EOF nebo NULL
- Chybová návratová hodnota se vrací i v případě čtení na konci souboru

Zápis do souboru

- Soubor musí být otevřen v módu podporujícím zápis
- Zápis znaku
```
int putc(int c, FILE *f);
```
- Zápis řetězce
```
int fputs(const char *s, FILE *f);
```
- Formátovaný zápis
```
int fprintf(FILE *f, const char *format, ...);
```
- Každý následující zápis se provede automaticky za dříve zapsaná data
- Pokud dojde k chybě při zápisu, vrací výše uvedené funkce hodnotu EOF

Ukončení práce se souborem

- Počet současně otevřených souborů je omezen
- Pokud při běhu programu dojde k závažné chybě (např. výpadek el. proudu), data zapsaná pouze do bufferu budou ztracena
- Vyprázdnění bufferu

```
int fflush(FILE *f);
```

  - V případě vstupu - zahození obsahu bufferu a načtení nového bloku
  - Toto se používá nejčastěji ve spojení se standardním vstupem (viz dále)
  - V případě výstupu - vynucení zápisu obsahu bufferu na disk
- Zavření proudu (včetně vyprázdnění bufferu)

```
int fclose(FILE *f);
```

- V případě neúspěchu vracejí obě funkce hodnotu EOF

Standardní datové proudy

- Načítání dat a výpisy pomocí konzole jsou realizovány datovými proudy
- Po spuštění programu jsou vytvořeny datové proudy

```
FILE *stdin;
FILE *stdout;
FILE *stderr;
```

  - stdin pro čtení (implicitně z konzole)
  - stdout pro zápis (implicitně do konzole)
  - stderr pro zápis (chybový výpis, implicitně do konzole)
- Implicitní nastavení proudů lze změnit za běhu programu (viz dále)
- Nebo již při spouštění programu z příkazové řádky

```
program.exe < vstup.txt > vystup.txt 2> chyby.txt
```

Binární soubory

Binární soubory

- Mohou mít libovolnou strukturu
- Data jsou uložena obvykle ve stejné podobě jako v proměnných
- Výhody:
  - Pro uložení je potřeba obvykle méně místa než u textových souborů
  - Nejsou potřeba konverze čísel na textové řetězce a naopak
  - Zápis i čtení dat bývají díky tomu rychlejší
- Nevýhody
  - Nelze je rozumně vytvářet, číst, editovat v textových editorech
  - Binární soubory nejsou obecně přenositelné
- Po otevření binárního souboru se implicitně neprovádí žádné úpravy (např. transformace zakódování konců řádků na ‘\n’)

Čtení bloků dat daného typu

- Čtení bloku dat lze provést voláním funkce
  `size_t fread(void *kam, size_t rozmer, size_t pocet, FILE *f)`
  - f je datový proud, ze kterého se čte
  - rozmer je velikost jedné položky
  - pocet udává počet položek
  - kam je adresa paměti, do které se data ukládají
  - vrací počet úspěšně přečtených položek
- Příklad použití:

```
#define VELIKOST_BLOKU 10
...
int data[VELIKOST_BLOKU];
FILE *fr = fopen("a.dat", "rb");
...
fread(data, sizeof(int), VELIKOST_BLOKU, fr);
```

Zápis bloku dat daného typu

- Zápis bloku dat lze provést voláním funkce
  `size_t fwrite(void *zdroj, size_t rozmer, size_t pocet, FILE *f)`
  - f je datový proud, do kterého zapisujeme
  - rozmer je velikost jedné položky
  - pocet udává počet položek
  - zdroj je adresa dat, která chceme zapisovat
  - vrací počet úspěšně zapsaných položek
- Příklad použití:

```
#define VELIKOST_BLOKU 10
...
int data[VELIKOST_BLOKU];
FILE *fw = fopen("out.dat", "wb");
...
fwrite(data, sizeof(int), VELIKOST_BLOKU,fw);
```

Posunutí se v souboru

- Obvykle data čteme / zapisujeme sekvenčně od začátku souboru
- Občas ale potřebujeme k datům přistupovat i jinak
- Posouvání se v souboru pomocí
  `int fseek(FILE *f, long posun, int odkud)`
  - posun udává, o kolik bytů se posouváme
    - záporný posun odpovídá směru k začátku souboru
    - kladný posun pak směru ke konci souboru
  - odkud udává výchozí pozici a bývá dáno jednou z konstant:
    - SEEK_SET (od začátku souboru)
    - SEEK_CUR (od aktuální pozice)
    - SEEK_END (od konce souboru)
  - vrací nulu v případě úspěchu, jinak nenulovou hodnotu

Zjištění aktuální pozice

- Funkce pro zjištění aktuální pozice v souboru
  ```
  long ftell(FILE *f)
  ```
  - vrací aktuální posunutí od začátku souboru v bytech
- Příklad použití:
  ```
  /* navrat na puvodni misto */
  long akt_pos = ftell(f);
  fseek(f, 0L, SEEK_SET);
  if (vyhledej(f, "ahoj") == NULL)
      fseek(f, akt_pos, SEEK_SET);
  ```
- Vstupní operace **nemá** přímo následovat po výstupní operaci nebo naopak bez předchozího volání fseek
- Občas se nic nestane, občas to působí špatně laditelné chyby
- V případě potřeby pomůže
  ```
  fseek(f, 0L, SEEK_CUR);
  ```

Další drobnosti

Vrácení znaku do bufferu

- V programech se někdy dozvíme o tom, že máme přestat číst až ve chvíli, kdy jsme přečetli jeden znak navíc…
- Funkce pro virtuální vrácení znaku do bufferu
  ```
  int ungetc(int c, FILE *f);
  ```
- Při úspěšném provedení je návratovou hodnotou vrácený znak c, jinak EOF
- Doporučuje se nevolat víckrát po sobě
- Lze „vrátit“ i jiný znak než ten, který se dříve přečetl

Změny v nastavení bufferování

- O vytvoření a používání bufferu se obvykle sami nestaráme
- Funkce pro přiřazení vlastního bufferu nově otevřenému proudu
  `void setbuf(FILE *f, char *buffer)`
  - buffer musí mít velikost minimálně BUFSIZ bytů
  - při použití hodnoty NULL, dojde k vypnutí bufferování
- Funkce s více možnostmi
  `void setvbuf(FILE *f, char *buffer, int mode, size_t size)`
  - mode umožňuje nastavit režim bufferování
    - _IOFBF – data se načítají do zaplnění bufferu
    - _IOLBF – do konce řádky nebo do zaplnění bufferu
    - _IONBF – vypnutí bufferování
  - Parametr size umožňuje zadat velikost bufferu

Dočasné soubory

- Vytvoření dočasného souboru
  `FILE *tmpfile()`
  - Otevře soubor v režimu "w+b"
  - Po uzavření se tento automaticky smaže
  - Vrací ukazatel na vytvořený soubor nebo NULL
- Pokud chceme dočasný soubor používat jiným způsobem, musíme si jej otevřít sami (pomocí fopen)
- Pro generování jmen pomocných souborů lze použít
  `char *tmpnam(char *str)`
  - Generuje unikátní název souboru v daném adresáři
  - Uloží vygenerovaný řetězec do str a vrací ho i jako návratovou hodnotu
  - Pokud má str hodnotu NULL, alokuje se i paměť pro výsledný řetězec

Další užitečné funkce

- Přesměrování proudu za běhu programu
  ```
  FILE *freopen(const char *name, const char *mode, FILE *f)
  ```
  - Otevře soubor name v režimu mode
  - Přesměruje do nově otevřeného souboru proud f
  - Používá se pro přesměrování standardních proudů do souborů
  - Při úspěchu vrací ukazatel na proud, jinak NULL
- Přejmenování souboru
  ```
  int rename(const char *old_name, const char *new_name)
  ```
  - V případě úspěchu vrací nulu, jinak nenulovou hodnotu
- Smazání souboru
  ```
  int remove(const char *name)
  ```
  - V případě úspěchu vrací nulu, jinak nenulovou hodnotu