Súborový systém a jazyk C

Náhradná Prednáška 9, PROG-2, 2020
Pavol Zajac

Súborový systém a súbory

Z hľadiska používateľa, súborový systém predstavuje úložné miesto pre dáta. Pre programátorov ako používateľov: v súborovom systéme máme uložené zdrojové súbory, skompilované programy, testovacie dáta, ... Pre programátorov z hľadiska funkcionality: súborový systém potrebujeme hlavne kvôli **perzistencii** ("trvanlivosti") dát. Normálne, ak sa program v pamäti ukončí, všetky dáta v pamäti, s ktorými sme pracovali zmiznú. Po reštarte programu začíname iba s tými dátami, ktoré sme špecifikovali v zdrojovom kóde pred prekladom. Často však chceme, aby program načítal vstupné dáta z úložiska (*vstupného súboru*), a spracované dáta uložil do nejakého úložiska (*výstupného súboru*). Jeden fyzický súbor môže byť z logického pohľadu použitý ako aj vstupný, tak aj výstupný súbor.

Ak si chcete pozrieť populárny pohľad, ako súborový systém funguje, našiel som jedno pekné ilustračné video:

https://www.youtube.com/watch?v=KN8YgJnShPM

Vznik jazyka C bol úzko prepojený s vývojom operačného systému UNIX. Štandardná knižnica jazyka C, o ktorej bude táto prednáška, reflektuje architektúru súborového systému v UNIX-e. Na tejto architektúre stavia aj modernejší LINUX. Opäť som vybral dobré video, ktoré vysvetľuje špecifiká LINUX systému:

https://www.youtube.com/watch?v=HbgzrKJvDRw

Čo je dôležité uvedomiť si, že všetko sa dá reprezentovať ako súbor:

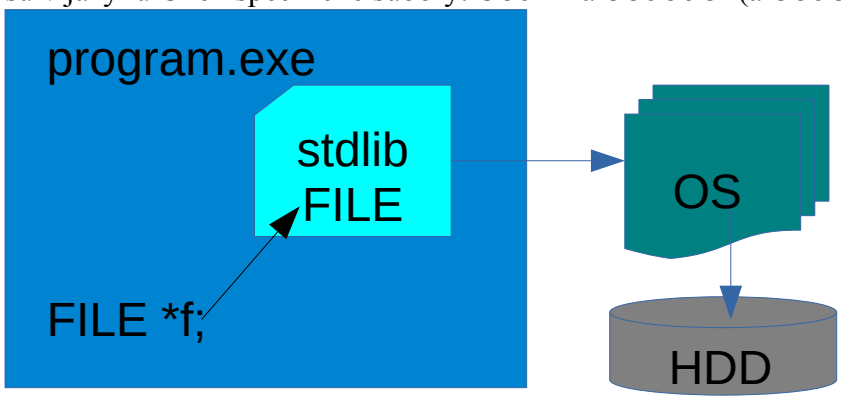
- perzistentné dátové úložisko /klasický súbor v súborovom systéme/
- postupnosť znakov, ktoré prichádzajú od používateľa z klávesnice cez konzolu (štandardný vstup)
- výstupná postupnosť znakov, ktorá sa posiela na konzolu (štandardný výstup)
- sieťové pripojenia, COM porty, rôzne I/O zariadenia /napr. tlačiareň, MIDI sequencer/, ...

Pre programátora, súbor bude **sekvenčná postupnosť bajtov**. Sekvenčná znamená, že nevieme adresovať jednotlivé bajty, ale spracúvame ich zaradom ako keby sme čítali/zapisovali na magnetickú pásku (*vyhľadajte si na internete, ak neviete čo je magnetická páska :)*). Miesto, kde sa zapisuje/číta sa označuje ako **aktuálna pozícia** v súbore. Niektoré súbory (napr. perzistentné súbory na disku) podporujú presun pozície v rámci súboru (napr. vrátime sa na začiatok), podobne, ako by sme previnuli magnetickú pásku. Iné súbory (napr. vstup z klávesnice, výstup na konzolu) takýto presun pozície neumožňujú. Totiž, ak sa už prečítal vstupný znak z klávesnice, nedá sa k nemu vrátiť, ale bajt na disku zostáva (pokým sme ho iným programom nezmenili, tzv. problémy konkurentného prístupu).

Ďalšiu vec, ktorej potrebuje programátor rozumieť, je, že štandardná knižnica jazyka C nepracuje so súborom priamo na systémovej úrovni, ale sprostredkovane cez knižničné funkcie. Existujú knižnice a API, ktoré podporujú priamu prácu so súborovým systémom a s príslušnými systémovými volaniami (funkcie operačného systému dostupné pre náš program priamo), ale tie sú neprenosné, a závislé na konkrétnom systéme (iné v LINUXe, vo Windows, ...). Na niektoré nízkoúrovňové úlohy je potrebné poznať aj tieto knižnice, ale v našom predmete ich používať nebudeme.

Abstrakcia súborov v štandardnej knižnici jazyka C

Väčšina funkcií zo štandardnej knižnice jazyka C na prácu so súbormi je definovaná v `stdio.h`. Patria k nim aj funkcie na prácu s konzolovým vstupom a výstupom, keďže konzolový vstup a výstup sú v jazyku C len špecifické súbory: `stdin` a `stdout` (a `stderr`).



Súčasťou nášho skompilovaného programu budú funkcie a globálne premenné zo štandardnej knižnice, ktoré umožňujú spoluprácu s (rôznymi) operačnými systémami. Na úrovni knižnice reprezentuje súbor skupina dát (premenná štruktúrového typu `FILE`), ku ktorej máme prístup ako programátori cez jej adresu, smerník typu `FILE*`. Knižničné funkcie obsluhujú interakciu s OS, manažujú pozíciu v súbore a zabezpečujú bufferovanie dát.

***Buffer (vyrovnávacia pamäť)*:** keď chceme zo súbora nejaký konkrétny bajt, je často neefektívne pýtať si každý bajt individuálne. Dátové úložiská poskytujú väčšie skupiny bajtov naraz (blok z disku, paket zo siete, riadok z klávesnice), resp. naraz zapíšeme väčšie bloky bajtov. Štandardná knižnica zabezpečuje dátový buffer: pole asociované zo súborom. Keď si napr. pýtame dáta zo súboru, najprv sa pozrie knižničná funkcia do vyrovnávacej pamäte. Ak sa tam nachádzajú dáta, použijú sa, inak sa načíta ďalší blok dát. Podobne pri zápise: najprv sa plní pamäť, a keď je dostatok dát, pošlú sa na výstupné zariadenie.

Efekty vyrovnávacej pamäte ste videli, ak ste v cykle načítavali po jednom znaky cez scanf (používa sa vstupný súbor stdin, ktorý je bufferovaný po riadkoch). Poznať výstupný buffer je dôležité, ak si chcete odložiť údaje do súboru: aj keď použijete funkciu na zápis, dáta sa vo fyzickom súbore môžu objaviť až oveľa neskôr (a ak medzitým havaruje program, nemusia sa zapísať vôbec).

Ak potrebujete explicitne kontrolovať buffer pre súbor, pozrite si funkcie `fflush`, `setbuf`, `setvbuf`.

Otvorenie súboru

Pri spustení štandarného konzolového programu máte k dispozícii 3 otvorené súbory, ku ktorým pristupujete cez globálne smerníky:

1. `stdin`: štandarný vstup, z klávesnice, používa sa vo funkciách ako `scanf`, `getchar`, ...
2. `stdout`: štandardný výstup, na konzolu, používa sa vo funkciách ako `printf`, `putchar`, ...
3. `stderr`: štandardný chybový výstup, tiež na konzolu, musíte ho použiť ručne:

```
if (argc < 2)
{
    fprintf(stderr, "Chybajuce parametre\n");
}
```

Každý z týchto súborov sa dá presmerovať pri volaní programu (*TODO: naštudujte si ako*). Keďže `stdout` a `stderr` sa dajú presmerovať zvlášť, môžete napr. vypisovať chyby na konzolu, ale správne výsledky do súboru.

Ak potrebujeme v programe používať aj iné súbory, musíme ich explicitne **otvoriť**, a po skončení práce s nimi ich **uzavrieť** (čím sa uvoľnia systémové prostriedky, a zabezpečí sa finálne uloženie na disk). Súbory `stdin`, `stdout`, `stderr` sú otvorené a uzavreté automaticky pred/po spustením `main`.

Súbor sa dá otvoriť pomocou funkcie

```
FILE* fopen(const char* filename, const char* mode);
```

Funkcia dostane meno súboru (`filename`), a mód otvorenia (`mode`, pozrite si presnú dokumentáciu!), a vráti adresu (handle), ktorú je potrebné uložiť do smerníka typu `FILE*`. Všetky ostatné funkcie pracujú s touto adresou, nie menom súboru!

Základné módy otvorenia sú:

1. "r" : *read mode*, súbor otvoríme na čítanie, ak existuje (a dá sa čítať).
2. "w" : *write mode*, súbor otvoríme na zápis. Ak existuje (a máme právo na zápis), doterajší obsah sa zmaže! Ak neexistuje, vytvorí nový súbor (ak na to máme oprávnenia).
3. "a" : *append mode*, súbor otvoríme na pripisovanie údajov. Súbor musí existovať, do súboru zapisujeme na koniec (predošlý obsah je pre nás neviditeľný).

Do módu sa tiež dá definovať, či so súborom pracujeme ako textovým (default možnosť) alebo binárnym ("b"). Binárny prístup umožňuje prácu aj so špeciálnymi znakmi, a mali by sme vidieť presne tie bajty aké sú v zariadení (pri textových sa niektoré postupnosti bajtov môžu prekladať).

POZOR! Otvorenie súboru môže zlyhať (súbor neexistuje, nemáme práva, ...). Vtedy `fopen` vráti adresu `NULL`, a tento prípad by sme vždy mali kontrolovať (inak program zhavaruje, keď budeme niečo chcieť robiť so súborom).

Príklad: Program otvorí výstupný súbor s menom, ktoré je prvým parametrom programu. Ak sa tento nedá otvoriť, použije na výstup stdin, a informuje o tom výpisom na štandardný chybový výstup:

```
int main(int argc, char* argv) {
  FILE *f;      //bude obsahovat adresu vystupneho suboru

  if (argc < 2) {
     fprintf(stderr, "Chyba parameter, pouzivam stdout\n");
     f = stdout;
           //stdout obsahuje adresu, ktora sa prekopiruje do f
  }
```

```
  else if ((f = fopen(argv[1], "r")) == NULL)
  {
     fprintf(stderr, "Chyba, neviem otvorit subor %s\n", argv[1]);
     f = stdout;
  }
  //TODO: pracujeme s f ...

  //ak sme nepracovali s stdout, mali by sme f uzavriet
  if (f != stdout) {
     fclose(f);
  }
}
```

Ako bolo vidieť v príklade, súbor uzavrieme cez príkaz `int fclose(FILE* file);` Podobne ako v prípade `free`, v pointeri `file` zostane adresa pamäťovej štruktúry, ktorá bude neplatná, a túto adresu už nemôžeme používať. POZOR, samotný *pointer* `file` môžeme znovu použiť: uložiť do neho adresu novo otvoreného súboru:

```
FILE *f;
f = fopen("input.txt", "r");
if (f != NULL) { //TODO: read all...
     fclose(f);
//POZOR: fclose sa nesmie pouzit s NULL, preto je v if bloku
}
f = fopen("output.txt", "w");  //pozrite aj funkciu freopen...
if (f != NULL) { //TODO: write result
     fclose(f);
}
```

Poznámka: Adresu otvoreného súboru je možné ďalej posielať iným funkciám. Napr. si môžete spraviť funkciu, ktorá zo súboru prečíta jeden štruktúrovaný údaj, a použiť ju v cykle na naplnenie poľa. Alebo jedna funkcia ošetrí otvorenie súborov, a iné funkcie spracujú obsah už otvorených súborov. Ako parameter týchto funkcií bude potom pointer typu `FILE*`. Ak chceme, aby funkcia otvorila súbor, mala by mať parameter meno súboru (pointer typu `char*`). Funkcia môže vrátiť aj adresu otvoreného súboru (vtedy je jej návratová hodnota typu `FILE*`), volajúca funkcia sa potom musí postarať o jeho uzavretie.

Čítanie dát zo súboru

Všetky vstupné funkcie očakávajú, že máte k dispozícii adresu otvoreného súboru na čítanie. Na čítanie textových dát sa používajú:

1. `int fscanf ( FILE * stream, const char * format, ... );`
   Funkcia `scanf` je vlastne len `fscanf(stdin, ...)`. Načíta formátovaný vstup zo súboru (formát a parametre sú presne také isté ako pre `scanf`).

2. `int fgetc ( FILE * stream );`
   Funkcia `getchar` je vlastne len `fgetc(stdin)`. Načíta jeden znak zo súboru.

3. `char * fgets ( char * str, int num, FILE * stream );`

   Použili sme ju už pri načítavaní reťazcov. Načíta do znakového poľa `str` reťazec zo vstupného súboru `stream`, maximálne jeden riadok alebo `num`-1 znakov (čo je menšie).

Na binárne dáta je okrem `fgetc` určená funkcia

1. `size_t fread ( void * ptr, size_t size, size_t count, FILE * stream );`

   Načíta na adresu `ptr` (napr. pole ľubovoľného typu) bajty zo súboru `stream`. Zo súboru prečíta `count` blokov veľkosti `size` bajtov, teda spolu `count*size` bajtov.

Pri čítaní dát je dôležité si uvedomiť, že môžeme postupne prečítať všetky dáta zo súboru, alebo môže nastať chyba v prístupe (napr. spadlo sieťové spojenie). Každá zo vstupných funkcií má spôsob, ako tieto stavy indikovať pomocou návratovej hodnoty.

TODO: Je dôležité prečítať si dokumentáciu k týmto funkciám a preštudovať, ako je indikované zlyhanie načítania a stav, keď už nemáme k dispozícii dáta.

Vo všeobecnosti funkcie používame v cykle, ktorý beží, až kým sa dáta zo vstupného súboru neminú (aktuálna pozícia v súbore sa dostane za posledný bajt). Stav, že vstupný súbor je otvorený, ale sme na pozícii za posledným bajtom sa označuje ako stav *end-of-file* (koniec súboru). Tento stav sa dá detegovať funkciou `feof`. Chybový stav sa dá detegovať funkciou `ferror`.

Funkcie `fscanf` a `fgetc` vrátia hodnotu EOF (makro označujúce -1), ak sa pokúsime čítať dáta za koncom súboru. Funkcia `fgets` vráti NULL, ak sme sa pokúsili čítať za koncom súboru. Funkcia `fread` vráti koľko reálne načítala dát (ak je to menej ako `count`, dosiahli sme koniec alebo chybu).

Prílohou prednášky budú príklady demonštrujúce použitie týchto funkcií.

Zápis dát do súboru

Všetky výstupné funkcie očakávajú, že máte k dispozícii adresu otvoreného súboru na zápis (alebo pripisovanie). Na zápis textových dát sa používajú:

1. `int fprintf ( FILE * stream, const char * format, ... );`

   Funkcia `printf` je vlastne len `fprintf(stdin, ...)`. Zapíše údaj formátovaným spôsobom do súboru (formát a parametre sú presne také isté ako pre `printf`).

2. `int fputc ( int character, FILE * stream );`

   Funkcia `putchar` je vlastne len `fputc(c, stdin)`. Zapíše jeden znak do súboru.

Na binárne dáta je okrem `fputc` určená funkcia

1. `size_t fwrite ( void * ptr, size_t size, size_t count, FILE * stream );`

   Zapíše obsah pamäte od adresy `ptr` (napr. pole ľubovoľného typu) bajty do súboru `stream`. Do súboru zapisuje `count` blokov veľkosti `size` bajtov, teda spolu `count*size` bajtov. Na rozdiel

od `fprintf` nerobí formátovanie, čiže ak zapisujete napr. pole `int`-ov, objaví sa na disku binárny obraz týchto dát ako sú uložené v pamäti. Tieto údaje sú potom bežne nečitateľné pre človeka, ale ľahko ich viete načítať naspäť cez `fread` /napr. rýchle uloženie stavu hry/.

Pomocou výstupov funkcií sa dá skontrolovať, či zápis prebehol úspešne. POZOR: zapisuje sa len do vyrovnávacej pamäte. Keď chcete, aby sa dáta okamžite posunuli operačnému systému (OS má ďalší level bufferovania, ktorý už ale neviete cez `stdio` kontrolovať), musíte volať funkciu `fflush` (alebo uzavrieť súbor, ale potom už dáta nemôžete ďalej zapisovať).

Prílohou prednášky budú príklady demonštrujúce použitie týchto funkcií.

Niektoré ďalšie užitočné funkcie

V prípade, že pracujete s diskovými súbormi, je niekedy užitočné posúvať v nich aktuálnu pozíciu manuálne (napr. najprv zistíte veľkosť vstupného súboru, alokujete pamäť, a načítate údaje). Na tento účel si pozrite ako sa používajú funkcie `rewind`, `ftell` / `fseek`, `fgetpos` / `fsetpos` .

Niekedy si chcete "pozrieť" znak, ktorý má v súbore nasledovať. Keď použijete napr. funkciu `fgetc`, zistíte, že nasleduje číslica a chcete načítať celé číslo cez `fscanf`, tak to už nie je možné, lebo fscanf už číslicu neuvidí (ak nepoužijete `fseek`, čo sa ale vždy nedá). Ak chcete znak "vrátiť" (do vyrovnávacej pamäte), môžete využiť funkciu `ungetc`.

Štandardná knižnica v `stdio.h` špecifikuje aj niektoré funkcie na správu súborov: `rename`, `remove`; a na generovanie dočasných súborov (`tmpfile`, `tmpnam`). Zložitejšie veci v systéme (priečinky, atribúty a pod.) potrebujete riešiť pomocou iných, nie vždy plne prenosných funkcií (ak potrebujete niečo riešiť, dokážete si ich naštudovať, ak ste zvládli štandardnú knižnicu).

Prílohou prednášky budú príklady demonštrujúce použitie niektorých z týchto funkcií.

Poznámka: Funkcie na výpis
Ak robíte funkcie na výpis /napr. štruktúry alebo poľa/ je výhodné spraviť ich univerzálne pridaním parametra typu FILE* (predpokladá sa otvorený výstupný súbor). Príklad (rozšírený z predošlej prednášky):

```
int f_vypis_lokalitu(FILE * f, const Lokalita *lokalita) {
     return fprintf(f, "%s ma suradnice %.5lf %.5lf\n",
          lokalita->nazov,
          lokalita->gps.lat,
          lokalita->gps.lon);
}
```

Ak chcete vypísať niečo na štandardný výstup (konzolu), použijete stdout ako parameter f. Alebo si môžete definovať makro:

```
#define  vypis_lokalitu(X) f_vypis_lokalitu(stdout, X)
```

Vždy keď napíšete iba `vypis_lokalitu(&ba)`, prekladač automaticky doplní `stdout` a zavolá reálne funkciu `f_vypis_lokalitu(stdout, &ba)` (takto funguje aj `printf`, v skutočnosti sa volá `fprintf`).