Standardna ulazno-izlazna biblioteka

Uvod

Može se koristiti na mnogim operativnim sistemima. Napisana od strane Denisa Ričija 1975. godine. Sa ovom bibliotekom se radi pomoću **tokova**. Kada otvaramo fajl preko standardne U/I biblioteke kažemo da smo pridružili tok fajlu.

Standardnom ulazu, izlazu i izlazu za grešku se pristupa preko datotečkih pokazivača **stdin**, **stdout i stderr**, koji su definisani u zaglavlju <stdio.h>. Fajl deskriptori koji odgovaraju ovim tokovima su STDIN_FILENO, STDOUT_FILENO, STDERR_FILENO.

Baferisanje

Cilj je korišćenje minimalnog broja read i write poziva. Baferisanje standardne U/I biblioteke se prilagođava tipu datotečkog pokazivača na koji šalje/sa koga prima podatke. Postoje 3 tipa baferisanja:

1. Režim punog bafera. U/I operacija se izvršava kada se standardni U/I bafer napuni. Fajlovi koji se nalaze na disku su obično puno baferisani od strane U/I biblioteke. Bafer se obično obezbeđuje pozivom funkcije malloc. Termin *flush(prazniti)* označava pisanje iz U/I bafera. Bafer se može sam isprazniti od strane U/I rutina ili ga mi možemo eksplicitno isprazniti pomoću funkcije *fflush*.
2. Linijsko baferisanje. U ovom slučaju, standardna U/I biblioteka vrši U/I kada se dođe do karaktera za novi red na ulazu ili izlazu. Ovo nam omogućava da ispišemo pojedinačne karaktere na izlaz (preko *fputc* funkcije) znajući da će se stvarni U/I izvršiti kada završimo pisanje svake linije. Linijsko baferisanje se koristi za tokove koji se odnose na terminal: standardni ulaz i izlaz, recimo. Kod ovog baferisanja se koristi fiksna veličina bafera, pa će se U/I izvršiti i pre pojave nove linije ako se napuni bafer. Svaki put kada se traži unos preko linijski baferisanog toka ili nebaferisanog toka, svi linijski baferisani izlazni tokovi se prazne.
3. Nebaferovani režim. U ovom slučaju, standardna U/I biblioteka ne baferiše karaktere. Očekuje se da će karakteri koji se štampaju pomoću *fputs* funkcije biti ispisani sto je pre moguće. Standardni izlaz za grešku je nebaferisan, da bi se greške prikazale što je pre moguće.

(PRIMER dup2_b.c sa 3. dvočasa**) –** obrisati prvi *printf*; prvi *printf* u tom slučaju šalje podatke u datoteku, pa se standardnom izlazu priključuje puno baferisanje. Zato se ispis koji bi trebalo da ide u datoteku prvo stavlja u bafer i tek kasnije ispisuje u komandnoj liniji.

Najčešći slučaj je da je standardni izlaz za grešku nebaferisan, svi ostali tokovi su linijski baferisani ako su povezani sa terminalom, inače su puno baferisani. Podrazumevano baferisanje za proizvoljan tok može se promeniti pomoću funkcija **setbuf i setvbuf**. Ove funkcije moraju se pozivati pošto je tok otvoren. Postojalo je više funkcija za postavljanje režima bafera, ali nije iz prvog puta određen najbolji interfejs. Najbolja je *setvbuf* jer je najgeneralnija. Imamo mogućnost da sami zadamo bafer (2. argument, dok je 4. velicina bafera). Ovo retko ima smisla, pa ćemo ovu funkciju pozivati tako sto ćemo zadavati NULL pokazivac za 2. argument, a 0 za 4. argument.

Da bi se tok ispraznio koristi se funkcija **fflush**. Pri korišćenju funkcija standardne biblioteke za ispis na ekran, podaci se smeštaju u bafer dok se on ne napuni ili dok se ne dođe do '\n'. Ako programer želi da se poruka momentalno ispiše na ekran ima 3 opcije:

1) navođenje \n na kraju poruke
2) eksplicitno pražnjenje bafera pomoću funkcije fflush
3) promena tipa baferisanja

Otvaranje toka

*FILE *fopen (const char *path, const char *mode);*
*FILE *fdopen (int fd, const char *mode);*
*FILE *freopen (const char *path, const char *mode, FILE *stream);*

Funkcija *fopen* otvara fajl čija je putanja navedena, *fdopen* prima postojeći fajl deskriptor i pridružuje mu tok. Režim otvaranja mora biti u saglasnosti sa režimom korišćenim pri otvaranju kada je dobijen deskriptor, možemo da suzimo prava ali ne da ih širimo. Ofset za tok se preuzima od trenutnog ofseta za deskriptor i nadalje su takođe povezani. Tokovi se na kraju sami zatvaraju. Funkcija *freopen* otvara fajl na zadatom toku, prvobitno zatvarajući tok ako je otvoren. Možemo na primer da otvorimo tok na standardnom ulazu (kao primer sa prethodnih časova koji koristi dup2 funkciju).

Obrnuto od *fdopen* postoji funkcija *fileno*. Naime, svakom toku je pridružen odgovarajući deskriptor, koga možemo dobiti pozivom ove funkcije. Možemo je iskoristiti ako hocemo da pozovemo *dup* ili *fcnt*l funkcije.

TABELA 5.2 - APUE

Pri pozivu *fdopen* sa otvaranjem fajla za pisanje ne briše se sadržaj fajla jer već to kontroliše prethodno pozvana *open* funkcija. Otvoreni tok se zatvara pomoću **fclose** funkcije.

Čitanje i pisanje sa toka

Funkcije za čitanje

*int getc(FILE *fp);*
*int fgetc(FILE *fp);*
int getchar(void);
getchar <=> getc(stdin)

Ove tri funkcije vraćaju podatak tipa int, da bi u njega mogla da se smesti konstanta EOF, definisana u <stdio.h>. Na Unixu se povratna vrednost može smestiti i u podatak tipa char. Ove funkcije vraćaju istu vrednost bilo da se dođe do EOF ili do greške. Za ispitivanje ovoga koriste se funkcije *ferror* i *feof.* Dva flega se čuvaju za svaki tok u FILE objektu:

- error fleg (fleg greške)
- eof fleg (fleg kraja fajla)

Pošto smo čitali sa toka možemo vratiti karakter koristeći funkciju *ungetc*. Karakteri koji se vraćaju nazad, dobijaju se narednim čitanjima sa toka u obrnutom redosledu od vraćanja. Implementacije su, međutim, obavezne da pamte samo poslednji vraćeni karakter. Nije obavezno da vratimo nazad karakter koji smo pročitali već možemo i neki drugi. Vraćanje EOF nije dozvoljeno. Ponekad moramo da vidimo naredni karakter da bi znali kako da radimo sa trenutnim karakterom (kod parsera recimo).

Funkcije za ispis

*int putc(int c, FILE *fp);*
*int fputc(int c, FILE *fp);*
int putchar(int c);

Čitanje i pisanje linije po linije

*char *fgets(char *s, int size, FILE *stream);*

*int fputs(const char *s, FILE *stream);*
*int puts(const char *s);*

Funkcija *fgets* uvek završava pročitani string sa \0. Ukoliko je pročitao novu liniju, i nju smešta u bafer. *fputs* ne dodaje znak za novi red, dok ga *puts* dodaje. Fukcija *gets* nije navedena i ne treba je koristiti jer kao argument nema dužinu bafera.

Sve funkcije za čitanje u Unix-u koriste *read*, dok sve funkcije za pisanje koriste *write*. Standardna biblioteka neće svakim pozivom *getc* da izvršava read, već ce jednim getc da povuče veći komad u neki svoj interni bafer. Potom se iz tog bafera svaki put učitava karakter po karakter. Ovo je ipak sporije od *read* i *write* jer postoje 2 bafera u kojima se zadržavaju podaci.

Pozicioniranje u toku

*int fseek(FILE *stream, long offset, int whence);*
*long ftell(FILE *stream);*
*void rewind(FILE *stream);*

Funkcija *ftell* vraća broj bajtova od početka fajla, *fseek* se koristi slično kao funkcija *lseek.* Pomoću funkcije *rewind* tok se postavlja na početak fajla.

Formatirani izlaz – printf funkcije.

Formatirani ulaz – scanf funkcije.

Privremeni fajlovi

*int mkstemp(char *template);*
*FILE *tmpfile(void);*

Funkcija *mkstemp* kreira fajl, otvara ga i vraća otvoreni fajl deskriptor za taj fajl. Pri tome zadnjih 6 karaktera stringa moraju biti XXXXXX. NAPOMENA: Treba koristiti fajl deskriptor koji vraća *mkstemp* ili ga zatvoriti pomoću *close* ako nece biti korišćen! Ako se ponovo otvori fajl pomoću *open* onda se nepotrebno koristi veći broj fajl deskriptora od potrebnog. Karaktere XXXXXX funkcija zamenjuje stringom tako da ime fajla bude jedinstveno. Fajl se kreira sa pravima 0600 i otvoreni fajl deskriptor se vraća kao povratna vrednost funkcije.

Funkcija *tmpnam* generiše string koji prestavlja ispravno ime putanje koje nije jednako već postojećem fajlu. Svakim pozivom ove funkcije generiše se drugačije ime putanje. Funkcija *tmpfile* kreira privremeni binarni fajl, koji se automatski briše kada se zatvori datotečki pokazivač ili pri završetku programa. Na Unix-u se binarni fajlovi ne razlikuju od ostalih fajlova.

Napomena: Funkciju *tmpnam* ne treba koristiti jer kreiranje imena fajla i kasnije kreiranje samog fajla ne predstavljaju atomičnu operaciju. Treba korisiti funkciju **mkstemp** i **tmpfile**.

(PRIMER tmp_files)

Standardna tehnika je da se prvo kreira jedinstveno ime putanje i kreir fajla pozivom funkcije *mkstemp,* i potom se fajl unlinkuje. Tako se osiguravamo da će fajl biti obrisan po prekidu programa.

Sistemski fajlovi i njihovi podaci

Password file

struct passwd je definisana u <pwd.h>, a odgovarajući podaci se nalaze u fajlu /etc/passwd.

less /etc/passwd

TABELA 6.1

Starije verzije UNIX-a su na mestu drugog polja čuvale enkriptovanu šifru, danas se ona nalazi na drugom mestu jer nije pogodno da bude vidljiva svima. Podrazumevani shell je /bin/sh. Ukoliko je za shell naveden /dev/null onda to ima za cilj da onemogući korisniku da se uloguje na sistem (ukoliko je korisnik neki uredjaj, npr). Pomoću naredbe **finger** možemo dobiti dodatne informacije o nekom korisniku. Pomoću komande **vipw** administratori mogu da menjaju password fajl.

*struct passwd *getpwuid(uid_t uid);*
*struct passwd *getpwnam(const char *name);*

Funkciju **getpwuid** koristi ls komanda da mapira numerički user ID koji se nalazi u i-nodu u korisničko login ime. Funkcija **getpwnam** koristi program login kada unosimo naše korisničko ime. Obe funkcije vraćaju pokazivač na **struct passwd** koju pune. Ova struktura je inače statička varijabla u funkciji, pa se presnimava svaki put kada pozovemo neku od ovih funkcija. Prethodne funkcije nam daju informacije o korisniku ako znamo njegov broj/ime korisnika.

Ukoliko je potrebno da prođemo kroz ceo password fajl, koristimo sledeće 3 funkcije.

*struct passwd *getpwent(void);*
void setpwent(void);
void endpwent(void);

Funkcija **getpwent** učitava red iz password fajla i smešta ga u odgovarajuću statičku strukturu. Funkcijom **setpwent** vraćamo se na poćetak fajla, dok funkcija **endpwent** zatvara password fajl.

Napomena: kao i *readdir* i ove funkcije vraćaju pokazivače na statičku memoriju, pa nije potrebno alocirati/dealocirati memoriju već samo deklarisati pokazivač.

Implementacija funkcije getpwnam:

```
struct passwd *getpwnam(const char *name) {
   struct passwd  *ptr;
   setpwent();
   while ((ptr = getpwent()) != NULL)
      if (strcmp(name, ptr->pw_name) == 0)
         break;     /* found a match */
   endpwent();
   return(ptr);    /*a ptr is NULL if no match found */
}
```

Shadow passwords

Enkriptovana šifra je kopija korisnikove šifre koja je prošla kroz enkripcioni algoritam u jednom smeru. Pošto je algoritam jednosmeran, ne možemo da pogodimo originalnu šifru iz enkriptovane verzije. Da bi se otežalo pogađanje šifre, sistemi smeštaju enkriptovanu šifru u drugi fajl, koji se obično zove *shadow password file*.

Jedina dva obavezna polja su login ime i enkriptovana šifra. Postoje funkcije koje služe za pristup shadow fajlu.

*struct spwd *getspnam(const char *name);*

*struct spwd *getspent(void); //cita sledeci red shadow fajla*
void setspent(void);
void endspent(void);

Group fajl

gr_mem predstavlja niz pokazivača na korisnička imena koja pripadaju toj grupi. Ovaj niz se završava null pokazivačem.

*struct group *getgrgid(gid_t gid);*
*struct group *getgrnam(const char *name);*

*struct group *getgrent(void); //cita sledeci red iz group fajla*
void setgrent(void);
void endgrent(void); //zatvara group fajl

Supplementary group IDs

U početku razvoja Unixa korisnik je mogao da pripada samo jednoj grupi. Kasnije je omogućeno korisnicima da pripadaju još maksimalno 16 grupa. Svaki put kada se izvršava proces, ne poredi se samo effective group ID sa fajl group ID-om već se i svi dodatni group ID-ovi porede sa fajl group ID-om.

int getgroups(int gidsetsize, gid_t grouplist[]);
int setgroups(int ngroups, const gid_t grouplist[]);
int initgroups(const char *username, gid_t basegid);

Fja **getgroups** popunjava niz *grouplist* sa dodatnim group ID-ovima. Ukoliko je *gidsetsize* 0, funkcija vraća broj dodatnih group ID-ova. Ovo omogućava onome ko poziva funkciju da alocira niz odgovarajuće dužine.

Ostali fajlovi sa podacima

/etc/services – fajl sa servisima koje obezbeđuju različiti mrežni serveri

/etc/protocols – informacije o protokolima

/etc/networks – informacije o mrežama

Interfejs za ove fajlove je isti kao onaj koji koristimo za password i group fajlove. Osnovni princip je da svakom od ovih fajlova mozemo da pristupamo pomoću odgovarajuće verzije 3 funkcije.

1. get funkcija cita naredni red, otvarajući fajl ako je to potrebno. Ove funkcije vracaju pokazivac na strukturu. Pokazivač *null* se vraca kada se stigne do kraja fajla. Većina get funkcija vraća pokazivač na statičku strukturu, pa moramo da je kopiramo ako želimo da je sačuvamo.
2. set funkcija koja otvara fajl, i pozicionira se na početak.
3. end funkcija koja zatvara fajl sa podacima.

Login accounting

Postoje dva fajla u UNIX sistemima. Prvi je **utmp** fajl, koji vodi računa o svim korisnicima koji su trenutno ulogovani. Drugi je **wtmp** fajl, koji vodi računa o svim prijavljivanjima i odjavljivanjima sa sistema. Ovo su binarni fajlovi i nalaze se na lokaciji */var/run.* U oba fajla se smeštaju redovi koji sadrže polja iz strukture

```
struct utmp {
   char  ut_line[8]; /* tty line: "ttyh0", "ttyd0", "ttyp0", ... */
   char  ut_name[8]; /* login name */
   long  ut_time;    /* seconds since Epoch */
 };
```

Pri prijavljivanju puni se ova struktura i upisuje u utmp fajl od strane *login* programa, i ista struktura se nadovezuje na wtmp fajl. Kada se vrši odjavljivanje, red u utmp fajlu se briše i novi red se dodaje u wtmp fajlu. Program **who** čita utmp fajl i prikazuje sadržaj ovog fajla u čitljivoj formi. Postoji i komanda **last** koja prolazi kroz wtmp fajl i štampa sadržaj na ekran.

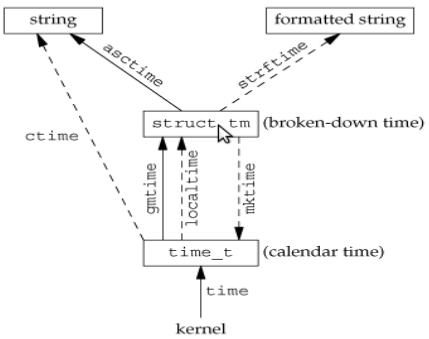
Za prolazak kroz utmp fajl može se koristi funkcija **getutid** i funkcije koje se mogu videti pristupanjem pomoćnoj stranici za ovu funkciju.

Rutine za datum i vreme

Osnovno vreme koje se čuva je broj sekundi koje su protekle od Epohe: 00:00:00 1. jaunara, 1970, Coordinated Universal Time (UTC). Funkcija **time** vraća trenutno vreme i datum. Vreme se vraća i smešta u promenljivu tipa **time_t**, a ukoliko se funkciji prenese pokazivač koji nije NULL, onda se vreme smešta i na lokaciji na koju on pokazuje. Funkcija **gettimeofday** vraća vreme veće tačnosti od funkcije time (do mikrosekundi). Ova funkcija smešta vreme u **struct timeval**, koja čuva sekunde i mikrosekunde:

```
struct timeval {
        time_t tv_sec;   /* seconds */
        long   tv_usec;  /* microseconds */
 };
```

Kada imamo vreme u sekundama od Epohe, pozivamo neku od dostupnih funkcija da prevedemo vreme u oblik pogodan za ljude. Isprekidane linije označavaju da se koristi lokalno, a pune linije da se koristi UTC vreme (standardno vreme po Griniču).



Dve funkcije **localtime** i **gmtime** prevode kalendarsko vreme u vreme razbijeno po vremenskim periodima, smešteno u strukturi **struct tm**:

```
struct tm {      /* a broken-down time */
   int  tm_sec;    /* seconds after the minute: [0 - 60] */
   int  tm_min;    /* minutes after the hour: [0 - 59] */
   int  tm_hour;   /* hours after midnight: [0 - 23] */
   int  tm_mday;   /* day of the month: [1 - 31] */
   int  tm_mon;    /* months since January: [0 - 11] */
   int  tm_year;   /* years since 1900 */
   int  tm_wday;   /* days since Sunday: [0 - 6] */
   int  tm_yday;   /* days since January 1: [0 - 365] */
   int  tm_isdst;  /* daylight saving time flag: <0, 0, >0 */
  };
```

Razlog zbog čega sekunde mogu uzimati vrednost 60 je postojanje prestupnih sekundi. Razlika između prethodne 2 funkcije je da prva pretvara kalendarsko vreme u lokalno vreme, uzimajući u obzir lokalnu vremensku zonu i daylight saving time flag, dok druga pretvara kalendarsko vreme u UTC vreme. Za obrnut smer se koristi funkcija **mktime** koja pretvara lokalno vreme iz struct tm u time_t vrednost.

Funkcije **asctime** i **ctime** vraćaju pokazivač na string koji je sličan izlazu iz **date** komande. Poslednja funkcija je **strftime**, i ona je najkomplikovanija.

```
size_t strftime(char *s, size_t max, const char *format, const struct tm *tm);
```

Ova funkcija vraća broj karaktera smeštenih u niz ukoliko ima mesta, inače 0. Prvi argument je bafer u koji se smešta vreme, drugi je njegova veličina, treći format. Koriste se specifikatori konverzije, kao kod printf-a.

Zadaci

Zadatak 1. Napisati program *ftw_nftw.c* koji pronalazi sve fajlove sa imenom *Makefile* u direktorijumu datim prvim argumentom komandne linije. Pretraga treba da se vrši i u svim njegovim poddirektorijumima. Štampati ime svakog takvog fajla i njegovu veličinu u bajtovima. Koristiti funkciju *ftw* ili *nftw*.

Komentar:

Postoji mnogo definisanih tipova koji predstavljaju modifikacije celih brojeva: off_t, uid_t, time_t. Postoje tipovi intmax_t, uintmax_t koji omogućavaju predstavljanje bilo kog drugog celog broja koji je istog znaka (u prvom slučaju bilo pozitivnog, 0 ili negativnog u drugom samo nenegativnog). Pri štampanju npr. veličine fajla koristićemo eksplicitnu konverziju tipa off_t u intmax_t i štampanje pomoću %jd (označava promenljivu dužinu celog broja). Uključiti zaglavlje stdint.h.

Zadatak 2. Napisati program koji ispisuje imena i prezimena svih studenata na alas-u kojima korisničko ime počinje sa *mi11* i čija se prva 2 slova imena podudaraju sa imenom onog koje napisao program (hard kodirati ta dva slova u sam program). Napomena: program pokrenuti na alasu.

Zadatak 3. Napisati program koji dobija trenutno vreme i štampa ga koristeći *strftime*, tako da izgleda kao podrazumevani izlaz iz komande date.

Zadatak 4. Napisati program koji kao argument komandne linije prima ime direktorijuma, zatim rekurzivno prolazi kroz sve poddirektorijume i briše fajlove koji na kraju svog imena imaju karakter ~ (*backup* fajlovi) i koji nisu modifikovani u zadnjih 30 dana. Voditi računa na koji se način piše regularni izraz da ne bi bili obrisani neki dodatni fajlovi. Savet: može se umesto 30 dana koristiti neki manji vremenski period za testiranje ili se može koristiti *utime* da se napravi neki odgovarajući fajl ako