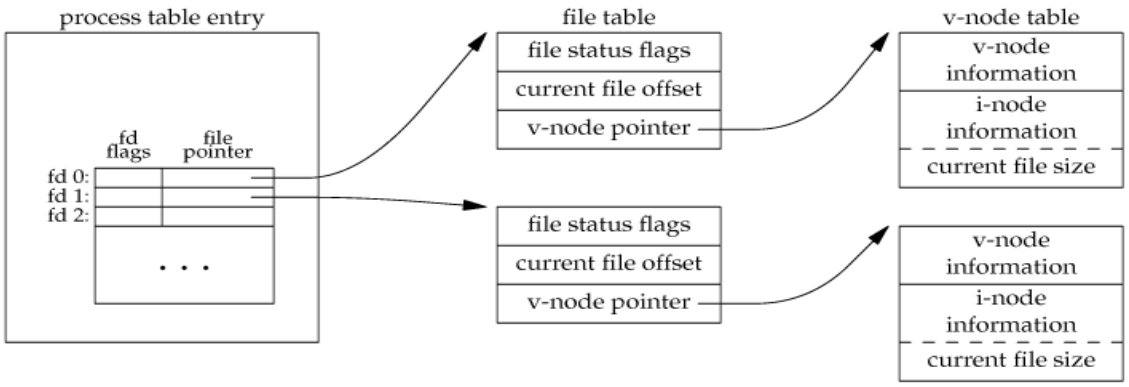
4.8 Deljenje fajlova

UNIX podržava deljenje otvorenih fajlova između različitih procesa. Kernel koristi 3 strukture podataka za otvoreni fajl:

1. Svaki proces ima **odrednicu** u tabeli procesa. U okviru svake odrednice u tabeli procesa nalazi se tabela otvorenih fajl deskriptora. Svakom deskriptoru su pridruženi: flegovi i pokazivač na tabelu fajl odrednica (file table entry).
2. Kernel održava fajl tabelu za sve otvorene fajlove, bez obzira o kom procesu se radi. Svaki red u tabeli sadrži:
   - statusne flegove fajla (read, write, append...)
   - trenutni ofset fajla
   - pokazivač na **v-nod** tabelu zapisa fajla
3. Svaki otvoreni fajl (ili uređaj) ima v-nod strukturu koja sadrži informacije o tipu fajla i pokazivače na funkcije koje rade sa fajlom. Za većinu fajlova v-nod sadrži i **i-nod fajla**. i-nod sadrži ime vlasnika fajla, veličinu fajla, pokazivače na lokacije gde se blokovi podataka fajla nalaze na disku, itd. Jedan deo v-noda je pokazivač na implementacije open, read, itd. za dati fajl. To omogućava da se izvršavaju različite operacije u zavisnosti od tipa fajla.

SLIKA 1: Kernel strukture podataka za otvorene fajlove

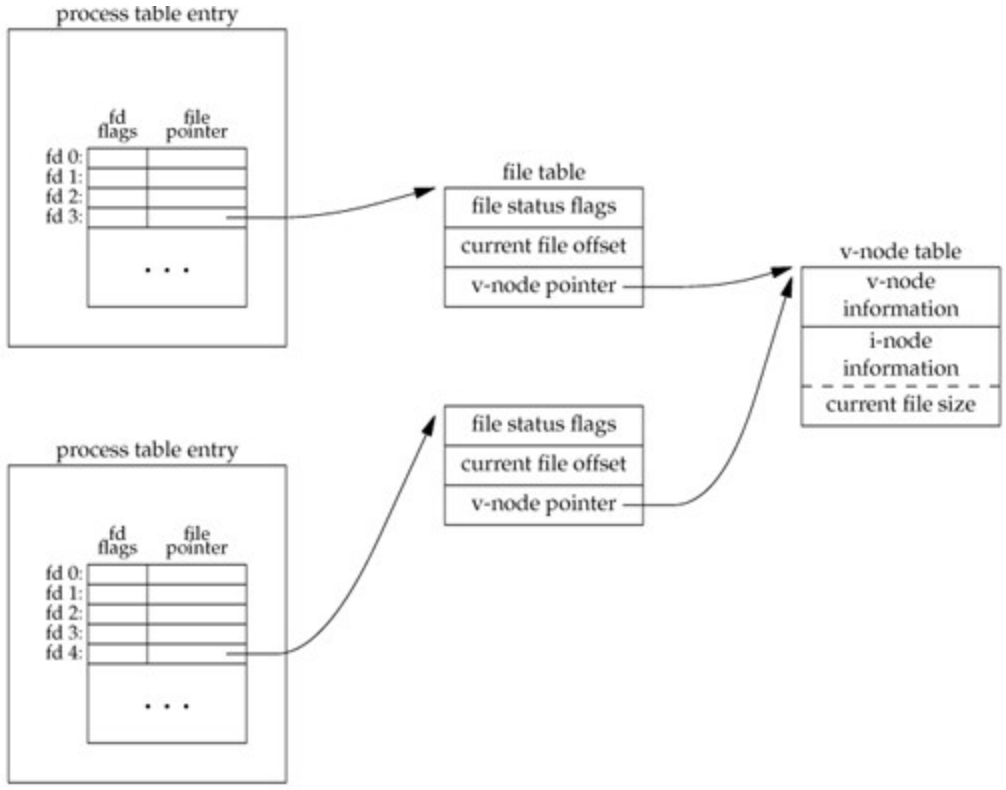


Kada se izvrši funkcija write trenutni ofset fajla se uvećava za broj upisanih bajtova. Ukoliko je povećana veličina fajla, informacija o veličini fajla se ažurira u i-nodu.

Ukoliko je pri otvaranju navedena opcija O_APPEND, onda se pri svakom write-u trenutni ofset fajla postavlja na veličinu fajla iz i-noda.

Ukoliko se pomoću lseek vrši pozicioniranje na kraj fajla, onda se samo ofset postavlja na veličinu fajla iz i-noda.

SLIKA 2: Dva procesa koji imaju isti fajl otvoren



4.9 Atomične operacije

Ranije na Unixu nije postojala opcija O_APPEND za open funkciju, pa se nadovezivanje na kraj vršilo pomoću naredbi:

```
if (lseek(fd, 0, SEEK_END) < 0)
      error_fatal(argv[0]);
if (write(fd, buf, 100) != 100)
      error_fatal(argv[0]);
```

Ove naredbe rade dobro za jedan proces ali kada imamo 2 ili više procesa dolazi do problema. Npr. ako imamo dva procesa A i B koji nadovezuju na isti fajl pomoću gornjih naredbi. Svaki proces ima svoju tabelu fajl odrednica, ali dele istu v-nod tabelu odrednica.
Ako je redosled izvršavanja:

A.lseek
B.lseek
B.write
A.write

onda dolazi do problema koji se zove **trka za resurse (race condition)**. U prethodnom slučaju može se desiti da proces B nešto upiše u fajl a da proces A to presnimi.

Rešenje je da pozicioniranje na kraj fajla i pisanje budu jedna (atomična) operacija. **Atomična operacija** označava da se ili svi koraci operacije izvršavaju ili nijedan. Bilo koja operacija koja zahteva

više od jednog funkcijskog poziva ne može biti atomična, jer uvek postoji mogućnost da kernel preda kontrolu drugom procesu. UNIX obezbeđuje opciju **O_APPEND** pri otvaranju fajla. Ona utiče da kernel pozicionira ofset na kraj fajla pre svakog write poziva.

Funkcije **pread** i **pwrite.**

pread je ekvivalentan pozivu lseek iza koje ide poziv read funkcije, sa dve razlike:

-pread je atomična operacija

-trenutni ofset fajla se ne menja

pwrite je ekvivalentan pozivu lseek iza koje ide poziv write funkcije, sa gore navedenim izuzecima.

Kada su navedene **O_CREAT** i **O_EXCL** opcije kod open funkcije, radi se o atomičnoj operaciji. U ovom slučaju open se neuspešno izvršava ako fajl već postoji. Ukoliko ovo ne bi bila atomišna operacija moglo bi doći do problema: proces bi mogao da proveri da li fajl postoji pa tek onda da ga kreira ukoliko ne postoji. Između ove dve operacije drugi proces bi mogao da kreira fajl sa istim imenom i nesto upiše u njega, što bi prvi proces obrisao.

4.10 Funkcije dup i dup2

```
int dup(int filedes);
int dup2(int filedes, int filedes2);
```

Već postojeći fajl deskriptor se kopira pomoću ovih funkcija. Za fajl deskriptor koji vraća **dup** se garantuje da je najmanji mogući neiskorišćeni broj. Funkcija **dup2** kao drugi argument prima vrednost na kojoj hoćemo da otvorimo fajl deskriptor, ukoliko je ta vrednost već iskorišćena, onda se prvo zatvara fajl s kojim je povezan *filedes2*, pa otvara. Novi fajl deskriptor deli istu tabelu fajl odrednica kao i original. Zbog toga dele iste statusne flegove (read, write, append) a imaju i isti ofset.

Primena dup2 je kada hoćemo da iskoristimo već napisanu biblioteku koja radi ispis na standardni izlaz. Želimo da ispis samo jedne funkcije preusmerimo u neki fajl, a hoćemo da se ostale naše poruke ispisuju na ekran. Otvorimo fajl u koji hoćemo da preusmerimo izlaz, *fd*, i koristimo poziv

```
dup2(fd,1);
```

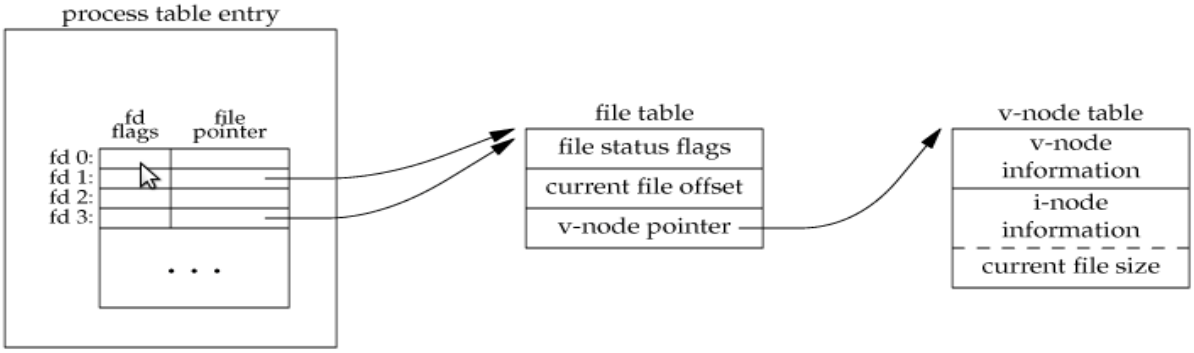
(PRIMER dup2_a)

Ako hoćemo da kasnije ponovo imamo mogućnost da ispisujemo na standardni izlaz, moramo zapamtiti fajl deskriptor koji se odnosi na standardni izlaz. Dakle, sve se upisuje u taj fajl, kad nam to više ne treba, ponovo uradimo dup2.

(PRIMER dup2_b)

NAPOMENA: Pri korišćenju funkcija dup i dup2 može biti neophodno da se koristi *fflush (stdout)* (označava eksplicitno pražnjenje bafera) posle svakog poziva printf funkcije. Razlozi za ovo će biti objašnjeni kasnije u toku kursa.

SLIKA 3: Kernel strukture podatak nakon izvršavanja **dup(1)**



4.11 fcntl funkcija (file control)

```
int fcntl (int fd, int cmd, ... /* arg */ );
```

Ova funkcija menja svojstva već otvorenog fajla. U zavisnosti od drugog argumenta fcntl se koristi u 5 različitih slučajeva (**nama bitni 3. i 5.**):

1. Dupliranje već postojećeg deskriptora (cmd = F_DUFPD), ovo se ređe koristi od kada su uvedene funkcije dup i dup2
2. Za dobijanje/postavljanje fajl deskriptor flegova (cmd = F_GETFD ili F_SETFD; jedini takav fleg je FD_CLOEXEC)
3. Za dobijanje/postavljanje statusnih flegova fajla (cmd = F_GETFL ili F_SETFL) – spominjani kada je rađena funkcija open (O_RDONLY, O_WRONLY, ...)
4. Za dobijanje/postavljanje asinhronog ulazno/izlaznog vlasnistva (cmd = F_GETOWN ili F_SETOWN); prvi služi za dobijanje proces IDa ili proces grup IDa koji prima SIGIO i SIGURG signale.
5. Za dobijanje/postavljanje katanaca (cmd = F_GETLK, F_SETLK ili F_SETLKW)

Makro **O_ACCMODE** je maska koju koristimo da bismo dobili informaciju o režimu pristupa.
(**PRIMER – flags**)
NAPOMENA: Preusmeravanje se ne računa kao argument komandne linije!!!
Navedeno je nekoliko pokretanja sa porukom koja se štampa.
1. Pokreće se program sa argumentom 0, a standardni ulaz se preusmerava na datoteku flags.c.
< označava da se umesto sa standardnog ulaza (našeg unosa u komandnoj liniji) čita iz datoteke flags.c.

```
$ ./flags 0 < flags.c
read only
```

2. Pokreće se program sa argumentom 0, a standardni izlaz se preusmerava na datoteku temp.foo.
> označava da se umesto na standardni izlaz (komandna linija) poruke upisuju u datoteku temp.foo i pri tome se *prethodni sadržaj ove datoteke briše*.

```
$ ./flags 1 > temp.foo
$ cat temp.foo
write only
```

3. Pokreće se program sa argumentom 2, a standardni izlaz za grešku se preusmerava na datoteku temp.foo. Na standardni izlaz za grešku pišemo na primer pomoću fprintf (stderr,"greska");
>> označava da se umesto ispisa na standardni izlaz (komandna linija) poruke upisuju u datoteku temp.foo, pri tome se *vrši nadovezivanje na prethodni sadržaj.*

```
$ ./flags 2 2>>temp.foo
```

```
write only, append
```

4. Pokreće se program sa argumentom 5, a deskriptor 5 se povezuje za čitanje i pisanje sa datotekom temp.foo.
<> označava da se vrši preusmeravanje i ulaza i izlaza.

```
$ ./flags 5 5<>temp.foo
read write
```

5. Pokreće se program sa argumentom 1, a standardni ulaz se preusmerava za čitanje sa datotekom temp.foo. Čeka se unos stringa aaa da bi se završio program.
<< aaa označava da se učitavaju reči dok se ne pojavi reč aaa

```
$ ./flags 1 1<<aaa
read only
```

Probati `./cp <<aaa`

6. Program se pokreće samo sa argumentom 1:

```
$ ./flags 1
read write
```

Možda iznenađujuće, ali sa standardnog izlaza se može čitati. To je zbog toga što se pre pokretanja programa obavljaju naredbe `dup(0), dup(1),` pa će i standardni ulaz, standardni izlaz i standardni izlaz za grešku pokazivati na istu stvar.

Ukoliko hoćemo da postavimo statusne flegove fajla to možemo uraditi sledećom funkcijom:

```
void set_fl(int fd, int flags) /* flags are file status flags to turn on */
{
    int val;

    if ((val = fcntl(fd, F_GETFL, 0)) < 0)
        err_sys("fcntl F_GETFL error");

    val |= flags;       /* turn on flags */

    if (fcntl(fd, F_SETFL, val) < 0)
        err_sys("fcntl F_SETFL error");
}
```

PITANJE: Ako smo otvorili fajl pomoću open za čitanje i pisanje sa uključenim O_APPEND flegom, da li možemo da čitamo na proizvoljnom mestu u fajlu koristeći lseeek? Da li možemo zameniti postojeće podatke u fajlu?

Zadaci:

1. U datoteci studenti.txt nalaze se podaci o studentima. Svaki red je oblika:
redni_broj korisnicko_ime ime prezime trenutna_godina studija
Redni broj je šestocifreni broj (od 000001 do 999999) korisničko ime je na primer mi09011. Napisati program *studenti.c* koji učitava redni broj studenta (bez vodećih nula) i štampa red koji se odnosi na njega. Pretpostavka je da je svaki red fiksne dužine od 30 bajtova (uključujući i znak za novi red), da su redni brojevi u redovima u rastućem poretku i da se ne preskaču. Nije dozvoljeno pretraživati celu datoteku (dakle ne treba koristiti petlju) već je potrebno samo se pozicionirati na traženu poziciju pomoću funkcije lseek.

2. Program dopisi_studenta.c otvara datoteku studenti.txt za čitanje i pisanje i ispisuje na standardni izlaz veličinu datoteke (savet: koristiti lseek). Program potom ne zatvarajući datoteku postavlja fleg za nadovezivanje (koristeći funkciju fcntl). Pozicionirati se na početak datoteke i ispisati nekog novog, fiktivnog korisnika (neki proizvoljan red). Ako je sve urađeno kako treba, bez obzira na pozicioniranje na početak, novi korisnik će biti dodat na kraj fajla, zbog korišćenja O_APPEND opcije. Pokrenuti program više puta da se vidi da se svaki put ispravno izvršava. Pretpostavke o formatu datoteke su isti kao i u prethodnom zadatku.

3. a) Napisati program koji prvo pomoću dup2 funkcije preusmerava standardni izlaz u prethodno otvorenu datoteku izlaz.txt, pomoću printf u tu datoteku upisuje "Upis u datoteku iz procesa (broj procesa)".

b) Prethodni zadatak dopuniti tako da se korišćenjem dup2 ponovo preusmerava standardni izlaz na komandnu liniju. U komandnoj liniji pomoću printf ispisati "Upis u datoteku zavrsen". (savet: fajl deskriptor koji se odnosi na standardni izlaz mora se prvo zapamtiti, pa tek onda raditi preusmeravanje pod a)).

c) Na kraju ponovo pomoću printf funkcije ispisati poruku u datoteku "Opet upis u datoteku".

4. Napisati program koji pomoću funkcija getchar i putchar kopira na standardni izlaz sadržaj datoteke koja se navodi kao prvi agrument komandne linije.

5. Probati da se korišćenjem funkcije *stat* (još nije rađena na vežbama) ispišu i-nod i veličina fajla.