3. Pregled sistema Unix

3.1 Operativni sistem i pomoćne stranice

Striktna **definicija operativnog sistema** je da je to softver koji kontroliše hardverske resurse kompjutera i obezbeđuje okruženje za izvršavanje programa. Zovemo ga i kernel (jezgro). Danas termin operativni sistem označava kernel i drugi integrisani softver koji omogućava udoban rad korisnicima.

Da bi programer mogao da pristupa kernelu obezbeđeni su **sistemski pozivi**. Npr. funkcije koje koristimo u C-u su implementirane tako da koriste sistemske pozive. U Unix-u za svaki sistemski poziv postoji funkcija istog imena u C standardnoj biblioteci. Proces poziva ovu funkciju, koja poziva odgovarajući servis kernela. *Pri pisanju programa možemo sistemske pozive koristiti kao i ostale C funkcije.* Funkcije standardne biblioteke možemo zameniti sistemskim pozivima, dok je obratno obično nemoguće uraditi. Sistemski poziv koji se koristi za alokaciju memorije u UNIX-u je **sbrk**. Funkcija malloc koristi ovaj sistemski poziv.

U Unix-u postoji više sekcija za pomoćne (man) stranice. Sekcije su:

1) Opšte naredbe (komande)
2) Sistemski pozivi
3) Funkcije C standardne biblioteke
4) Specijali fajlovi i drajveri
5) Formati datoteka
6) Igre i skrinsejveri
7) Razno
8) Komande za administriranje sistema

Ukoliko korisnik hoće da dobije informaciji o printf funkciji C standardne biblioteke, potrebno je da u shell-u otkuca *man 3 printf*, a ukoliko hoće da vidi pomoćnu stranicu za read sistemski poziv potrebno je u shell-u otkucati naredbu *man 2 read.* Još neki primeri su *man 4 random, man 5 passwd, man 7 fifo*, *man 7 man-pages, man 8 mount*. Više informacija o nekoj sekciji može se dobiti pomoću komande *man br_sekcije intro,* npr. *man 4 intro.*

Napomena: Ukoliko kompajler javi da neke konstante nisu definisane (npr. STDIN_FILENO) onda je obično najlakše naći i uključiti zaglavlje funkcije koja koristi tu konstantu (npr. unistd.h za read funkciju).

Shell je interpreter komandne linije u kome korisnik može zadavati naredbe. On obezbeđuje interfejs za pokretanje drugih programa. Podrazumevani shell u Linux-u je Bourne-again shell (bash).

SLIKA 1.1

3.2 Unix sistem datoteka

Sistem datoteka je **hijerarhijski organizovan**. Na vrhu hijerarhije nalazi se osnovni (root) direktorijum koji se označava sa /. Direktorijum je datoteka koja sadrži **direkorijumske odrednice** (directory entries). Svaka direktorijumska odrednica sadrži ime datoteke, kao i osobine te datoteke (veličina, tip, prava pristupa). Dve odrednice se kreiraju svaki put kada se kreira novi direktorijum: . i .. Prva odrednica odnosi se na tekući direktorijum a druga na roditeljski (nad) direktorijum. U osnovnom direktorijumu je .. isto kao i .

U svakom trenutku se u shell-u nalazimo u jednom direktorijumu i on se naziva **tekući direktorijum**. Putanja se sastoji iz imena datoteka i predstavlja poziciju datoteke. Može biti **apsolutna** (ukoliko počinje sa /) ili **relativna** (odnosi se na lokaciju u odnosu na tekući direktorijum). Npr. jedna

apsolutna putanja je */home/mi08001* a jedna relativna *../../Zadaci*. Pri logovanju na Unix za tekući direktorijum postavlja se naš polazni (home) direktorijum. Polazni direktorijum za studente na alas-u je /home/mi08___.

3.3 Pokretanja i prevođenje programa

Za prevođenje C programa koristi se GNU C kompajler *gcc*. Npr. prevođenje programa *izracunaj.c* pri kome se izvršnoj datoteci daje ime *izracunaj* vrši se sa

```
gcc -o izracunaj izracunaj.c
```

a pokretanje sa

```
./izracunaj
```

U Unix-u izvršne datoteke mogu imati proizvoljnu ekstenziju. Ukoliko u shell-u ukucamo

```
echo $PATH
```

dobijamo listu svih direktorijuma koji se pretražuju kada je potrebno izvršiti neku naredbu. Ukoliko se u toj listi nalazi i . onda se program izracunaj može pokrenuti i bez navođenja putanje, tj. sa

```
izracunaj
```

3.4 Prvi primer: kopiranje datoteka pomoću read i write.

(PRIMER cp) Sistemski pozivi **read** i **write** se koriste za čitanje i pisanje na Unix-u. Svaka funkcija standardne biblioteke koja čita (piše) poziva read (write). Funkcije napisane u C-u su komfornije za rad i one pokušavaju da učitaju što veću količinu podataka i smeste u svoj interni bafer. Mana standardne biblioteke je nedostatak efikasnosti. Kod sistemskog poziva vrši se prebacivanje u kernel režim rada, kod funkcija standardne biblioteke, deo vremena se provodi u korisničkom režimu rada. Funkcija read odjedanput učita 4kb podataka, dok getc pri prvom pozivu smešta podatke u bafer i onda 4096 puta pristupa tom baferu. Kod čitanja sa diska to ne mora biti bitno, jer najviše vremena protekne na pozicioniranje glave na disku, pa je vreme koje se potroši za funkciju getc zanemarljivo. U nekim drugim slučajevima može postojati veća razlika u efikasnosti.

3.5 Pisanje Makefile-a

Kada se pišu veći projekti (više od jedne C datoteke), postaje zamorno stalno prevoditi sve programe i ponavljati iste naredbe. Moguće je da su od velikog broja datoteka samo jedna ili dve izmenjene pa se ponavljanje prevođenja odnosi samo na one delove koji uključuju ove datoteke. Da bi se ovaj postupak automatizovao, u Unix-u je razvijen alat koji se zove *make*. Skup pravila koja određuju kada je potrebno koje prevođenje izvršiti smeštaju se u datoeku koja se zove *makefile* ili *Makefile*. Najjednostavnije je Makefile pozicionirati u tekući direktorijum, i onda se sva prevođenja koja su zadata ovim fajlom izvršavaju sa

```
make
```

Primer jednostavnog Makefile-a koji prevodi nezavisne programe *prvi.c* i *drugi.c* je:

```
PROGS   = prvi drugi
CC      = gcc
CFLAGS  = -Wall
progs: $(PROGS)
```

```
.PHONY : clean beauty
clean :
        rm -f $(PROGS) *~
beauty :
        indent prvi.c drugi.c
```

Prva 3 reda definisu vrednosti promenljivih PROGS, CC, CFLAGS. Sadržaju promenljive se pristupa npr. sa $(PROGS). Linija 4 određuje fajlove čija se vremena proverevaju kako bi se utvrdilo da li treba izvršiti kompilaciju. Svi fajlovi koji se nalaze sa desne strane ove linije moraju se generisati, zato *Makefile* zna da je prvo potrebno izvršiti kompilaciju da bi se dobio fajl *cp*. Ne bi bilo ispravno zadnju liniju zameniti sa:

```
        indent $(PROGS).c
```

jer bi se uzimajući vrednost promenljive PROGS pokušalo izvršavanje naredbe:

```
        indent prvi drugi.c
```

Bitno: redovi čije su prve reči *rm* i *indent* ispred sebe imaju znak *tab* a ne beline u vidu *space*-a. Prva četiri reda su neophodna da bi Makefile radio kako treba, ostali redovi su opcioni. Ako su i ovi opcioni redovi navedeni, brisanje nepotrebnih (generisanih) fajlova može se uraditi pomoću komande

make clean

a poravnanje teksta unutar fajla sa kodom sa

make beauty

Ako se pri kompaljiranju koristi ovaj Makefile, potrebno je samo umesto *prvi drugi* staviti ime/imena fajlova koji se prevode bez ekstenzije *.c*. Ako postoji samo jedan izvorni fajl, onda pojavljivanje drugog treba ukloniti na svim mestima gde se pojavljuje u Makefile-u. Mogu se navoditi i druge opcije, navedeni Makefile je dovoljan za potrebe ovog kursa. Detaljno uputstvo za pisanje Makefile-a može se naći u skripti *GNU programerski alati.*

3.6 Pisanje shell skriptova

Shell skripte su datoteke koje mogu obavljati različite zadatke i omogućavaju nam da automatski ovaljamo niz radnji. Za pisanje shell skripte može se koristiti proizvoljan editor teksta. Bitno je da imamo pravo izvršavanja na datoteci u kojoj je skript. To se može postići jednom od sledeće dve naredbe:

```
chmod +x ime_skripte
chmod 755 ime_skripte
```

Sama datoteka ne mora imati ekstenziju, ali mi ćemo u našim primerima dodavati ekstenziju .sh.

3.7 Osnovno o procesima

Dva bitna koncepta na Unix-u su rad sa datotekama i rad sa procesima. **Program** je izvršna datoteka koja se nalazi na disku. **Proces** je instanca programa u izvršavanju, tj. kada se program pokrene zovemo ga procesom. Preveden program se nalazi na disku. Ako dva puta pokrenemo program ./cp, to je isti program ali su dve različite instance. Može se reći da je proces program učitan u memoriju.

Ako imamo jedan procesor **samo jedan proces u trenutku** moze da se izvršava. Operativni sistem svakom procesu dodeljuje po jedan mali komad vremena da se taj proces izvršava na procesoru.

Korisniku izgleda da se više programa istovremeno izvršava: Web čitac, editor teksta...

Svaki proces ima jedinstven broj koji se naziva **proces ID**. To je uvek nenegativan ceo broj.
getpid() - sistemski poziv služi za očitavanje proces IDa.

(PRIMER pid) Fja getpid() vraca ID procesa iz koga je pozvana.
getuid() sistemski poziv vraća ime onoga u čije ime se proces izvršava.
getgid() sistemski poziv vraća u ime čije grupe se proces izvršava.
Procesi su kao i datoteke, poređani u hijerarhiju. *Proces moze biti kreiran samo od strane drugog procesa. Onaj koji kreira se zovi roditeljski proces a kreiran proces dete proces.*
getppid() - proces ID roditeljskog procesa.

3.8 Obrada grešaka

Kada dođe do greške, najčešće funkcija vraća negativnu vrednost a promenljiva *errno* definisana u errno.h se postavlja na vrednost koja daje dodatne informacije. (U knjizi APUE na više mesta se nalazi greška, kada se navodi da funkcija vraća vrednost 1 umesto da se navede da vraća -1 u slučaju greške). Svaka od tih vrednosti je definisana kao konstanta koja počinje slovom E. Npr. errno može uzeti vrednost EACCES (to je najčešće vrednost 13) što je znak za problem pristupanja fajlu (najčešće zbog nedovoljnih prava pristupa). Sa *man 3 errno* može se dobiti spisak svih ovakvih konstanti.
Bitno: Vrednost promenljive errno se ne resetuje kada se neka funkcija uspešno izvrši. Zato njenu vrednost treba proveravati samo u slučaju kada se zna da je došlo do greške. Dve funkcije C standardne biblioteke se koriste za štampanje poruka o greškama. Prva se zove **strerror** i u zavisnosti od vrednosti konstante koju prima vraća opis greške u obliku niske koja se odnosi na tu konstantu. Druga se zove **perror** i na osnovu vrednosti promenljive errno štampa poruku koju prima kao argument zajedno sa opisom greške na standardni izlaz za greške.

3.9 Primitivni sistemski tipovi

Postoji mnogo tipova podataka na Unixu koji predstavljaju suženje celobrojnih tipova na određeni skup vrednosti. To su npr. tipovi *off_t, clock_t, mode_t, off_t, size_t, ssize_t,* itd. Veći spisak ovih tipova može se pronaći u knjizi APUE, sekcija 2.8. Promenljive ovih tipova se mogu porediti sa celobrojnim konstantama bez problema, ali pri štampanju vrednosti ovih promenljivih kompajler može javiti upozorenja. Da bi se izbegla upozorenja pri štampanju treba uključiti zaglavlje **stdint.h**, koristiti specifikator %jd u printf funkcijama i eksplictno konvertovati podatak u **intmax_t** (za celobrojne tipove) ili **uintmax_t** (za celobrojne nenegativne tipove) pri štampanju. Primeri:

```
printf (“%jd“, (intmax_t)promenljiva);
printf (“%ju“, (uintmax_t)promenljiva);
```

4. Čitanje i pisanje datoteka

4.1 Uvod

Većina U/I (ulazno/izlaznih) operacija u Unixu može da se izvrši pomoću 5 funkcija: **open, read, write, lseek i close**. Ove funkcije nazivaju se **nebaferisanim ulazom/izlazom**, za razliku od funkcija standardne biblioteke (npr. printf, fopen...). Termin nebaferisan označava da svaki poziv read i write uzrokuje sistemski poziv u kernelu.

U kernelu, svim otvorenim fajlovima se pristupa pomoću **fajl deskriptora**. Kada otvaramo ili

kreiramo fajl, kernel vraća fajl deskriptor procesu. Fajl deskriptor 0 se na početku izvršavanja svakog programa dodeljuje standardnom ulazu, 1 se povezuje sa standardnim izlazom, a 2 sa standardnim izlazom za grešku. Ovim brojevima redom odgovaraju konstante STDIN_FILENO, STDOUT_FILENO, STDERR_FILENO definisane u zaglavlju <*unistd.h*>. U ranijim verzijama Unix-a postojalo je ograničenje za maksimalan broj otvorenih fajl deskriptora, danas je taj broj skoro neograničen.

4.2 open funkcija

Pomoćnu stranicu možemo videti pomoću *man 2 open.* Postoje dve verzije funkcije, jedna sa 2 a druga sa 3 argumenta. Prvi argument je putanja datoteke koju otvaramo (bilo apsolutna ili relativna). Drugi argument je **tačno jedna** od opcija

O_RDONLY – otvoriti samo za čitanje
O_WRONLY – otvoriti samo za pisanje
O_RDWR – otvoriti i za čitanje i za pisanje

kao i neke od dodatnih opcija:

O_APPEND – svakim pozivom write upisivati na kraj fajla
O_CREAT – kreirati fajl ako ne postoji, zahteva se i treći argument, mod, koji definiše prava pristupa novog fajla
O_EXCL – generiše grešku ako je opcija O_CREAT navedena i fajl vec postoji
O_TRUNC – ako fajl postoji i uspešno je otvoren za pisanje, smanjiti njegovu dužinu na 0

Postoji još opcija koje se mogu navesti pri pozivu funkcije open, ali su one manje bitne. Ova funkcija **vraća fajl deskriptor dodeljen navedenom fajlu**, tj. najmanji ceo broj koji prethodno nije iskorišćen za deskriptor. U slučaju greške vraća se -1.

4.3 creat funkcija

Ova funkcija služi za kreiranje novih fajlova i ekvivalentna je sa

open (pathname, O_WRONLY | O_CREAT | O_TRUNC, mode);

Mod označava prava pristupa fajlu. Bitan nedostatak ove funkcije je što se fajl otvara samo za pisanje. Zato ako želimo da i čitamo i pišemo, moramo da pozivamo redom creat, close pa open. Zbog toga je **lakše koristiti open** funkciju, koja ima veću funkcionalnost od creat.

Napomena: U gornjem pozivu open briše se sadržaj fajla ali se **ne kreira novi fajl**. To znači da ukoliko je fajl već postojao neće mu biti dodeljena nova prava pristupa. Ukoliko želimo da mu dodelimo nova prava, moramo prvo da pozovemo funkciju *unlink* pa tek onda open.

4.4 close funkcija

Funkcija kao jedini argument prima fajl deskriptor. Ovom funkcijom se zatvara taj fajl deskriptor, tako da se više ne odnosi na fajl. Kada se proces završi svi fajl deskriptori se automatski zatvaraju.

4.5 lseek funkcija

Svaki otvoreni fajl u procesu ima pridružen **trenutni ofset fajla** (current file offset), uobičajeno nenegativan broj koji sadrži broj bajtova od početka fajla. Ovaj broj se postavlja na 0 kada se fajl

otvori, izuzev kada je ukljucena opcija O_APPEND (u tom slučaju postavlja se na veličinu fajla). Read i write pri čitanju povećavaju ovu vrednost. TOF se može postaviti funkcijom lseek a deklaracija ove funkcije je:

```
off_t lseek(int filedes, off_t offset, int whence);
```

Prvi argument je fajl deskriptor. Treći argument može biti jedna od 3 vrednosti:
SEEK_SET – ofset fajla se postavlja na *offset* bajtova od pocetka fajla
SEEK_CUR – ofset fajla se postavlja na zbir TOF i *offset. Offset* može biti pozitivan ili negativan
SEEK_END – ofset fajla se postavlja na zbir veličine fajla i *offset*. *Offset* moze biti pozitivan ili negativan
Uspešan poziv lseek vraća novi TOF. Kako možemo izračunati TOF? Kako se možemo pozicionirati 5 bajtova ispred kraja fajla? Kako se možemo pozicionirati na sredinu fajla?

TOF se može izračunati pomoću poziva:

```
off_t currpos;
currpos = lseek(fd, 0, SEEK_CUR);
```

Funkcija lseek **ne radi nikakve U/I operacije**, već samo postavlja novi TOF.

(PRIMER – hole)
Ne odvajaju se blokovi memorije za prostor između starog kraja fajla i lokacije na kojoj počinjemo da pišemo.

```
ls -l file.hole
od -c file.hole
```

postmatramo sadržaj fajla, -c označava da se sadržaj prikazuje preko karaktera.
Napomena: za korišćenje ovih funkcija neophodno **uključiti sva zaglavlja** navedena u man stranici za odgovarajuću funkciju a ne samo jendo!!! Ovo može biti slučaj još kod nekih funkcija.

4.5 read funkcija

```
ssize_t read (int fd, void *buf, size_t count);
```

ssize_t i size_t su primitivni sistemski tipovi. Prvi može biti i pozitivan i negativan ceo broj a drugi može biti samo pozitivan ceo broj. Možemo ih koristi kao i druge cele brojeve i umesto njih koristiti tip int.

Funkcija read čita sa fajl deskriptora *fd* maksimalno *count* bajtova i smešta ih u prostor na koji pokazuje pokazivač *buf*. Ukoliko se pokušava sa čitanjem 100 bajtova a 30 je ostalo u fajlu – u prvom pozivu read vraća se 30, u drugom 0. Read čita sa trenutnog ofseta fajla, a pre uspešnog povratka ofset se povećava za broj pročitanih bajtova.

Tabela 1. Vremena čitanja za različite veličine bafera programa cp

BUFFSIZE	User CPU (seconds	System CPU (seconds)	Clock time	#loops
1	124.89	161.65	288.64	103,316,352
2	63.10	80.96	145.81	51,658,#176
4	31.84	40.00	72.75	25,829,088
8	15.17	21.01	36.85	12,914,544
16	7.86	10.27	18.76	6,457,272
32	4.13	5.01	9.76	3228636
64	2.11	2.48	6.76	1,614,318
128	1.01	1.27	6.82	807,159
256	0.56	0.62	6.80	403,579
512	0.27	0.41	7.03	201,789
1,024	0.17	0.23	7.84	100,894
2,048	0.05	0.19	6.82	50,447
4,096	0.03	0.16	6.86	25,223
8,192	0.01	0.18	6.67	12,611
16,384	0.02	0.18	6.87	6,305
32,768	0.00	0.16	6.70	3,152
65,536	0.02	0.19	6.92	1,576
131,072	0.00	0.16	6.84	788
262,144	0.01	0.25	7.30	394
524,288	0.00	0.22	7.35	198

NAPOMENA: Funkcije read i write vraćaju -1 u slučaju greške (**u knjizi greškom stoji 1**)

4.5 write funkcija

```
ssize_t write (int fd, const void *buf, size_t count);
```

Funkcija write pokušava da u fajl povezan sa fajl deskriptorom *fd* upiše *count* bajtova bafera *buf*. Ukoliko se povratna vrednost ne poklapa sa vrednošću trećeg argumenta, došlo je do greške. Najcešći uzroci greške su popunjavanje prostora na disku ili prekoračenje dozvoljene veličine fajla u

datom procesu.

Upisivanje kreće od trenutnog ofseta fajla izuzev u slučaju kada je uključena opcija O_APPEND pri otvaranju fajla. U tom slučaju se fajl ofset postavlja na kraj fajla pre svakog pisanja. Posle svakog uspešnog write poziva, ofset fajla se povećava za broj upisanih bajtova.

Zadaci:

1. Napisati program hello_world.c koji štampa poruku "Hello, world!" na standardni izlaz. Napisati shell skriptu koja prvo poravnava kod programa u Kernighan i Ritchie stilu koristeći komandu *indent*, vrši prevođenje programa, i na kraju pokreće program. Napisati zatim i Makefile za ovaj program, pokrenuti ga i proveriti da li ispravno radi.

2. Napisati program spoj_tabele.c koji prvo prepisuje sadržaj datoteke primera.txt u datoteku tabele.txt, a zatim na ovu datoteku nadovezuje i sadržaj datoteke calco.txt. Koristiti read i write sistemske pozive.

3. U datoteci brojevi.txt se u svakom redu nalazi trocifreni broj kao i pozicija na kojoj se on nalazi kada se sortiraju ti brojevi. Napisati program ispisi_sortirano.c koji redom učitava brojeve iz te datoteke i upisuje ih u datoteku sortirani_brojevi.txt. Kada se učita broj on se automatski upisuje na za njega odgovarajuću poziciju korišćenjem funkije lseek. Svaki red novokreirane datoteke sadrži samo trocifreni broj. Koristiti *read* i *write* sistemske pozive i činjenicu da se zna tačan broj bajtova za svaki red (može se pretpostaviti da je pozicija broja jednocifren broj).

Ako je datoteka brojevi.txt:
333 3
777 7
444 4
111 1
000 0
222 2
666 6
555 5

onda datoteka sortirani_brojevi.txt treba da sadrži:
000
111
222
333
444
555
666
777

4. Napisati program zameni_karakter.c koji kao argumente komandne linije prima ime datoteke, karakter i korak (ceo broj). Program u datoteci na svim pozicijama čiji indeks je deljiv korakom menja postojeći karakter karakterom iz komandne linije. Koristiti funkcije write i lseek. Npr. ako se program pokrene sa:
./zameni_karakter dat.txt A 4

onda na pozicijama 0, 4, 8, itd. u fajlu dat.txt treba postaviti karakter A.