

Verifikacija softvera - Vežbe

Ivan Ristović

2026-01-01

Sadržaj

Predgovor	1
I Osnove	3
1 Okruženje izvršavanja	5
1.1 Strategije prevođenja programa	5
1.2 Povezivanje (linkovanje)	6
1.2.1 Dinamičko povezivanje	6
1.2.2 Statičko povezivanje	6
1.2.3 Poređenje tipova povezivanja	7
1.2.4 Primer	7
2 Pregled Linux fajl sistema	9
2.1 Glavni direktorijumi	9
2.2 /proc: procesi i informacije o sistemu	10
2.3 /dev: uređaji i specijalni fajlovi	11
2.4 /tmp: privremeni fajlovi	12
2.5 Tipovi fajlova	13
2.6 Korisne komande za istraživanje	14
3 Dijagnoziranje problema	15
3.1 Metodologija dijagnoziranja problema	15
3.2 Alati za analizu performansi	16
3.2.1 uptime	16
3.2.2 top / htop	17
3.2.3 ps	17
3.2.4 vmstat	18
3.2.5 iostat	18
3.2.6 swapon	19
3.2.7 mpstat	19
3.2.8 lsof	20
3.2.9 pidstat	20

3.2.10	strace	21
3.2.11	Analiza mreže	23
3.2.12	System Activity Reporter (sar)	25
3.3	Kako birati alate tokom analize	26
3.4	Laboratorijski primeri	27
3.4.1	Pregled primera	27
3.4.2	lab001: CPU petlja	28
3.4.3	lab002: čekanje na konekciju	28
3.4.4	lab003: čitanje nula bajtova	30
3.4.5	lab004: kratkotrajni child procesi	32
3.4.6	lab005: sinhroni upisi na disk	34
3.4.7	lab006: više CPU vrućih funkcija	36
3.4.8	lab007: proces uglavnom spava	37
3.4.9	lab008: disk latencije sa izuzecima	38
3.4.10	lab013: rekurzija i stek	40
4	Efikasno korišćenje terminala	43
4.1	Osnovne komande	43
4.2	Pregledanje i obrada teksta	44
4.3	Pronalaženje fajlova: find	45
4.4	Procesi i periodično izvršavanje	45
4.5	Pozadinski procesi	46
4.6	Direktorijumi i stek putanja: pushd / popd	46
4.7	Preusmeravanje i povezivanje komandi	46
4.8	Skraćenice i posebne promenljive (Bash)	47
4.9	Bash prečice sa tastature	48
4.10	Mreža, arhive i pomoćne komande	48
4.11	Niz komandi	49
5	Efektivno pisanje Bash skripti	51
5.1	Zaglavlje	51
5.2	Argumenti i zamene parametara	51
5.2.1	Zamene	52
5.2.2	Zamena komande i aritmetička ekspanzija	52
5.3	Provera argumenata i parsiranje opcija	53
5.3.1	Duge opcije ručnim upravljanjem	53
5.3.2	Kratke i duge opcije zajedno	53
5.4	Uslovi i status komande	53
5.5	Funkcije, nizovi i petlje	54
5.5.1	Funkcije i statusi	54
5.5.2	Nizovi	54
5.5.3	Bezbedna obrada imena datoteka	54
5.6	Redirekcija ulaza i izlaza	54
5.6.1	“Here”-dokument i “here”-niska	55
5.7	Cevovodi i tee	55
5.8	Privremene datoteke i čišćenje	56

5.9 Signali	56
5.10 Pozadinski procesi	56
5.10.1 Pokretanje više zadataka i provera statusa	57
5.10.2 Ograničenje broja paralelnih zadataka	57
5.10.3 Zaustavljanje pozadinskih procesa tokom čišćenja	58
5.11 Potprocesi i zamena procesa	58
5.12 Primer: brojanje podudaranja	58
5.13 Zadaci za vežbu	58

Predgovor

Kolekcija materijata sa vežbi za kurs Verifikacija softvera na Matematičkom fakultetu, Univerzitetu u Beogradu.

Materijali prate praktičan tok rada: od dijagnoziranja simptoma i debugovanja, preko testiranja i profajliranja, do simboličkog izvršavanja, proveravanja modela i formalne verifikacije. Svaka tema je organizovana kroz kratko objašnjenje, komande, očekivane nalaze i male primere koji se nalaze uz odgovarajuće poglavlje.

Pokrivene su sledeće celine:

- **Dijagnozovanje problema:** osnovni Linux alati, posmatranje procesa i analiza ponašanja programa metodom crne kutije.
- **Debugovanje:** *gdb*, *Qt Creator*, *core dump* analiza, *watchpoint*-i, niti, prikazivanje na proces, *Java JDWP*, agenti i *native/AOT* izvršive verzije.
- **Refaktorisanje:** prepoznavanje lošeg dizajna, izdvajanje odgovornosti i primeri kroz *C++*, *Java* i *C#* kod.
- **Testiranje jedinica koda:** *QtTest*, *JUnit*, *xUnit*, *NUnit*, pokrivenost koda, *lcov*, *JaCoCo* i *.NET* alati za pokrivenost.
- **Testiranje pomoću objekata imitatora:** ručni imitatori, *Mockito*, *Mog* i testiranje koda koji zavisi od servisa, vremena ili baze podataka.
- **Fuzz testiranje:** *libFuzzer*, *tmux* tokovi rada, *Jazzer* za Java kod i *AFL++* na realnijem *C/.C++* primeru.
- **Profajliranje instrumentacijom:** *Valgrind* alati kao što su *Memcheck*, *Massif*, *DHAT*, *Cachegrind*, *Callgrind*, *Helgrind* i *DRD*, kao i *gprof* i *LLVM XRay*.
- **Profajliranje uzorkovanjem:** *perf*, flame graph tok rada, *Intel VTune Profiler* i *Java async-profiler*.
- **eBPF:** model izvršavanja, *verifier*, mape, *bpftime*, *BCC* alati, sistemsko profajliranje i ograničenja dozvola.

- **KLEE**: simboličko izvršavanje, generisanje testova, pronalaženje grešaka, rad sa bibliotekama i heuristike pretrage.
- **CBMC**: proveravanje svojstava C programa, odmotavanje petlji, pretpostavke, postuslovi, *harness* funkcije i konkurentni primeri.
- **Clang alati**: *Clang* statički analizator, *clang-tidy*, sanitizeri, pojedinačni primeri grešaka i analiza projekata.
- **Dafny**: specifikacije, pre/postuslovi, invarijante petlji i dokazi korektnosti za male algoritme.
- **Instalacije**: kratke beleške za alate i pakete koji su potrebni za lokalno pokretanje vežbi.

Deo I

Osнове

Glava 1

Okruženje izvršavanja

Pre izbora debugerskog pristupa treba znati kako program nastaje i gde se izvršava. Na osnovu toga, možemo odlučiti koje tehnike i alate možemo primeniti i koje informacije možemo dobiti. Na primer, C program kompiliran unapred možda ne sadrži simbole za debugovanje dok JVM koji izvršava Java program već ima sve neophodne informacije

1.1 Strategije prevođenja programa

Interpreter, JIT (Just-In-Time) i AOT (Ahead-Of-Time) strategije različito mapiraju izvorni kod, međukod i mašinski kod. To utiče na breakpoint-e, simbole, stack trace, profajliranje, instrumentaciju i reprodukciju grešaka.

[+] označava povoljan aspekt, [/] kompromis ili oprez, a [−] nepovoljan aspekt za debugovanje, profajliranje ili isporuku.

Interpreter	JIT	AOT
[/] Tumači kod tokom izvršavanja	[/] Tumači, zatim prevodi vruće putanje	[/] Prevodi pre pokretanja
[+] Brz razvojni ciklus	[+] Brz razvojni ciklus	[−] Sporiji ciklus zbog punog prevođenja
[/] Izvršava samo posećene putanje	[/] Kreće od bajtkoda ili međukoda	[/] Optimizuje ceo poznati program
[+] Greške se vide tokom izvršavanja	[/] Runtime menja kod tokom rada	[−] Optimizacije mogu zakloniti uzrok
[+] Prenosiv uz isti interpreter	[+] Prenosiv uz isti runtime	[−] Vezan za ciljnu platformu
[−] Sporije pokretanje	[−] Sporije pokretanje i zagrevanje	[+] Brzo pokretanje

Interpreter	JIT	AOT
[−] Niže vršne performanse	[+] Visoke posle zagrevanja	[+] Visoke uz dobre optimizacije
[/] Profajl meri i interpreter	[+] Runtime često ima dobru podršku	[−] Teže zbog inlining-a i simbola
[+] Laka instrumentacija	[+] Runtime hook-ovi pomažu	[−] Instrumentacija pre prevođenja ili binarno
[+] Dinamičke osobine su prirodne	[+] Dinamičke osobine su prirodne	[−] Dinamika je obično ograničena
[−] Veći memorijski otisak	[−] Runtime, JIT kod i metapodaci	[+] Manji memorijski otisak
[−] Potreban interpreter/runtime	[−] Potreban runtime	[+] Runtime nije nužan kod statičkog povezivanja

1.2 Povezivanje (linkovanje)

Prevodilac prevodi izvorni kod u objektne fajlove, a linker ih povezuje sa potrebnim bibliotekama. To povezivanje može biti **statičko** ili **dinamičko**.

1.2.1 Dinamičko povezivanje

Program se pri pokretanju povezuje sa bibliotekama koje su zasebni fajlovi na sistemu. Izvršni fajl je obično manji, a više programa može da koristi istu biblioteku.

```
gcc main.c -o program
./program
```

Zavisnosti programa mogu da se pregledaju pomoću `ldd`:

```
ldd ./program
```

Ako potrebna biblioteka nedostaje ili nije dostupna verzija koju program očekuje, program možda neće moći da se pokrene.

1.2.2 Statičko povezivanje

Biblioteke se ugrađuju u izvršni fajl. Program može biti veći, ali manje zavisi od dostupnosti tih biblioteka u sistemu.

```
gcc -static main.c -o program-static
./program-static
```

Statičko povezivanje zahteva da statičke verzije biblioteka budu instalirane. Ako nisu dostupne, linker može da prijavi grešku.

1.2.3 Poređenje tipova povezivanja

Osobina	Dinamičko povezivanje	Statičko povezivanje
Veličina izvršnog fajla	Obično manja	Obično veća
Biblioteke pri pokretanju	Potrebne su u sistemu	Uglavnom su ugrađene u program
Zajednička upotreba biblioteka	Više programa može da deli istu biblioteku	Svaki program sadrži svoju kopiju
Ažuriranje biblioteka	Ažurirana sistemska biblioteka može koristiti više programa	Ažuriranje biblioteke obično zahteva ponovno povezivanje programa

1.2.4 Primer

main.c:

```
#include <stdio.h>

int main(void) {
    printf("Hello, world!");
    return 0;
}
```

Dinamičko povezivanje:

```
gcc main.c -o hello
file ./hello
ldd ./hello
```

Statičko povezivanje:

```
gcc -static main.c -o hello-static
file ./hello
```

Opcija `-static` traži statičko povezivanje, uspeh zavisi od toga da li su dostupne odgovarajuće statičke biblioteke na sistemu. U ovom jednostavnom primeru, to će biti slučaj.

```
`<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6IjAwX2Jhc2ljcy8wMF9saW51eCJ9 -->`{=html}
```

```
````{=html}
```

```
<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6IjAwX2Jhc2ljcy8wMF9saW51eCIsImJvb2tJdGVtVHlwZSI6Im
```



## Glava 2

# Pregled Linux fajl sistema

Korenski direktorijum Linux sistema označava se znakom `/`. Svi fajlovi, direktorijumi i priključeni sistemi fajlova nalaze se ispod njega. Njegov sadržaj može da se razlikuje između distribucija i sistema.

Pregled sadržaja:

```
ls -lah /
```

### 2.1 Glavni direktorijumi

Putanja	Kratak opis
<code>/bin</code>	Osnovne korisničke komande, poput <code>ls</code> i <code>cp</code> . Na mnogim savremenim sistemima simbolička je veza ka <code>/usr/bin</code> .
<code>/boot</code>	Fajlovi potrebni za pokretanje sistema, kao što su kernel i konfiguracija pokretačkog programa.
<code>/dev</code>	Specijalni fajlovi koji predstavljaju uređaje i sistemske interfejse.
<code>/etc</code>	Sistemska konfiguracija, uglavnom u obliku tekstualnih fajlova.
<code>/home</code>	Lični direktorijumi korisnika, na primer <code>/home/ana</code> .
<code>/lib</code> , <code>/lib64</code>	Osnovne biblioteke i moduli potrebni sistemskim programima. Na mnogim sistemima povezani su sa direktorijumima u <code>/usr</code> .

Putanja	Kratak opis
<code>/media</code>	Uobičajeno mesto za automatski priključene prenosive uređaje, poput USB diskova.
<code>/mnt</code>	Često se koristi za privremeno ručno priključivanje sistema fajlova.
<code>/opt</code>	Dodatni, često nezavisno instalirani softverski paketi.
<code>/proc</code>	Virtuelni sistem fajlova sa informacijama o procesima i kernelu.
<code>/root</code>	Lični direktorijum korisnika <code>root</code> ; nije isto što i <code>/</code> .
<code>/run</code>	Privremeni podaci o aktivnom sistemu, procesima i servisima; obično se pravi pri pokretanju sistema.
<code>/sbin</code>	Sistemske administrativne komande. Na mnogim savremenim sistemima simbolička je veza ka <code>/usr/sbin</code> .
<code>/srv</code>	Podaci koje pružaju servisi, na primer sadržaj web servera; upotreba zavisi od sistema.
<code>/sys</code>	Virtuelni sistem fajlova za uređaje, drajvere i informacije o kernelu.
<code>/tmp</code>	Privremeni fajlovi; često se brišu pri ponovnom pokretanju ili automatskim čišćenjem.
<code>/usr</code>	Većina korisničkih programa, biblioteka, dokumentacije i drugih instaliranih fajlova.
<code>/var</code>	Podaci koji se menjaju tokom rada sistema, poput logova, keša i redova za štampanje.

## 2.2 `/proc`: procesi i informacije o sistemu

`/proc` nije običan direktorijum sa trajno sačuvanim fajlovima. To je virtuelni sistem fajlova koji kernel prikazuje kao fajlove i direktorijume. Sadržaj se uglavnom generiše pri čitanju, pa prikazana veličina fajla ne mora da odgovara količini podataka sačuvanih na disku.

Putanja	Sadržaj
<code>/proc/cpuinfo</code>	Informacije o procesorima i njihovim mogućnostima.

Putanja	Sadržaj
/proc/meminfo	Podaci o memoriji, uključujući slobodnu i dostupnu memoriju.
/proc/uptime	Vreme rada sistema i vreme provedeno u mirovanju.
/proc/loadavg	Prosečno opterećenje sistema u nekoliko vremenskih perioda.
/proc/mounts	Trenutno priključeni sistemi fajlova.
/proc/PID/	Informacije o procesu čiji je identifikacioni broj PID.
/proc/self/	Informacije o procesu koji trenutno čita tu putanju.

Primeri:

```
cat /proc/uptime
head /proc/meminfo
grep -E 'model name|processor' /proc/cpuinfo
```

U korenu /proc nalaze se direktorijumi procesa, često imenovani njihovim PID brojevima:

```
ls /proc
```

Ako je PID procesa 1234, njegove informacije mogu biti dostupne pod /proc/1234/:

```
ls /proc/1234
cat /proc/1234/status
```

PID procesa može da se pronađe komandom `pgrep`:

```
pgrep -a firefox
```

Neki fajlovi u /proc mogu da se menjaju upisivanjem, ali takve promene utiču na ponašanje kernela. Upisivanje u te fajlove zahteva poznavanje njihove namene.

## 2.3 /dev: uređaji i specijalni fajlovi

U /dev se nalaze specijalni fajlovi koje programi koriste za pristup uređajima ili određenim sistemskim funkcijama. To nisu uobičajeni fajlovi čiji se sadržaj jednostavno čita sa diska. Dostupni uređaji i nazivi zavise od računara.

Putanja	Uobičajena namena
/dev/null	Odbacuje sve što se u njega upiše; čitanje odmah daje kraj ulaza.
/dev/zero	Daje niz bajtova čija je vrednost nula.
/dev/random	Interfejs za dobijanje nasumičnih bajtova.
/dev/urandom	Interfejs za dobijanje kriptografski sigurnih nasumičnih bajtova.
/dev/tty	Terminal povezan sa trenutnom sesijom, ako postoji.
/dev/pts/	Virtuelni terminali, na primer oni koje koriste terminalski emulatori i SSH sesije.
/dev/sda, /dev/nvme0n1	Mogući nazivi diskova; stvarni nazivi zavise od uređaja i sistema.
/dev/loop0	Virtuelni blok-uređaj koji može da predstavlja fajl kao disk.

Primeri:

```
printf 'Ova poruka se ne prikazuje\n' > /dev/null
head -c 16 /dev/urandom | od -An -tx1
```

Pisanje u pogrešan uređaj pod /dev može da uništi podatke. Pre rada sa diskovima potrebno je proveriti uređaj i komandu.

## 2.4 /tmp: privremeni fajlovi

/tmp služi za fajlove koji su programima potrebni privremeno. Sadržaj može biti obrisani pri ponovnom pokretanju ili automatskim čišćenjem. Zbog toga /tmp nije prikladno mesto za jedinu kopiju važnih podataka.

Privremeni direktorijum može da se napravi komandom `mktemp -d`:

```
tmpdir=$(mktemp -d)
printf 'Privremeni fajl\n' > "$tmpdir/beleška.txt"
cat "$tmpdir/beleška.txt"
rm -r -- "$tmpdir"
```

Za pojedinačni privremeni fajl koristi se `mktemp`:

```
tmpfile=$(mktemp)
printf 'rezultat\n' > "$tmpfile"
cat "$tmpfile"
rm -- "$tmpfile"
```

Na mnogim sistemima `/tmp` ima dozvole `1777`: svi korisnici mogu da prave fajlove, dok „sticky bit” ograničava brisanje fajlova drugih korisnika. Dozvole se mogu pregledati ovako:

```
ls -ld /tmp
```

## 2.5 Tipovi fajlova

Linux fajlovi mogu da predstavljaju obične podatke, direktorijume, veze, uređaje i druge objekte. Tip se može proveriti komandom `file`:

```
file /etc/passwd /dev/null /proc/cpuinfo
```

Tip	Opis	Oznaka u <code>ls -l</code>
Običan fajl	Tekst, program, slika, arhiva ili drugi podaci.	-
Direktorijum	Sadrži imena drugih fajlova i direktorijuma.	d
Simbolička veza	Pokazuje na drugi fajl ili putanju.	l
Blok-uređaj	Uređaj koji prenosi podatke u blokovima, često disk ili particija.	b
Znakovni uređaj	Uređaj koji prenosi podatke kao tok znakova, često terminal ili drugi uređaj.	c
FIFO, odnosno imenovana cev	Mehanizam za prenos podataka između procesa.	p
Unix domen socket	Lokalna komunikaciona tačka koju mogu da koriste procesi.	s

Primer pregleda tipova i dozvola:

```
ls -l /etc/passwd /dev/null /dev/tty
```

Prvi znak u izlazu komande `ls -l` označava tip fajla:

```
-rw-r--r-- običan fajl
drwxr-xr-x direktorijum
lrwxrwxrwx simbolička veza
crw-rw-rw- znakovni uređaj
```

Ostali znakovi prikazuju dozvole vlasnika, grupe i ostalih korisnika.

## 2.6 Korisne komande za istraživanje

```
ls -lah / # detaljan pregled korena
find / -maxdepth 1 -type d 2>/dev/null # direktorijumi neposredno ispod /
df -h # priključeni sistemi fajlova i zauzeće
mount # priključeni sistemi fajlova
stat /tmp # detalji o putanji
readlink -f /bin # konačna putanja, ako je /bin simbolička veza
```

Pri pregledu celog sistema mogu da se pojave poruke o nedostatku dozvola. **sudo** treba koristiti samo kada su potrebna administratorska prava i kada je namena komande jasna.

## Glava 3

# Dijagnoziranje problema

Ciljevi:

- Posmatrati program kao crnu kutiju (često ne znamo koji se kod izvršava)
- Analizirati ponašanje kako bi se pronašli uzroci problema
- Proširiti znanje o osnovnim alatima za merenje performansi
- Napraviti temelj za metodičnu analizu sistema - ne možemo raditi kvalitetnu analizu ako ne znamo šta i kako možemo analizirati

### 3.1 Metodologija dijagnoziranja problema

Prvi korak u dijagnozi problema je određivanje opsega: da li je problem na nivou celog sistema, jednog procesa, jedne niti, diska, mreže ili spoljnog servisa. Alati iz ovog poglavlja ne traže poznavanje izvornog koda programa. Njihov cilj je da daju merenja i tragove koji usmeravaju dalju analizu.

Tipični koraci su:

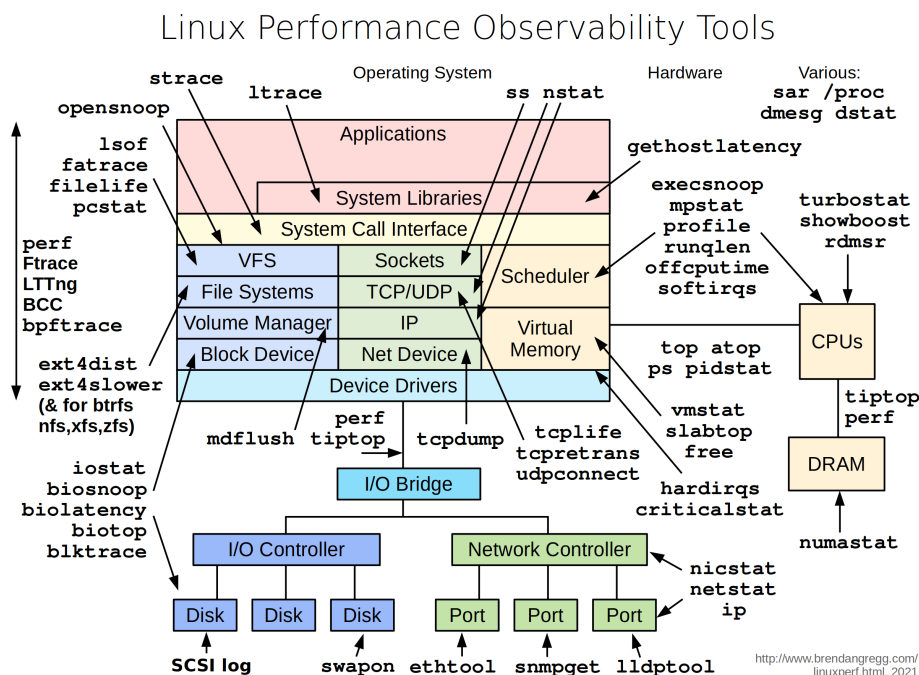
1. **Detekcija simptoma:** šta se vidi spolja, bez čitanja izvornog koda.
2. **Reprodukcija:** tačna komanda, ulaz i vreme pokretanja.
3. **Prvo merenje:** sistemski pregled koji potvrđuje da se simptom zaista dešava.
4. **Sužavanje:** proces, nit, resurs ili sistemski poziv koji objašnjava simptom.
5. **Potvrda:** alat bliži uzroku, na primer debager, profajler, sanitizer, Valgrind, KLEE ili CBMC.
6. **Tumačenje:** šta iz izlaza alata dokazuje uzrok i šta bi bio sledeći inženjerski korak.

Korisno je znati gde Linux čuva sirove podatke o procesima, jezgrima, uređajima i sistemskoj konfiguraciji:

- `/proc` sadrži informacije o procesima, memoriji, CPU jezgrima, montiranim sistemima datoteka i parametrima kernela.
- `/sys` prikazuje model uređaja, drajvere, klase uređaja i deo konfiguracije hardvera.

## 3.2 Alati za analizu performansi

Mnogi alati prikazani u nastavku čitaju podatke iz ovih direktorijuma i formiraju ih u oblik pogodan za brzu analizu.



Širi pregled Linux observability alata može se naći na linku.

### 3.2.1 uptime

`uptime` daje najbrži odgovor na pitanje da li je sistem opterećen. Prikazuje trenutno vreme, koliko dugo sistem radi, broj korisnika i prosečno opterećenje u poslednjih 1, 5 i 15 minuta.

Alat je koristan kada treba brzo proveriti:

- da li je problem kratkotrajan ili traje duže;

- da li se opterećenje povećava ili smanjuje;
- da li ima smisla odmah preći na detaljnije alate kao što su `top`, `vmstat` ili `pidstat`.

```
$ uptime
14:17:27 up 5:43, 1 user, load average: 0.41, 0.93, 1.06
```

Prosek opterećenja dat je u intervalima od približno 1, 5 i 15 minuta. Vrednost predstavlja broj zadataka koji su spremni za izvršavanje ili čekaju neprekidiv I/O događaj. Zato visok load ne znači uvek da je CPU zauzet; uzrok može biti i spor disk ili mrežni sistem datoteka.

### 3.2.2 top / htop

`top` i `htop` prikazuju interaktivni pregled sistema i procesa. Koriste se kada treba brzo videti koji proces troši CPU, memoriju ili veliki broj niti, kao i da li se stanje menja tokom vremena.

Najčešće se posmatraju:

- opšti CPU i memorijski redovi na vrhu prikaza;
- kolone `%CPU`, `%MEM`, `RES`, `S`, `TIME+` i `COMMAND`;
- procesi koji često menjaju stanje ili zauzimaju jedno CPU jezgro;
- niti procesa, ako se prikaz proširi na thread-ove.

Ovi alati su dobri za interaktivnu analizu dugotrajnog problema. Manje su pogodni za procese koji traju kratko, jer uzorkuju stanje u intervalima. Za takve slučajeve korisniji su `pidstat`, `sar` ili alat koji čuva istoriju, kao što je `atop`. `top` i `htop` takođe troše resurse, pa ih treba pažljivo koristiti na već opterećenom sistemu.

### 3.2.3 ps

`ps` prikazuje trenutni snimak procesa. Za razliku od `top`, ne prati promene tokom vremena. Koristan je kada treba sačuvati stanje sistema u izveštaju, proveriti hijerarhiju procesa, argumente kojima je proces pokrenut ili stanje procesa.

`ps -ef f` prikazuje roditelj-dete odnose u obliku stabla. To je korisno kod servera koji pokreću radne procese, shell skripti koje pokreću druge programe i aplikacija koje ostavljaju procese nakon završetka roditelja.

```
$ ps -ef f
[...]
root 1044 1 0 08:34 ? Ssl 0:00 /usr/bin/lightdm
root 1056 1044 7 08:34 tty7 Ssl+ 26:47 _ /usr/lib/Xorg :0 -seat seat0 -auth /ru
root 1088 1044 0 08:34 ? Sl 0:00 _ lightdm --session-child 13 16
ivan 1118 1088 0 08:34 ? Ss 0:08 _ i3
```

```

root 1050 1 0 08:34 ? Ssl 0:00 /usr/lib/accounts-daemon
ivan 1095 1 0 08:34 ? Ss 0:00 /usr/lib/systemd/systemd --user
ivan 1096 1095 0 08:34 ? S 0:00 _ (sd-pam)
[...]
```

### 3.2.4 vmstat

**vmstat** daje sistemski pregled procesa, memorije, swap-a, ulaza/izlaza, prekida, promene konteksta i CPU vremena. Nema detalje po procesu, ali je veoma koristan za prvi pregled jer pokriva više klasa problema u jednom prikazu.

Tipična upotreba je pokretanje sa intervalom od jedne sekunde dok se problem reprodukuje. Prvi red često predstavlja prosek od pokretanja sistema, pa se za aktivnu dijagnozu obično gledaju redovi koji dolaze posle njega.

```

$ vmstat -Sm 1
procs -----memory----- --swap-- -----io----- -system-- -----cpu-----
 r b swpd free buff cache si so bi bo in cs us sy id wa st gu
 2 0 46 782 251 7993 0 0 3406 312 6224 35 12 4 83 1 0 0
 1 0 46 806 251 7986 0 0 0 0 2564 6940 4 3 93 0 0 0
 1 0 46 802 251 7986 0 0 0 0 1713 4499 3 2 95 0 0 0
 0 0 46 801 251 7986 0 0 0 0 1730 4536 2 2 96 0 0 0
~C
```

Važne kolone:

- **r** označava broj procesa spremnih za izvršavanje; trajno visoka vrednost ukazuje na CPU čekanje.
- **b** označava procese blokirane na I/O operacijama.
- **si** i **so** pokazuju čitanje iz swap-a i pisanje u swap; nenulte vrednosti tokom problema često ukazuju na memorijski pritisak.
- **bi** i **bo** pokazuju ulaz i izlaz prema blok uređajima.
- **us** je korisničko CPU vreme, odnosno vreme aplikacije, JVM-a ili runtime-a.
- **sy** je sistemsko CPU vreme, odnosno vreme provedeno u kernelu.
- **wa** je čekanje na I/O; visoka vrednost upućuje na disk, mrežni storage ili blokirajući uređaj.

### 3.2.5 iostat

**iostat** analizira opterećenje blok uređaja. Koristi se kada **vmstat** ili **top** pokažu čekanje na I/O, kada aplikacija sporo čita ili piše podatke, ili kada treba uporediti više diskova i particija.

Opcija `-x` daje proširenu statistiku, `-m` prikazuje protok u MB/s, `-d` ograničava prikaz na uređaje, a `-z` izostavlja uređaje bez aktivnosti.

```
$ iostat -xmdz 1
Linux [...] 09/23/2023 _x86_64_ (8 CPU)

Device r/s rMB/s rrqm/s %rrqm r_await rareq-sz w/s wMB/s ...
dm-0 8.97 0.31 0.00 0.00 0.45 35.22 22.48 0.23 ...
dm-1 0.06 0.00 0.00 0.00 0.26 5.58 0.07 0.00 ...
loop0 0.00 0.00 0.00 0.00 0.00 1.27 0.00 0.00 ...
nvme0n1 6.49 0.31 2.55 28.24 0.27 48.76 12.25 0.23 ...
sda 25.68 3.00 0.46 1.78 2.95 119.49 0.52 0.08 ...

~C
```

Važno je posmatrati i protok i kašnjenje. Velik **rMB/s** ili **wMB/s** znači da uređaj radi mnogo posla, dok veliki **await** znači da operacije čekaju. Ako je **%util** blizu 100% i **await** raste, disk ili storage sloj je verovatan uzrok usporenja.

### 3.2.6 swapon

**swapon** prikazuje aktivne swap prostore. Samo postojanje swap-a nije problem; problem je kada sistem tokom reprodukcije aktivno prebacuje stranice memorije između RAM-a i swap-a. Zato se **swapon** obično koristi zajedno sa kolonama **si** i **so** iz **vmstat**-a ili memorijskim izveštajem iz **sar**-a.

```
$ swapon -s
Filename Type Size Used Priority
/dev/dm-1 partition 17898688 3991268 -2
```

### 3.2.7 mpstat

**mpstat** prikazuje CPU statistiku po jezgru. Koristi se kada prosečna zauzetost CPU-a ne objašnjava problem, na primer kada je samo jedno jezgro stalno zauzeto, a ostatak sistema deluje neiskorišćeno.

Opcija `-P ALL` prikazuje sva jezgra. To pomaže kod aplikacija koje imaju usko grlo u jednoj niti, kod procesa vezanih za određeni CPU, kod softirq opterećenja i kod sistema na kojima raspodela posla nije ravnomerna.

```
$ mpstat -P ALL 1
Linux [...] 09/23/2023 _x86_64_ (8 CPU)

CPU %usr %nice %sys %iowait %irq %soft %steal %guest %gnice %idle
all 4.14 0.00 2.01 0.50 0.38 0.25 0.00 0.00 0.00 92.73
 0 3.03 0.00 1.01 0.00 0.00 1.01 0.00 0.00 0.00 94.95
 1 5.94 0.00 2.97 0.00 0.00 0.00 0.00 0.00 0.00 91.09
 2 3.96 0.00 1.98 0.00 0.99 0.99 0.00 0.00 0.00 92.08
```

3	4.04	0.00	2.02	0.00	1.01	0.00	0.00	0.00	0.00	92.93
4	5.00	0.00	1.00	0.00	1.00	0.00	0.00	0.00	0.00	93.00
5	4.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	95.00
6	3.06	0.00	3.06	4.08	0.00	0.00	0.00	0.00	0.00	89.80
7	4.00	0.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	93.00
~C										

### 3.2.8 lsof

`lsof` prikazuje otvorene datotečne deskriptore. Na Linux sistemu to obuhvata regularne datoteke, direktorijume, biblioteke, pipe-ove, socket-e, terminale i anoniman kernel objekat izložen kao datotečni deskriptor.

Alat je koristan kada treba odgovoriti na pitanja:

- koji proces drži otvorenu datoteku, port ili socket;
- zašto se direktorijum, disk ili mount tačka ne mogu demontirati;
- da li proces curi datotečne deskriptore;
- da li proces i dalje drži obrisanu datoteku, pa prostor na disku nije oslobođen.

```
$ lsof -p <PID>
$ lsof -iTCP -sTCP:LISTEN -P -n
$ lsof +L1
```

Opcije `-P` i `-n` izbegavaju prevođenje portova i adresa u imena. To ubrzava prikaz i uklanjanje zavisnost od DNS-a tokom dijagnoze.

### 3.2.9 pidstat

`pidstat` meri potrošnju resursa po procesu ili niti kroz vreme. Koristi se kada `top` pokaže sumnjiv proces, ali je potrebno sačuvati precizniji vremenski trag ili razdvojiti CPU, memoriju, disk i promene konteksta.

Korisne varijante su:

- `pidstat -p <PID> 1` za praćenje jednog procesa;
- `pidstat -t -p <PID> 1` za praćenje niti unutar procesa;
- `pidstat -d 1` za I/O po procesu;
- `pidstat -r 1` za memoriju po procesu;
- `pidstat -w 1` za dobrovoljne i nedobrovoljne promene konteksta.

```
$ pidstat -t 1
11:04:46 AM UID PID %usr %system %guest %wait %CPU CPU Command
11:04:46 AM 0 1 0.00 0.00 0.00 0.00 0.01 1 systemd
```



```
mmap(0x7f76305ea000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
close(3)
```

```
[...]
```

```
$ strace -e close ls
close(3) = 0
close(3) = 0
close(3) = 0
close(3) = 0
close(3) = 0
close(3) = 0
INDEX lab002.c lab004.c lab006.c lab008.c Makefile
lab001.c lab003.c lab005.c lab007.c lab013.c README.md
close(1) = 0
close(2) = 0
+++ exited with 0 +++
```

Opcija `-e` ograničava praćenje na izabrane sistemske pozive ili grupe poziva. Na primer, `-e trace=file` prati pristup datotekama, `-e trace=network` mrežne pozive, a `-e trace=process` kreiranje i završavanje procesa.

```
$ strace -t -r -p <PID> -o <REPORT_OUTPUT>
-t - timestamp
-r - relative time for each syscall
-p - PID
-o - output path
```

Kada se `strace` priključi na postojeći proces, korisno je zapisati izlaz u datoteku. Opcija `-t` dodaje vreme događaja, a `-r` vreme proteklo od prethodnog sistemskog poziva. To pomaže da se pronađu dugi zastoji.

```
$ strace -c ls
INDEX lab002.c lab004.c lab006.c lab008.c Makefile
lab001.c lab003.c lab005.c lab007.c lab013.c README.md

```

0.00	0.000000	0	2	read
0.00	0.000000	0	2	write
0.00	0.000000	0	7	close
0.00	0.000000	0	13	mmap
0.00	0.000000	0	4	mprotect
0.00	0.000000	0	1	munmap
0.00	0.000000	0	3	brk
0.00	0.000000	0	2	ioctl
0.00	0.000000	0	2	pread64
0.00	0.000000	0	1	1 access
0.00	0.000000	0	1	execve
0.00	0.000000	0	6	4 prctl

0.00	0.000000	0	2	1 arch_prctl
0.00	0.000000	0	2	getdents64
0.00	0.000000	0	1	set_tid_address
0.00	0.000000	0	5	openat
0.00	0.000000	0	6	newfstatat
0.00	0.000000	0	1	set_robust_list
0.00	0.000000	0	1	prlimit64
0.00	0.000000	0	1	getrandom
0.00	0.000000	0	1	rseq
100.00	0.000000	0	64	6 total

Opcija `-c` daje agregirani izveštaj. Ona je korisna kada ne treba čitati svaki poziv, već samo proceniti gde proces provodi vreme i koji pozivi se ponavljaju. Za programe koji pokreću child procese često se dodaje i `-f`, kako bi se pratio ceo procesni skup.

### 3.2.11 Analiza mreže

Mrežni problem se često prvo vidi kao spor odgovor aplikacije, odbijena konekcija, zauzet port ili timeout. Pre detaljne analize protokola korisno je proveriti da li proces sluša na očekivanom portu, da li postoji konekcija ka udaljenom sistemu i da li paketi zaista prolaze kroz interfejs.

Za duboku analizu protokola koristi se alat kao što je **Wireshark**. U komandnoj liniji se najčešće kombinuju `ss`, `lsof`, `strace` i `tcpdump`.

#### 3.2.11.1 tcpdump

`tcpdump` hvata pakete na mrežnom interfejsu. Koristi se kada treba dokazati da paket napušta mašinu, da odgovor stiže, da se koristi pogrešan port ili da mrežni tok izgleda drugačije od očekivanog.

Alat često zahteva administratorske privilegije. Na zauzetim sistemima treba koristiti filtere, jer sirov mrežni saobraćaj brzo postaje prevelik za čitanje.

```
$ sudo tcpdump dst port 123
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
21:02:19.112502 IP test33.ntp > 199.30.140.74.ntp: NTPv4, Client, length 48
21:02:19.113888 IP 216.239.35.0.ntp > test33.ntp: NTPv4, Server, length 48
21:02:20.150347 IP test33.ntp > 216.239.35.0.ntp: NTPv4, Client, length 48
21:02:20.150991 IP 216.239.35.0.ntp > test33.ntp: NTPv4, Server, length 48

$ sudo tcpdump -nn -i any host <HOST> and port <PORT>
$ sudo tcpdump -nn -i any -w trace.pcap
```

Opcija `-nn` isključuje razrešavanje imena i portova. Opcija `-i any` hvata saobraćaj sa svih interfejsa, što je korisno kada nije jasno kojim putem paket ide. Zapis u `pcap` datoteku se kasnije može otvoriti u Wireshark-u.

### 3.2.11.2 netstat / ss

`netstat` i `ss` prikazuju socket-e, konekcije, portove i procese koji ih koriste. `netstat` je stariji alat, dok je `ss` deo modernog `iproute2` skupa i obično daje brži i potpuniji prikaz.

Ovi alati se koriste kada treba proveriti:

- da li servis sluša na očekivanom portu;
- koji proces je zauzeo port;
- da li su konekcije u stanjima LISTEN, ESTABLISHED, TIME-WAIT ili CLOSE-WAIT;
- da li redovi Recv-Q i Send-Q rastu, što može ukazati na zagušenje ili aplikaciju koja ne čita podatke.

```
$ netstat -tulnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name
tcp 0 0 127.0.0.1:57343 0.0.0.0:* LISTEN 3501/sshd
tcp 0 0 0.0.0.0:27036 0.0.0.0:* LISTEN 3501/sshd
tcp 0 0 127.0.0.1:33095 0.0.0.0:* LISTEN 3501/sshd
tcp 0 0 127.0.0.1:35831 0.0.0.0:* LISTEN 3501/sshd
tcp 0 0 127.0.0.1:43221 0.0.0.0:* LISTEN -
tcp 0 0 127.0.0.1:27060 0.0.0.0:* LISTEN 3501/sshd
tcp6 0 0 :::1:631 :::* LISTEN -
tcp6 0 0 :::6600 :::* LISTEN 1135/mysqld
udp 0 0 0.0.0.0:27036 0.0.0.0:* - 3501/sshd
udp 0 0 0.0.0.0:5353 0.0.0.0:* - -
udp 0 0 0.0.0.0:46653 0.0.0.0:* - -
udp 0 0 0.0.0.0:38865 0.0.0.0:* - -
udp6 0 0 :::44350 :::* - -
udp6 0 0 :::5353 :::* - -
udp6 0 0 :::46653 :::* - -
```

`ss` je preporučeni izbor za nove analize:

```
$ ss -tunpl
Netid State Recv-Q Send-Q Local Address:Port Peer Address:Port Process
udp UNCONN 0 0 0.0.0.0:27036 0.0.0.0:* users:(("steam",p
udp UNCONN 0 0 0.0.0.0:5353 0.0.0.0:* -
udp UNCONN 0 0 0.0.0.0:46653 0.0.0.0:* -
udp UNCONN 0 0 0.0.0.0:38865 0.0.0.0:* -
```

```

udp UNCONN 0 0 [::]:44350 [::]:*
udp UNCONN 0 0 [::]:5353 [::]:*
udp UNCONN 0 0 [::]:46653 [::]:*
tcp LISTEN 0 128 127.0.0.1:57343 0.0.0.0:* users:(("steam",pid=3501,fd=
tcp LISTEN 0 128 0.0.0.0:27036 0.0.0.0:* users:(("steam",pid=3501,fd=
tcp LISTEN 0 4096 127.0.0.1:631 0.0.0.0:*
tcp LISTEN 0 128 127.0.0.1:33095 0.0.0.0:* users:(("steam",pid=3501,fd=
tcp LISTEN 0 128 127.0.0.1:35831 0.0.0.0:* users:(("steam",pid=3501,fd=
tcp LISTEN 0 4096 127.0.0.1:43221 0.0.0.0:*
tcp LISTEN 0 128 127.0.0.1:27060 0.0.0.0:* users:(("steam",pid=3501,fd=
tcp LISTEN 0 4096 [::1]:631 [::]:*
tcp LISTEN 0 5 *:6600 *: users:(("mpd",pid=1135,fd=4)

```

Opcije u primeru znače: `-t` za TCP, `-u` za UDP, `-n` za numerički prikaz adresa i portova, `-p` za proces, a `-l` za socket-e koji slušaju.

### 3.2.12 System Activity Reporter (sar)

`sar` prikuplja i prikazuje sistemske metrike kroz vreme. Pripada paketu `sysstat`. Koristan je kada problem nije moguće uhvatiti interaktivno, jer se na mnogim sistemima podaci periodično čuvaju i mogu se analizirati naknadno.

`sar` može da prati CPU, memoriju, swap, mrežne uređaje, diskove i druge sistemske metrike. U nastavku su tri česta preseka: CPU, memorija i mreža.

```

$ sar -u 1 3
11:12:22 AM CPU %user %nice %system %iowait %steal %idle
11:12:23 AM all 3.88 0.00 1.88 0.25 0.00 93.99
11:12:24 AM all 4.45 0.00 1.11 0.12 0.00 94.31
11:12:25 AM all 3.98 0.00 1.99 0.25 0.00 93.78
Average: all 4.10 0.00 1.66 0.21 0.00 94.03

$ sar -r 1 3
11:13:26 AM kbmfree kbavail kbmused %memused kbbuffers kbcached kbcommit %commit kbswpd
11:13:27 AM 5841564 7857352 7545328 46.37 31224 2286720 21332312 62.43 672
11:13:28 AM 5841060 7856848 7545928 46.38 31224 2286720 21328940 62.42 672
11:13:29 AM 5837028 7852948 7549820 46.40 31232 2286852 21332780 62.43 672
Average: 5839884 7855716 7547025 46.38 31227 2286764 21331344 62.43 672

$ sar -n DEV 1 1
11:14:55 AM IFACE rxpck/s txpck/s rxkB/s txkB/s rxcmp/s txcmp/s rxcst/s %if
11:14:56 AM lo 9.00 9.00 0.74 0.74 0.00 0.00 0.00
11:14:56 AM enp3s0 0.00 0.00 0.00 0.00 0.00 0.00 0.00
11:14:56 AM wlp2s0 1.00 2.00 0.38 1.20 0.00 0.00 0.00
11:14:56 AM docker0 0.00 0.00 0.00 0.00 0.00 0.00 0.00
11:14:56 AM nordlynx 1.00 2.00 0.34 1.08 0.00 0.00 0.00

```

Average:	IFACE	rxpck/s	txpck/s	rxkB/s	txkB/s	rxcmp/s	txcmp/s	rxm
Average:	lo	9.00	9.00	0.74	0.74	0.00	0.00	
Average:	enp3s0	0.00	0.00	0.00	0.00	0.00	0.00	
Average:	wlp2s0	1.00	2.00	0.38	1.20	0.00	0.00	
Average:	docker0	0.00	0.00	0.00	0.00	0.00	0.00	
Average:	nordlynx	1.00	2.00	0.34	1.08	0.00	0.00	

Kada je prikupljanje istorije uključeno, **sar** se koristi za analizu problema koji se desio ranije. Na primer, može se proveriti da li je u vreme incidenta porastao **%iowait**, potrošnja memorije, swap aktivnost ili mrežni protok. Time se dijagnoza ne oslanja samo na trenutno stanje sistema.

### 3.3 Kako birati alate tokom analize

U opštem slučaju, dovoljno je pratiti naredne korake:

1. zabeležiti vreme, komandu, ulaz i simptome problema;
2. proveriti opšte stanje sistema pomoću **uptime**, **top**, **vmstat** ili **sar**;
3. suziti problem na proces ili nit pomoću **ps**, **pidstat**, **lsuf** i **ss**;
4. proveriti konkretan resurs: CPU pomoću **mpstat**, disk pomoću **iostat**, mrežu pomoću **ss** i **tcpdump**;
5. ako je potrebno, analizirati sistemске pozive pomoću **strace**.

U praksi je najvažnije brzo preći sa opšteg simptoma na alat koji meri bliži nivo sistema. Sledeća tabela daje početni izbor alata. Nije zamena za detaljnu analizu, već mapa za prvi sledeći korak.

Simptom	Prvi alati	Sledeći korak
Visok CPU na sistemu	<b>uptime</b> , <b>top</b> , <b>mpstat</b>	Ako je kriv jedan proces, preći na <b>pidstat</b> , debager ili uzorkujući profajler.
Proces troši CPU	<b>top -H</b> , <b>pidstat</b> , <b>strace -c</b>	Ako nema mnogo sistemskih poziva, uzorkovati stek pomoću <b>perf-a</b> ili otvoriti proces u debageru.
Proces spava ili deluje zaglavljeno	<b>ps</b> , <b>pidstat -w</b> , <b>strace -tt -T</b>	Tražiti sistemski poziv u kome proces čeka: <b>accept</b> , <b>read</b> , <b>futex</b> , disk I/O ili mrežu.
Disk je spor ili zauzet	<b>iostat -x</b> , <b>pidstat -d</b> , <b>lsuf</b>	Povezati uređaj, proces i datoteku, pa proveriti obrazac čitanja ili upisa.
Mrežna konekcija ne radi	<b>ss</b> , <b>lsuf -i</b> , <b>tcpdump</b>	Prvo potvrditi da proces sluša, zatim da paket stiže do mašine i da odgovor izlazi.

Simptom	Prvi alati	Sledeći korak
Proces koristi previše memorije	<b>top, ps, pmap</b>	Za C/C++ programe preći na Massif, DHAT, Memcheck ili sanitizer; za JVM/.NET koristiti odgovarajući runtime alat.

### 3.4 Laboratorijski primeri

Primeri su preuzeti iz perf-labs repozitorijuma Brendana Gregg-a, korišćenog tokom prezentacije o alatima za merenje performansi Linux kernel-a. Svaki program namerno pravi jedan simptom. Cilj vežbe nije odmah otvoriti izvorni kod, već prvo posmatrati proces kao crnu kutiju i zatim potvrditi uzrok.

Pre pokretanja primera treba ih prevesti:

```
$ make all
```

Pokretati jedan primer po jedan, u posebnom terminalu. Posle analize zaustaviti proces pomoću **Ctrl+C** ili komandom:

```
$ pkill lab001
```

Tok dijagnoze za svaki primer je isti:

1. proveriti opšte stanje sistema;
2. pronaći proces koji pravi simptom;
3. odrediti resurs koji je usko grlo;
4. potvrditi uzrok alatom koji posmatra bliži nivo sistema.

#### 3.4.1 Pregled primera

Primer	Simptom	Prvi alati
lab001	CPU petlja u jednoj funkciji	<b>top, pidstat, strace</b>
lab002	proces čeka konekciju	<b>ss, strace, tcpdump</b>
lab003	veliki broj sistemskih poziva	<b>vmstat, pidstat, strace</b>
lab004	kratkotrajni child procesi	<b>pidstat, ps, strace -f</b>
lab005	sinhroni upisi na disk	<b>vmstat, iostat, pidstat -d</b>
lab006	CPU rad raspodeljen kroz više funkcija	<b>top, mpstat, pidstat</b>
lab007	proces uglavnom spava	<b>top, pidstat -w, strace</b>
lab008	upisi na disk sa povremenim kašnjenjem	<b>iostat, pidstat -d, strace -T</b>
lab013	iscrpeljivanje steka rekurzijom	<b>ulimit, dmesg, sar -r</b>

### 3.4.2 lab001: CPU petlja

Ovaj program zauzima CPU bez sistemskih poziva i bez I/O aktivnosti. Kod je mali, ali je koristan za vežbanje razlike između procesa koji radi u korisničkom prostoru i procesa koji čeka kernel ili uređaj.

```
/*
 * lab001 - CPU hot stack.
 *
 * 21-May-2015 Brendan Gregg Created this.
 */

void func_c() {
 for (;;) {
 }
}

void func_b() { func_c(); }

void func_a() { func_b(); }

int main(int argc, char *argv[]) {
 func_a();
 return (0);
}
```

Pokrenuti program i posmatrati ga iz drugog terminala:

```
$./lab001 &
$ top -p $(pgrep lab001)
$ pidstat -p $(pgrep lab001) 1
$ strace -p $(pgrep lab001)
```

Očekivano je da `top` i `pidstat` pokažu visoku CPU potrošnju. `strace` neće prikazivati stalne sistemske pozive, jer se program vrti u funkciji `func_c`. Time se problem svodi na CPU-bound kod u korisničkom prostoru. Za tačnu lokaciju vrućeg steka koriste se debager ili profajler.

### 3.4.3 lab002: čekanje na konekciju

Ovaj program otvara TCP port i zatim čeka klijentsku konekciju. Simptom je proces koji postoji, ali troši malo CPU vremena i deluje neaktivan.

```
/*
 * lab002 - waiting on accept().
 *
 * 21-May-2015 Brendan Gregg Created this.
 */
```

```
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define BUFSIZE 256
#define PORT 12340

void connection_handler(int connection_fd) {
 int nbytes;
 char buffer[BUFSIZE];

 nbytes = read(connection_fd, buffer, BUFSIZE);
 buffer[nbytes] = 0;

 nbytes = sprintf(buffer, "Shhh... I'm sleeping.\n");
 write(connection_fd, buffer, nbytes);
}

void wait_for_work(int socket_fd) {
 int connection_fd;
 struct sockaddr_in client;
 socklen_t length;

 length = sizeof(client);

 while ((connection_fd =
 accept(socket_fd, (struct sockaddr *)&client, &length)) > -1) {
 connection_handler(connection_fd);
 close(connection_fd);
 }
}

int main(int argc, char *argv[]) {
 struct sockaddr_in address;
 int socket_fd, connection_fd;
 size_t address_length;
 pid_t child;
 struct hostent *server;

 if ((socket_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
 printf("ERROR: socket() failed\n");
 }
```

```

 exit(1);
 }

 bzero((char *)&address, sizeof(address));
 address.sin_family = AF_INET;
 address.sin_port = htons(PORT);
 address.sin_addr.s_addr = INADDR_ANY;

 if (bind(socket_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
 printf("ERROR: bind() failed");
 exit(2);
 }

 if (listen(socket_fd, 5) != 0) {
 printf("ERROR: listen() failed\n");
 exit(3);
 }

 wait_for_work(socket_fd);

 close(socket_fd);
 return (0);
}

```

Prvo proveriti da li proces sluša na očekivanom portu:

```

$./lab002 &
$ ss -ltnp 'sport = :12340'
$ lsof -iTCP:12340 -sTCP:LISTEN -P -n

```

Zatim proveriti na kom sistemskom pozivu proces čeka:

```

$ strace -tt -T -p $(pgrep lab002)

```

Očekivani trag pokazuje da proces čeka u `accept`. Kada se klijent poveže, `accept` se završava, proces čita podatke i šalje odgovor.

```

$ nc 127.0.0.1 12340

```

Ako klijent ne može da se poveže, proveriti port pomoću `ss`, lokalni firewall i paketni trag pomoću `tcpdump`. Ako port nije otvoren, problem je pre mrežnog toka: proces nije pokrenut, nije uspeo `bind`, ili sluša na drugoj adresi.

#### 3.4.4 lab003: čitanje nula bajtova

Ovaj program pravi datoteku, otvara je za čitanje i zatim u beskonačnoj petlji poziva `read` sa veličinom 0. Poziv uspeva, ali ne prenosi podatke. Simptom je neobično velika aktivnost sistemskih poziva bez stvarnog disk I/O rada.

```
/*
 * lab003 - 0 byte successful reads.
 *
 * 21-May-2015 Brendan Gregg Created this.
 */

#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

const char *datafile = "lab003.data";

#define BUFSIZE (8 * 1024)
#define FILESIZE (10 * 1024 * 1024)

void init_data() {
 int fd;
 long long i;
 char buf[BUFSIZE];

 if ((fd = open(datafile, O_CREAT | O_WRONLY, 0644)) < 0) {
 printf("ERROR: Can't open datafile: %s\n", datafile);
 exit(1);
 }

 for (i = 0; i < FILESIZE / BUFSIZE; i++) {
 if (write(fd, buf, BUFSIZE) < 0) {
 printf("ERROR: write error.\n");
 exit(2);
 }
 }

 close(fd);
}

void load_data() {
 int fd;
 char buf[1];

 if ((fd = open(datafile, O_RDONLY)) < 0) {
 printf("ERROR: Can't open database: %s\n", datafile);
 exit(2);
 }
}
```

```

 }

 for (;;) {
 read(fd, &buf, 0);
 }

 (void)close(fd);
}

void import_database() { load_data(); }

int main() {
 init_data();

 for (;;) {
 import_database();
 }

 return (0);
}

```

Krenuti od opšteg pregleda:

```

$./lab003 &
$ vmstat 1
$ pidstat -p $(pgrep lab003) 1
$ iostat -x 1
$ sar -n DEV 1

```

Ako CPU raste, a disk i mreža ne pokazuju odgovarajući promet, suziti analizu na sistemske pozive:

```

$ strace -tp $(pgrep lab003) 2>&1 | head -n 20
$ strace -c -p $(pgrep lab003)

```

Očekivani nalaz je ponavljanje poziva nalik na `read(fd, ..., 0) = 0`. To objašnjava zašto proces troši CPU i sistemsko vreme, a ne proizvodi korisno čitanje podataka.

### 3.4.5 lab004: kratkotrajni child procesi

Ovaj program u petlji pokreće komandu `cksum` nad datotekom. Problem može lako promaći u `top`-u, jer child procesi kratko traju.

```

/*
 * lab004 - short lived cksum processes.
 *
 * 21-May-2015 Brendan Gregg Created this.

```

```
*/

#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

const char *datafile = "lab004.data";

#define BUFSIZE (8 * 1024)
#define FILESIZE (10 * 1024 * 1024)

void init_data() {
 int fd;
 long long i;
 char buf[BUFSIZE];

 if ((fd = open(datafile, O_CREAT | O_WRONLY, 0644)) < 0) {
 printf("ERROR: Can't open datafile: %s\n", datafile);
 exit(1);
 }

 for (i = 0; i < FILESIZE / BUFSIZE; i++) {
 if (write(fd, buf, BUFSIZE) < 0) {
 printf("ERROR: write error.\n");
 exit(2);
 }
 }

 close(fd);
}

void check_log() {
 char cmd[256];
 snprintf(cmd, sizeof(cmd), "cksum %s >/dev/null 2>&1", datafile);

 system(cmd);
}

int main() {
 init_data();

 for (;;) {
```

```

 check_log();
 }

 return (0);
}

```

Početi od procesa roditelja, a zatim pratiti child procese:

```

$./lab004 &
$ pidstat -p $(pgrep lab004) 1
$ ps -ef f | grep -E 'lab004|cksum'
$ strace -f -e trace=process -p $(pgrep lab004)

```

Opcija `-f` je bitna jer prati child procese. Očekivani trag pokazuje ponavljanje kreiranja procesa, izvršavanje `cksum` i čekanje da child proces završi. Ako se gleda samo roditeljski proces, CPU ili I/O slika može izgledati nepotpuno.

Za proveru disk aktivnosti koristiti:

```

$ pidstat -d 1
$ iostat -x 1

```

### 3.4.6 lab005: sinhroni upisi na disk

Ovaj program otvara datoteku sa `O_SYNC` i stalno piše u nju. Svaki upis čeka da podaci budu sinhronizovani sa storage slojem. Simptom može biti visok `iowait`, zauzet disk i spor odziv drugih procesa.

```

/*
 * lab005 - sync disk writes to a file.
 *
 * 21-May-2015 Brendan Gregg Created this.
 */

#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

const char *datafile = "lab005.data";

#define BUFSIZE (8 * 1024)
#define FILESIZE (10 * 1024 * 1024)

void write_log(int fd) {
 char buf[BUFSIZE];

```

```

 int i;

 for (;;) {
 for (i = 0; i < FILESIZE / BUFSIZE; i++) {
 if (write(fd, buf, BUFSIZE) < 0) {
 printf("ERROR: write error.\n");
 exit(2);
 }
 }

 if (lseek(fd, 0, SEEK_SET) < 0) {
 printf("ERROR: seek() failed.\n");
 exit(3);
 }
 }
}

int main() {
 int fd;

 if ((fd = open(datafile, O_CREAT | O_WRONLY | O_SYNC, 0644)) < 0) {
 printf("ERROR: writing to %s\n", datafile);
 exit(1);
 }

 write_log(fd);

 return (0);
}

```

Za ovaj primer je zgodna USE metodologija: proveriti iskorišćenost, zasićenje i greške za CPU, memoriju, disk i mrežu.

```

$./lab005 &
$ htop
$ mpstat -P ALL 1
$ vmstat 1
$ iostat -x 1
$ sar -n DEV 1

```

Ako CPU nije potpuno zauzet, nema swap aktivnosti, a `iostat` pokazuje veliki `%util` ili rast `await`, uzrok je najverovatnije disk. Zatim proveriti procesni I/O i otvorenu datoteku:

```

$ pidstat -d -p $(pgrep lab005) 1
$ lsof -p $(pgrep lab005) | grep lab005.data

```

Bitna veza sa kodom je opcija `O_SYNC`. Ona menja cenu svakog `write` poziva, pa problem nije samo u količini podataka već i u načinu pisanja.

### 3.4.7 lab006: više CPU vrućih funkcija

Ovaj program troši CPU u dve funkcije. Za razliku od `lab001`, vruće vreme nije vezano za jednu jedinu petlju. Primer pokazuje granicu sistemske dijagnoze: alati kao `top` i `pidstat` mogu da potvrde CPU problem, ali ne mogu sami da raspodele vreme po funkcijama.

```
/*
 * lab006 - CPU hot stacks.
 *
 * 21-May-2015 Brendan Gregg Created this.
 */

#define SPINS (1000 * 1000) /* tune as desired */

void func_c() {
 int i;
 for (i = 0; i < SPINS; i++) {
 }
}

void func_b() { func_c(); }

void func_e() {
 int i;
 for (i = 0; i < 5 * SPINS; i++) {
 }
}

void func_a() {
 for (;;) {
 func_b();
 func_e();
 }
}

int main(int argc, char *argv[]) {
 func_a();
 return (0);
}
```

Prvo potvrditi da je problem CPU-bound:

```
$./lab006 &
$ top -p $(pgrep lab006)
$ mpstat -P ALL 1
$ pidstat -p $(pgrep lab006) 1
$ strace -p $(pgrep lab006)
```

Ako `strace` ne prikazuje stalne sistemske pozive, a CPU potrošnja je visoka, proces radi u korisničkom prostoru. Sledeći korak je uzorkovanje steka pomoću profajlera ili analiza procesa u debageru. Ovaj primer zato povezuje poglavlje o dijagnozi sa poglavljima o debugovanju i profajliranju.

### 3.4.8 lab007: proces uglavnom spava

Ovaj program periodično spava, pa zatim kratko radi u petlji. Simptom je proces koji je prisutan, ali troši malo CPU vremena. Takav proces može izgledati nebitno u `top`-u, iako je važan za razumevanje kašnjenja.

```
/*
 * lab007 - asleep in function call.
 *
 * 21-May-2015 Brendan Gregg Created this.
 */

#include <unistd.h>

#define SPINS (10 * 1000 * 1000) /* tune to ~1% CPU */

void func_l() {
 int i, j;

 for (;;) {
 usleep(1000 * 1000);
 for (i = 0, j = 0; i < SPINS; i++) {
 j++;
 }
 }
}

void func_k() { func_l(); }

void func_j() { func_k(); }

int main(int argc, char *argv[]) {
 func_j();
 return (0);
}
```

Posmatrati CPU vreme i promene konteksta:

```
$./lab007 &
$ top -p $(pgrep lab007)
$ pidstat -w -p $(pgrep lab007) 1
$ strace -tt -T -p $(pgrep lab007)
```

Očekivani trag pokazuje periodične pozive za spavanje, najčešće kroz `nanosleep` ili srodan sistemski poziv. `pidstat -w` pokazuje promene konteksta, jer proces dobrovoljno prepušta CPU dok spava.

### 3.4.9 lab008: disk latencije sa izuzecima

Ovaj program takođe koristi `O_SYNC`, ali povremeno piše veći blok podataka. Simptom nije samo zauzet disk, već i povremeni skokovi latencije pojedinačnih `write` poziva.

```
/*
 * lab008 - sync disk writes to a file, with (syscall) latency outliers.
 *
 * 21-May-2015 Brendan Gregg Created this.
 */

#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

const char *datafile = "lab008.data";

#define BUFSIZE (8 * 1024)
#define BIGSIZE (10 * 1024 * 1024)
#define FILESIZE (10 * 1024 * 1024)

void write_log(int fd) {
 char *buf, *big;
 long long i;
 int ret, j;

 buf = malloc(BUFSIZE);
 big = malloc(BIGSIZE);
 if (buf == NULL || big == NULL) {
 printf("ERROR: malloc buffers.\n");
 exit(1);
 }
}
```

```

 }
 bzero(buf, BUFSIZE);
 bzero(big, BIGSIZE);

 for (;;) {
 for (i = 0, j = 0; i < FILESIZE;) {
 if ((j++ % 100) == 0) {
 ret = write(fd, big, BIGSIZE);
 i += BIGSIZE;
 } else {
 ret = write(fd, buf, BUFSIZE);
 i += BUFSIZE;
 }

 if (ret < 0) {
 printf("ERROR: write error.\n");
 exit(2);
 }
 }

 if (lseek(fd, 0, SEEK_SET) < 0) {
 printf("ERROR: seek() failed.\n");
 exit(3);
 }

 free(buf);
 free(big);
 }

 int main() {
 int fd;

 if ((fd = open(datafile, O_CREAT | O_WRONLY | O_SYNC, 0644)) < 0) {
 printf("ERROR: writing to %s\n", datafile);
 exit(1);
 }

 write_log(fd);

 return (0);
 }

```

Prvo potvrditi da je usko grlo disk:

```
$./lab008 &
$ iostat -x 1
$ pidstat -d -p $(pgrep lab008) 1
```

Zatim izmeriti trajanje sistemskih poziva:

```
$ strace -tt -T -e trace=write,lseek -p $(pgrep lab008)
```

Opcija `-T` dodaje trajanje svakog sistemskog poziva. Očekivano je da se vide obični upisi i povremeni znatno skuplji upisi. U kodu se to poklapa sa granom koja periodično šalje veći bafer.

### 3.4.10 lab013: rekurzija i stek

Ovaj program beskonačno rekurzivno poziva istu funkciju. Simptom može izgledati kao memorijski problem, ali uzrok nije heap alokacija već rast steka do ograničenja procesa.

```
/*
 * lab013 - out of memory by recursion. tricky.
 *
 * 21-May-2015 Brendan Gregg Created this.
 */

void func_r() {
 int i;
 for (i = 0; i < 100; i++) {
 }
 func_r();
}

int main(int argc, char *argv[]) {
 func_r();
 return (0);
}
```

Pošto proces može brzo da se završi, prvo proveriti ograničenje steka i poruke kernela posle pada:

```
$ ulimit -s
$./lab013
$ dmesg -T | tail -n 20
```

Ako sistem generiše core dump, sledeći korak je analiza core datoteke u debuggeru. Ako se želi posmatranje dok proces radi, može se privremeno povećati ograničenje steka u shell-u iz kog se pokreće primer:

```
$ ulimit -s 65536
$./lab013 &
$ pidstat -r -p $(pgrep lab013) 1
```

Ovaj primer je dobar podsetnik da globalna memorijska slika ne mora odmah da pokaže uzrok. Lokalna ograničenja procesa, kao što je veličina steka, mogu biti dovoljna da program padne.



## Glava 4

# Efikasno korišćenje terminala

### 4.1 Osnovne komande

Komanda	Namena	Primer
<code>pwd</code>	Prikazuje trenutni direktorijum	<code>pwd</code>
<code>ls</code>	Prikazuje sadržaj direktorijuma	<code>ls -lah</code>
<code>cd</code>	Menja direktorijum	<code>cd ~/Projekti</code>
<code>mkdir</code>	Pravi direktorijum	<code>mkdir -p projekat/src</code>
<code>touch</code>	Pravi prazan fajl ili menja vreme izmene	<code>touch beleške.txt</code>
<code>cp</code>	Kopira fajl ili direktorijum	<code>cp izvor.txt kopija.txt</code>
<code>mv</code>	Premešta ili preimenuje	<code>mv staro.txt novo.txt</code>
<code>rm</code>	Briše fajlove	<code>rm fajl.txt</code>
<code>rmdir</code>	Briše prazan direktorijum	<code>rmdir prazan_direktorijum</code>
<code>file</code>	Prikazuje tip fajla	<code>file slika.png</code>
<code>stat</code>	Prikazuje detalje o fajlu	<code>stat fajl.txt</code>
<code>du</code>	Prikazuje zauzeće prostora	<code>du -sh *</code>
<code>df</code>	Prikazuje slobodan prostor na diskovima	<code>df -h</code>

Komanda	Namena	Primer
<b>tree</b>	Prikazuje stablo direktorijuma; možda treba instalirati	<b>tree -L 2</b>

Pažnja: **rm** trajno briše fajlove u uobičajenoj konfiguraciji. Proveri putanju pre upotrebe, naročito uz opcije **-r** i **-f**.

## 4.2 Pregledanje i obrada teksta

Komanda	Namena	Primer
<b>cat</b>	Ispisuje ceo fajl	<b>cat config.txt</b>
<b>less</b>	Pregledanje fajla stranicu po stranicu	<b>less veliki.log</b>
<b>head</b>	Prikazuje početak fajla	<b>head -n 20 podaci.csv</b>
<b>tail</b>	Prikazuje kraj fajla	<b>tail -n 20 aplikacija.log</b>
<b>tail -f</b>	Prati nove redove u fajlu uživo	<b>tail -f aplikacija.log</b>
<b>wc</b>	Broji redove, reči ili bajtove	<b>wc -l *.log</b>
<b>sort</b>	Sortira redove	<b>sort imena.txt</b>
<b>uniq</b>	Sažima susedne ponovljene redove	<b>sort imena.txt   uniq -c</b>
<b>cut</b>	Izdvađa kolone ili delove redova	<b>cut -d, -f1 podaci.csv</b>
<b>tr</b>	Menja ili uklanja znakove	<b>tr '[:lower:]' '[:upper:]' &lt; ulaz.txt</b>
<b>sed</b>	Pronalazi i menja tekst	<b>sed 's/staro/novo/g' fajl.txt</b>
<b>awk</b>	Obradjuje tekst po kolonama i pravilima	<b>awk '{print \$1}' access.log</b>
<b>grep</b>	Traži tekst ili obrazac	<b>grep -n 'ERROR' aplikacija.log</b>
<b>tee</b>	Prikazuje izlaz i upisuje ga u fajl	<b>komanda   tee izlaz.txt</b>

```
grep 'tekst' fajl.txt
grep -in 'greška' *.log
```

# bez razlikovanja velikih i malih slova, sa brojem reda

```
grep -RIn 'TODO' src/ # rekurzivno, sa brojem reda, ignoriše binarne fajlove
grep -E 'warn|error' app.log # prošireni regularni izrazi
```

## 4.3 Pronalaženje fajlova: find

`find` traži fajlove i direktorijume po putanji, imenu, tipu, veličini i drugim svojstvima.

```
find . -name '*.png' # PNG fajlovi ispod trenutnog direktorijuma
find . -type d -name 'build' # direktorijumi imena build
find . -type f -iname '*.jpg' # fajlovi, bez razlikovanja veličine slova
find . -type f -mtime -7 # fajlovi menjani u poslednjih 7 dana
find . -type f -size +100M # fajlovi veći od 100 MB
```

Pokretanje komande nad pronađenim fajlovima:

```
find . -type f -name '*.tmp' -delete
find . -type f -name '*.png' -exec file {} \;
find . -type f -name '*.png' -exec cp -t /tmp/slike {} +
```

`-delete` briše pronađene fajlove. Pre brisanja prvo proveri šta će biti obuhvaćeno:

```
find . -type f -name '*.tmp' -print
```

Za imena koja sadrže razmake ili neobične znakove, bezbednije je koristiti `-print0` i `xargs -0`:

```
find . -type f -name '*.png' -print0 | xargs -0 -r file
```

## 4.4 Procesi i periodično izvršavanje

Komanda	Namena	Primer
<code>ps</code>	Prikazuje procese	<code>ps aux</code>
<code>top</code>	Prikazuje procese i opterećenje uživo	<code>top</code>
<code>pgrep</code>	Traži procese po imenu	<code>pgrep -a firefox</code>
<code>kill</code>	Šalje signal procesu	<code>kill 1234</code>
<code>jobs</code>	Prikazuje poslove pokrenute iz trenutnog terminala	<code>jobs</code>
<code>bg</code>	Nastavlja zaustavljen posao u pozadini	<code>bg %1</code>
<code>fg</code>	Vraća posao u prednji plan	<code>fg %1</code>

Komanda	Namena	Primer
<code>watch</code>	Periodično ponavlja komandu i prikazuje rezultat	<code>watch -n 2 'date; df -h'</code>
<code>sleep</code>	Čeka zadati broj sekundi	<code>sleep 5</code>
<code>timeout</code>	Prekida komandu posle zadanog vremena	<code>timeout 10s neka_komanda</code>

`watch` ponavlja komandu podrazumevano na svakih nekoliko sekundi; `-n` menja razmak. Za izlaz pritisni `Ctrl+C`.

```
watch -n 1 'grep MemAvailable /proc/meminfo'
watch -d -n 2 'ls -l output/' # ističe razlike između prikaza
```

## 4.5 Pozadinski procesi

```
du -sh ~/Downloads &
jobs
```

`&` pokreće komandu u pozadini. U terminalu:

- `Ctrl+C` prekida komandu u prednjem planu.
- `Ctrl+Z` privremeno zaustavlja komandu; zatim `bg` može da je nastavi u pozadini.
- `fg` vraća pozadinski ili zaustavljeni posao u prednji plan.

## 4.6 Direktorijumi i stek putanja: `pushd` / `popd`

`pushd` menja direktorijum i pamti prethodni na steku. `popd` uklanja vrh steka i vraća se na tu putanju.

```
pushd ~/Projekti/aplikacija
radi u tom direktorijumu
popd
```

Pregled steka:

```
dirs -v
```

Na mnogim Bash instalacijama `pushd` bez argumenata menja mesta između tekuće i prethodne putanje.

## 4.7 Preusmeravanje i povezivanje komandi

Zapis	Značenje
>	Upisuje izlaz u fajl, prepisujući ga
>>	Dodaje izlaz na kraj fajla
<	Prosleđuje fajl kao ulaz
2>	Preusmerava standardni izlaz za greške
&>	Preusmerava standardni izlaz i greške u Bash-u
\	Prosleđuje izlaz jedne komande kao ulaz drugoj
&&	Pokreće sledeću komandu samo ako je prethodna uspela
\  \	Pokreće sledeću komandu samo ako je prethodna pala
;	Pokreće komande redom, bez obzira na uspeh prethodne

```
ls -lah > spisak.txt
echo 'dodatni red' >> spisak.txt
grep 'ERROR' app.log 2> greske.txt
komanda > sve.txt 2>&1
printf '%s\n' *.txt | sort
```

&& je koristan za niz koraka u kom se svaki sledeći izvršava samo ako je prethodni uspeo:

```
gcc foo.c -o test && ./test && ./package
```

Uzastopne komande mogu da se grupišu i vitičastim zagradama; razmaci su obavezni:

```
{ echo 'Početak'; date; } > izvestaj.txt
```

## 4.8 Skraćenice i posebne promenljive (Bash)

Zapis	Značenje
\$\$	PID trenutne Bash ljuke
\$_	PID poslednjeg procesa pokrenutog u pozadini
\$?	Izlazni status poslednje komande (0 obično znači uspeh)
\$#	Broj argumenata prosleđenih skripti ili funkciji
\$0	Ime skripte ili ljuke
\$1, \$2, ...	Prvi, drugi i sledeći argument
"\$@"	Svi argumenti, svaki kao poseban element; uglavnom najbolji izbor za prosleđivanje argumenata
"\$*"	Svi argumenti spojeni u jedan niz

Zapis	Značenje
<code>\$_</code>	Poslednji argument prethodne komande u Bash-u
<code>!!</code>	Prethodna komanda, proširuje se pre izvršavanja
<code>!\$</code>	Poslednji argument prethodne komande, proširuje se pre izvršavanja

Primer skripte koja koristi argumente:

```
#!/usr/bin/env bash
printf 'Skripta: %s\n' "$0"
printf 'Broj argumenata: %s\n' "$#"
printf 'Prvi argument: %s\n' "${1:-nije prosleđen}"
printf 'Svi argumenti:\n'
printf ' - %s\n' "$@"
```

Bezbednije je obično koristiti `"$@"` umesto `$$` ili `$$*`, jer navodnici čuvaju argumente koji sadrže razmake.

## 4.9 Bash prečice sa tastature

Prečica	Radnja
<code>Ctrl+C</code>	Prekida komandu
<code>Ctrl+Z</code>	Zaustavlja komandu i vraća prompt
<code>Ctrl+D</code>	Šalje EOF; u praznom promptu obično zatvara ljusku
<code>Ctrl+L</code>	Čisti prikaz terminala
<code>Ctrl+A</code>	Prelazi na početak reda
<code>Ctrl+E</code>	Prelazi na kraj reda
<code>Alt+B / Alt+F</code>	Pomeranje unazad / unapred za reč
<code>Ctrl+R</code>	Pretražuje istoriju komandi
<code>Tab</code>	Dopunjava ime komande, fajla ili putanje
<code>↑ / ↓</code>	Kreće se kroz istoriju komandi

## 4.10 Mreža, arhive i pomoćne komande

Komanda	Namena	Primer
<code>curl</code>	Preuzima podatke ili šalje HTTP zahteve	<code>curl -L -o arhiva.zip 'https://primer.test/a.zip'</code>

Komanda	Namena	Primer
wget	Preuzima fajlove	wget 'https://primer.test/fajl.tar.gz'
ssh	Povezuje se na udaljeni računar	ssh korisnik@server
scp	Kopira fajlove preko SSH-a	scp fajl.txt korisnik@server:/tmp/
tar	Pravi ili raspakuje arhive	tar -xf arhiva.tar.gz
zip / unzip	Kreira ili raspakuje ZIP arhive	unzip arhiva.zip
man	Otvora priručnik komande	man find
command -v	Prikazuje putanju komande ako je dostupna	command -v python3
history	Prikazuje istoriju komandi	history
clear	Čisti prikaz terminala	clear

## 4.11 Niz komandi

Kompajliranje C programa, pokretanje testa i pakovanje rezultata:

```
gcc foo.c -o test &&
./test &&
mkdir -p package &&
tar -czf package/test.tar.gz test
```

Ako kompilacija ili test ne uspeju, sledeći koraci se neće izvršiti zbog &&.



## Glava 5

# Efektivno pisanje Bash skripti

Efikasna dijagnostika sistema i automatizacija rutinskih zadataka zahtevaju razumevanje načina na koji se komandna linija i alati koriste za rešavanje problema. Ovo poglavlje prikazuje jednostavne i napredne tehnike Bash skriptovanja.

### 5.1 Zaglavlje

Zaglavlje (eng. *shebang*) je prvi red skripte, koji počinje znakovima `#!`. On određuje koji program treba da izvrši skriptu kada se pokrene kao izvršna datoteka.

```
#!/bin/bash
#!/usr/bin/python
#!/usr/bin/env bash
```

Skript se nalazi u fajlu `01_shebang`.

### 5.2 Argumenti i zamene parametara

Pozicioni parametri predstavljaju argumente prosledene skripti: `$0` je ime skripte, `$1`, `$2` i ostali su pojedinačni argumenti, a `$#` je njihov broj. `"$@"` prosleđuje svaki argument kao zasebnu celinu, uključujući argumente sa razmacima.

```
$./02_args first "second arg"
```

```
Script: ./02_args
Argument count: 2
Argument: <first>
Argument: <second arg>
```

Skript se nalazi u fajlu 02\_args.

Napomena: *Proslediti argument koji sadrži razmak*

### 5.2.1 Zamene

Primer pokazuje podrazumevane vrednosti, broj znakova i izdvajanje dela teksta:

```
$./03_substitutions
Hello, Guest
File: report.txt
Uppercase name: GUEST
First three characters: Gue
Character count: 5
```

Skript se nalazi u fajlu 03\_substitutions.

Oblici zamene parametara koji se često koriste:

```
${var:-default} # Koristi default ako je var nedefinisan ili prazan.
${var-default} # Koristi default samo ako var nije definisan.
${var:?message} # Ispiši poruku i završi ako je var nedefinisan ili prazan.
${var:=default} # Dodeli default ako je var nedefinisan ili prazan.
```

### 5.2.2 Zamena komande i aritmetička ekspanzija

Zamena komande čuva standardni izlaz komande u promenljivoj. Koristite oblik `$(...)`, koji je pregledniji od starih povratnih navodnika (```).

```
$./04_command_subst
Today: 2024-01-15
Files in current directory: 5
```

Skript se nalazi u fajlu 04\_command\_subst.

Aritmetička ekspanzija služi za celobrojne izraze:

```
$./05_arithmetic
count=11, next=8
```

Skript se nalazi u fajlu 05\_arithmetic.

## 5.3 Provera argumenata i parsiranje opcija

### 5.3.1 Duge opcije ručnim upravljanjem

`getopts` obrađuje kratke opcije. Za jednostavan skup dugih opcija može se koristiti petlja:

```
$./06_parse_long --verbose --output result.txt input.txt
```

```
Input: input.txt
Output: result.txt
```

Oznaka `--` završava obradu opcija. Korisna je kada ime datoteke počinje crtičom:

```
$./06_parse_long -- -odd-filename.txt
```

Skript se nalazi u fajlu `06_parse_long`.

### 5.3.2 Kratke i duge opcije zajedno

`getopts` obrađuje kratke opcije. Za kombinovano parsiranje kratkih i dugih opcija koristi se `getopt`:

```
$./07_parse_options -v -o result.txt input.txt
$./07_parse_options --verbose --output result.txt input.txt
$./07_parse_options -o result.txt --verbose input.txt
```

Skript se nalazi u fajlu `07_parse_options`.

## 5.4 Uslovi i status komande

Status komande je 0 kada je uspešna, a različit od nule kada nije. Na primer, `grep` vraća status 1 kada ne pronađe podudaranje.

```
$./08_command_status
```

```
Errors found
```

Skript se nalazi u fajlu `08_command_status`.

Uslovi za datoteke mogu da se izraze pomoću `[ ]` operatora:

```
$./09_file_conditions /tmp
```

```
Directory: /tmp
```

Skript se nalazi u fajlu `09_file_conditions`.

Operatori `&&` i `||` pokreću sledeću komandu u zavisnosti od statusa prethodne:

```
mkdir -p output && printf 'Output directory is ready\n'
```

## 5.5 Funkcije, nizovi i petlje

### 5.5.1 Funkcije i statusi

Funkcija vraća status poslednje izvršene komande, osim ako status nije izričito postavljen pomoću `return`. `local` ograničava promenljivu na funkciju.

```
$./10_functions /etc/passwd
```

```
File is readable
```

Skript se nalazi u fajlu `10_functions`.

### 5.5.2 Nizovi

Nizovi čuvaju listu vrednosti bez gubitka razmaka ili posebnog značenja znakova kao što je `*`.

```
$./11_arrays
```

```
There are 3 entries
Entry: <first file.txt>
Entry: <second.txt>
Entry: <*.log>
All entries: <first file.txt> <second.txt> <*.log> <new file.csv>
```

Skript se nalazi u fajlu `11_arrays`.

Koristite "`${array[@]}`" kada treba da se svaki element prosledi kao poseban argument.

### 5.5.3 Bezbedna obrada imena datoteka

Imena datoteka mogu da sadrže razmake, tabulate, pa čak i nove redove. Kombinacija `find -print0` i `read -d ''` bezbedno obrađuje takva imena.

```
$./12_safe_filenames
```

Skript se nalazi u fajlu `12_safe_filenames`.

## 5.6 Redirekcija ulaza i izlaza

Proces obično ima tri standardna toka: standardni ulaz (0), standardni izlaz (1) i standardni izlaz za greške (2).

```
$./13_redirections
```

Skript se nalazi u fajlu 13\_redirections.

Redosled redirekcija je važan. > `run.log 2>&1` prvo preusmerava standardni izlaz u datoteku, a zatim preusmerava izlaz za greške tamo gde je standardni izlaz trenutno usmeren.

### 5.6.1 “Here”-dokument i “here”-niska

“Here”-dokument prosleđuje više redova kao standardni ulaz:

```
$./14_here_doc
```

```
Hello, class.
This text is supplied on standard input.
```

Skript se nalazi u fajlu 14\_here\_doc.

Ako je oznaka razgraničenja pod navodnicima, promenljive i komande se ne proširuju:

```
$./15_here_doc_literal
```

```
This is literal text: $HOME is not expanded.
```

Skript se nalazi u fajlu 15\_here\_doc\_literal.

“Here”-niska prosleđuje jedan tekstualni niz, uz završni novi red:

```
$./16_here_string
```

```
haystack with a needle
```

Skript se nalazi u fajlu 16\_here\_string.

## 5.7 Cevovodi i tee

Cevovod (`|`) prosleđuje standardni izlaz jedne komande na standardni ulaz sledeće.

```
$./17_pipes
```

Skript se nalazi u fajlu 17\_pipes.

`tee` istovremeno prikazuje ulaz na standardnom izlazu i upisuje ga u datoteku. Sa opcijom `-a` dodaje sadržaj na kraj datoteke.

```
$./18_tee_log
```

Skript se nalazi u fajlu 18\_tee\_log.

Primer beleženja izlaza komande:

```
$./19_pipefail
```

Skript se nalazi u fajlu 19\_pipefail.

Bez `pipefail`, status cevovoda je obično status poslednje komande. Sa `pipefail`, cevovod prijavljuje neuspeh ako ne uspe neka od komandi u njemu.

## 5.8 Privremene datoteke i čišćenje

`mktemp` kreira jedinstveno ime privremene datoteke. `trap` pokreće funkciju za čišćenje pri izlasku iz skripte.

```
$./20_temp_files
```

Skript se nalazi u fajlu 20\_temp\_files.

Primer sa privremenim direktorijumom:

```
$./21_temp_dir
```

Skript se nalazi u fajlu 21\_temp\_dir.

Oznaka `--` ukazuje da se argumenti opcija završavaju. To sprečava da se putanja koja počinje crticom protumači kao opcija komande.

## 5.9 Signali

`trap` povezuje komandu ili funkciju sa signalom ili završetkom skripte. Ctrl-C obično šalje signal `INT`; signal `TERM` traži da se proces zaustavi.

```
$./22_signals &
```

Pritisnite **Ctrl+C** da zaustavite skriptu.

Skript se nalazi u fajlu 22\_signals.

Funkcija za čišćenje treba da bude jednostavna i bezbedna za pozivanje tokom završavanja skripte.

## 5.10 Pozadinski procesi

Znak `&` pokreće komandu u pozadini. PID poslednjeg pokrenutog pozadinskog procesa dostupan je u promenljivoj `$!`.

```
$./23_background
```

```
Started jobs 5678 and 5679
First job exit status: 0
Second job exit status: 0
```

Skript se nalazi u fajlu 23\_\_background.

`wait` PID čeka završetak procesa i vraća njegov status. Sačuvajte `$!` odmah nakon pokretanja procesa, jer se vrednost menja nakon sledeće pozadinske komande.

### 5.10.1 Pokretanje više zadataka i provera statusa

```
$./24_multi_task
```

```
alpha: starting
beta: starting
beta: finished
alpha: finished
All tasks succeeded
```

Skript se nalazi u fajlu 24\_\_multi\_\_task.

Izlaz pozadinskih procesa može da se prepliće. Da biste sačuvali izlaz svakog zadatka zasebno, redirektujte ga u posebnu datoteku:

```
$./25_separate_output
```

Skript se nalazi u fajlu 25\_\_separate\_\_output.

### 5.10.2 Ograničenje broja paralelnih zadataka

`wait -n` čeka da se završi bilo koji od navedenih pozadinskih zadataka. Primer ograničava broj istovremenih zadataka na dva.

```
$./26_parallel_limit
```

```
Processing alpha
Processing beta
alpha: finished
Processing gamma
Processing delta
beta: finished
All tasks finished
```

Skript se nalazi u fajlu 26\_\_parallel\_\_limit.

`wait -n` je dostupan u novijim verzijama Bash-a.

### 5.10.3 Zaustavljanje pozadinskih procesa tokom čišćenja

Skripta koja pokreće pozadinske procese treba po potrebi da ih zaustavi i sačeka pri izlasku.

```
$./27_cleanup_bg &
```

Pritisnite **Ctrl+C** da zaustavite skriptu.

Skript se nalazi u fajlu `27_cleanup_bg`.

Ovaj primer šalje signal za prekid i čeka procese. Složenije skripte možda moraju da vode računa i o procesima koje su pokrenuli njihovi pozadinski zadaci.

## 5.11 Potprocesi i zamena procesa

Komande u zagradama pokreću se u potprocesu (eng. *subshell*). Promena direktorijuma ili vrednosti promenljive unutar potprocesa ne menja roditeljski proces.

```
$./28_subshell
```

```
Before: /home/user
Inside subshell: /tmp
After: /home/user
```

Skript se nalazi u fajlu `28_subshell`.

Zamena procesa omogućava da se izlaz komande koristi kao datoteka slična ulazu. U ovom primeru, `diff` dobija dva zasebna ulaza:

```
$./29_proc_subst
```

Skript se nalazi u fajlu `29_proc_subst`.

## 5.12 Primer: brojanje podudaranja

Ova skripta kombinuje parsiranje opcija, proveru ulazne datoteke, privremenu datoteku, čišćenje, obradu izlaznog statusa komande i opcioni izlaz u datoteku.

```
$./30_count_matches -v ERROR application.log
$./30_count_matches -o matches.txt ERROR application.log
```

Skript se nalazi u fajlu `30_count_matches`.

## 5.13 Zadaci za vežbu

- Dodajte opciju `-n` koja ograničava broj prikazanih rezultata.

- Izmenite primer sa `tee` tako da dodaje sadržaj u dnevnik umesto da ga zamenjuje.
- Dodajte proveru da se ulazna datoteka razlikuje od izlazne.
- Izmenite primer sa pozadinskim procesima tako da svaki zadatak upisuje rezultat u zasebnu datoteku.
- Testirajte skripte sa praznim argumentima, nepostojećim datotekama i imenima koja sadrže razmake.

