ADMINISTRACIJA PROCESA

System Startup

1. What occurs from the power-off position until your operating system is fully available is called the boot process. In the simplest terms, the boot process consists of the Read-Only Memory's (ROM, or NVRAM, or firmware) loading of the program for actually booting (starting) the system. This initial step (commonly called **bootstrapping**) identifies the devices on the system that can be booted or started from. You can boot or start from only one device at a time, but, because many different devices can be identified as bootable, one of those other identified devices can be used if one bootable device has a failure. These devices may load automatically, or you may be shown a list of devices from which you can choose.
   The boot device doesn't have to be a physical hard drive, because the system can boot from the network or from removable storage such as a CD-ROM or floppy diskette. A boot device simply holds the information about where to load the operating system. The bootstrap phase only identifies the hardware available for booting and whether it is usable.

2. **Control is then transferred to the kernel**. The operating system has not been loaded at this point, and the system is not usable for production processes. Some systems show the boot process by means of messages on the screen, and others hide the system messages from the users by using graphical figures to represent the boot process.
   After the initial bootstrapping, the boot program begins loading a kernel, which typically resides in the root partition of the system. The kernel on most Unix systems is called unix; in Linux systems, it might be called `vmunix` or `vmlinuz`.
   The kernel's initial tasks, which vary according to hardware and Unix version, are followed by the initialization phase, in which the system processes and scripts are started. The **`init`** process **is the first job started and is the parent of all other processes**. It must be running for the system to run. The init process calls the initialization scripts and completes administrative tasks relative to the system, such as starting sendmail, the X or window server (that provides the graphical user interface), and so on. The init process looks into the initialization specification file, usually called `/etc/inittab`. This file identifies how init should interpret different **run levels** and what scripts and processes should be started in each run level. A run level is a grouping of processes (programs in the most basic sense) or daemons (processes that run all the time).

Kreiranje procesa

Linux koristi standardni UNIX proces mehanizam (**fork**) koji razdvaja kreiranje procesa i njegovo izvršenje u dve različite operacije:

- sistemski poziv **fork**, koji kreira novi proces,
- sistemski poziv **exec**, koji izvršava program u resursima novostvorenog procesa.

Pod UNIX sistemom sve informacije koje operativni sistem mora čuvati da bi kontrolisao jedan proces predstavljaju kontekst tog procesa. Pod Linux operativnim sistemom, svaki proces je u potpunosti opisan identitetom, okolinom, i kontekstom.

Identitet procesa obuhvata sledeće informacije:

- Identifikator procesa (Process ID - PID);
- Akreditivi (Credentials)- Svaki proces pripada jednom korisniku koji ima svoj user ID i jedan ili više grupnih IDs koji određuju prava pristupa procesu u radu sa datotekama;
- Ličnost (Personality). Ova informacija se ne koristi kod drugih UNIX sistema, a Linux svakom procesu dodeljuje lični identifikator koji može imati uticaja za neke sistemske pozive.

Okolina procesa se nasleđuje od procesa roditelja (odnosno od procesa koji je izazvao kreiranje datog procesa). U okolinu procesa spadaju vektor argumenata koje proces roditelj prosleđuje programu i vektor okoline, odnosno lista promenljivih koje definišu okolinu procesa (environment).

Kontekst procesa je stanje procesa u datom trenutku vremena. Kontekst procesa čine sledeće komponente:

- kontekst za raspoređivanje (scheduling context), koji služi za efikasnu suspenziju ili ponovni start procesa. Obuhvata sve CPU registre, prioritet procesa i kernelski stek procesa;
- statistički kontekst, koji sadrži informacije o resursima koje proces koristi u jednom trenutku, kao i kompletnu upotrebu resursa za vreme trajanja jednog procesa accounting information);

- tabela datoteka (file table), tj. polje ukazivača na kernelske strukture datoteka;
- kontekst sistema datoteka (file-system context);
- tabela za upravljanje signalima (signal-handler table), koja definiše ukazivače na programe koji se pozivaju nakon određenog signala;
- kontekst virtulene memorije (virtual-memory context), koji potpuno opisuje korišćenje memorije od strane procesa.

Kreiranje niti

Linux koristi istu internu reprezentaciju za procese i niti - nit (thread) je jednostavno novi roces koji deli adresni prostor roditelja. Za razliku od novog procesa koji pomoću sistemskog poziva fork formira novi kontekst sa jedinstvenim adresnim prostorom, nit nastaje sistemskim pozivom `clone` koji kreira novi kontekst, ali dozvoljava novom procesu da deli adresni prostor roditelja.

PROCESI, RODITELJI i DECA

A *process*, in simple terms, is an instance of a running program.
Svaki program koji korisnik pokrene kreira proces. Procesima se dodeljuju jednoznačni numerički identifikatori (PID, Process Identification Number), koje sistem dalje koristi za identifikaciju i praćenje procesa do kraja njegovog izvršenja.
Proces pokrenut pod Linux sistemom može u toku svog rada pokrenuti podproces, sam proces koji kreira/pokreće novi postaje proces roditelj, a podproces predstavlja proces-dete.

> Npr., novi proces se kreira ukoliko korisnik želi da koristi vi editor - program koji je pokrenut kreira proces-dete, čiji je proces-roditelj komandni interpreter. Proces roditelj, obezbeđuje okolinu (environment) koja je neophodna za normalno izvršenje procesa, i u ovom slučaju (konandni interpreter - vi) prelazi u suspendovano stanje (WAIT) za vreme izvršenja podprocesa. Kada podproces normalno ili nasilno završi svoj rad (na primer, korisnik prekine njegovo izvršenje) proces roditelj preuzima kontrolu i komandni prompt se vraća korisniku. Ukoliko se podproces smesti u pozadinu (background), komandni interpreter ne čeka kraj izvršenja procesa u suspendovanom stanju. Komandni prompt se odmah vraća, a korisnik može zadati drugu komandu i na taj način pokrenuti drugi proces koji će se izvršavati paralelno ili kvazi-paralelno sa prethodnim. Jednostavne interne komande, poput `cd`, izvršava komandni interpreter - za njihovo izvršenje se ne kreira novi proces.

Behind the scenes, a *fork library call* or an *execve system call* is used to start the new program.

- **A fork is produced when the current running program is copied to make a child**, an exact copy of the running program. The forked program has a new PID and a different parent process ID (of course), and the child's resource utilizations are all reset. For example, by default, the forked child and its parent share file descriptors and can share open files.
- In contrast to forking a process, you can replace **the current running process with a new process**. The Unix shell includes a built-in command called exec that replaces the running shell with a new program. (Behind the scenes, this uses the execve system call.) For example, typing `exec date` will run the date program, and the original shell will be closed.

The parent of all processes, `init`, is started when the operating system boots. Historically, it is process ID number 1. As other programs are started, each is assigned a unique process identifier, PID.
`init` je zadužen za upravljanje ostalim procesima. Proces init je proces roditelj procesa login koji proverava unešeno korisničko ime i lozinku, a zatim pokreće proces shell, odnosno komandni interpreter. Init čeka u suspendovanom stanju da korisnik završi svoj rad u komandnom interpreteru i da se odjavi sa sistema, nakon čega opet preuzima kontrolu i pokreće novi login proces.

Zavisno od mesta u hijerarhiji procesa, načina izvršenja i trenutne funkcionalnosti, procesi mogu pripadati sledećim kategorijama:

- **Daemon** - procesi koje je pokrenuo kernel i koji se izvršavaju u pozadini. Na primer, lpd (line printer scheduler) se najčešće pokreće pri podizanju sistema s namenom da prihvata poslove za štampu i da njima upravlja. Ukoliko nema zahteva za štampu, lpd se izvršava ali je

neaktivan. Kada neko pošalje zahtev za štampu lpd postaje aktivan dok se zahtev ne odštampa.

- **Proces roditelj** (Parent) - proces koji kreira podproces je proces roditelj. Proces roditelj obezbeđuje okolinu koja je neophodna za normalno izvršenje procesa, i može preći u suspendovano stanje za vreme izvršenja podprocesa. Osim procesa init svi procesi imaju svoje roditelje.
- **Podproces** (Child) - proces koji pokreće proces roditelj.
- **Orphan** - ukoliko korisnik pokrene komandu iz terminala u grafičkom okruženju i zatvori prozor pre nego što komanda završi svoj rad, proces koji je komanda kreirala postaje siroče (orphan), odnosno proces bez roditelja. Ovakvi procesi mogu nastati u svim situacijama kada se nasilno prekine izvršenje proces roditelja. Da bi seodržala hijerarhijska struktura procesa `init` kao glavni proces "usvaja" sve siročiće a zatim prekida njihovo izvršenje.
- **Zombie** (**Defunct**) - proces koji izgubi vezu s procesom-roditeljem ostaje izgubljen u sistemu, a jedini resurs koji troši jeste jedno mesto u tabeli procesa. Defunct procesi ne utiču na pad performansi računara i uklanjaju se prilikom sledećeg podizanja operativnog sistema, ako ih prethodno ne ukloni init ili vi.

DOBIJANJE INFORMACIJA O PROCESIMA

ps

Komanda ps (process status) prikazuje na ekranu listu sa podacima o aktivnim procesima (PID procesa, ime komande kojom je proces iniciran itd.).

```
$ ps
PID TTY TIME CMD
18358 ttyp3 00:00:00 sh
18361 ttyp3 00:01:31 abiword
18789 ttyp3 00:00:00 ps
```

```
$ ps x
```

pored prethodinh daje i informaciju o statusu procesa. Neka od stanja

State	Function
I	Idle, sleeping for more 20 seconds
D	Waiting for disk or other uninterruptible wait
R	Runnable, active use
S	Sleeping for less than 20 seconds
T	Stopped or traced process
Z	Zombie process; a dead or defunct process

Depending on the Unix system and the type of ps command installed, you may have different or additional state identifiers. Be sure to read the ps(1) man page for details.

Ako se komanda pokrene nekoliko puta može se jednostavnim upoređivanjem utrošenog procesorskog vremena utvrditi da li je proces aktivan ili ne. Ukoliko procesorsko vreme raste, proces je aktivan.
Tri najčešće koriććena argumenta komande ps su -e (every process), -f (full listing), i -u (user).

- `ps -e` ps prikazuje `PID`, `TTY`, `TIME` i `CMD` svih procesa na sistemu .
- `ps -f` ps prikazuje dodatne informacije o svim procesima koji su pokrenuti iz tekućeg shell konteksta ili Terminal prozora. U dodatne informacije spadaju: ID korisnika koji je pokrenuo komandu koja je inicirala proces (UID), identifikator proces roditelja (PPID), prioritet procesa (C) i vreme kada je proces počeo sa izvršenjem (STIME).
- ps -u UID ps prikazuje PID, TTY, TIME i CMD svih procesa koje je inicirao korisnik čiji je UID naveden kao parametar.

Komanda ps -ef kombinuje argumente -e i -f, odnosno prikazuje detaljne informacije o svim procesima na sistemu.
U polju TTY procesa koje je inicirao korisnik koji je lokalno prijavljen na sistem i koji ne koristi

grafičko okruženje stajaće `tty#`. U polju TTY procesa koje je inicirao korisnik koji radi u grafičkom okruženju ili je prijavljen na sistem preko mreže, stoji `pts#`. **Pseudoterminal (pts**) je ime uređaja dodeljenog remote login sesijama i prozorima. Svaki prozor koji korisnik otvori nakon prijavljivanja na sistem dobija novi pts#.

Da bi korisnik uništio neki proces neophodno je da poznaje njegov PID. Na većini sistema izvršava se veliki broj procesa, tako da je listing komande ps -ef dugačak. Ukoliko je ime komande kojom je proces iniciran poznato, PID se lako može odrediti korišćenjem ps u pipe sprezi sa komandom grep.

```
$ ps -ef | grep tuxracer
```

Komanda ps -ef prikazuje detaljne informacije o svakom procesu, uključujući i identifikator proces roditelja PPID. **PPID** se koristi u slučajevima kada nije dovoljno uništiti proces u kom se izvršava blokirana aplikacija, već i njen proces roditelj. Ukoliko se proces roditelj uništi prvi, svi podprocesi se automatski uništavaju.

Sistemski procesi se mogu videti koričćenjem opcije a:

```
$ ps ax
```

Dodatni argumenti komande `ps` zavise od same distribucije Linux sistema.
Koristan argument komande `ps` je `--forest` kojim se zahteva prikazivanje hijerarhijskog stabla procesa.

```
$ ps -e --forest
```

Atributi procesa
Each process has an environment with various attributes such as command-line arguments, user environment variables, file descriptors, working directory, file creation mask, controlling terminal (console), resource limitations, and a lot more. Many of the attributes are shared with the parent process. The kernel also knows about, and can report, numerous other process attributes. On a NetBSD box, the kernel keeps track of around 85 different attributes for each process. Some examples include reporting the process ID of its parent, the real and effective user and group IDs, the time the process was invoked, and resource utilization, such as memory usage and total time spent executing the program. A real user or group ID is generally the user and group that initially started the program. The effective user or group ID is when a process is running with different (such as enhanced) permissions.
To view the various process attributes, you can use the
`ps -o switch`,

It is used for defining the output format. For example, to output the process IDs, parent PIDs, and the size of the processes in virtual memory of your own running processes, issue the command:

```
$ ps -o user,pid,ppid,vsz,comm
```

The following are commonly used output format fields:

Field	**Definition**
user	Effective user ID of the process
pid	Process ID
ppid	Process ID of the parent
pcpu	Percentage of CPU time used
rss	Real memory size in kilobytes
pmem	Percentage of rss to physical memory
vsz	Kilobytes of the process in virtual memory
tty	Controlling terminal name
state (or s)	Process state
stime	Time started

time	Accumulated user and system CPU time
command (or comm)	Command name

`pstree`

A process tree displays the lineage of your different processes, placing a child process with its parent. Note that only one parent process exists per child process but that each parent can have multiple children.

```
$ pstree
```

`time`

Komandom time se određuje vreme potrebno za izvršenje komande, odnosno procesa. Komanda na ekranu prikazuje tri vremena: realno (real), sistemsko (system) i korisničko (user). Realno vreme obuhvata interval od zadavanja komande do potpunog izvršenja i povratka komandnog prompta, uključujući i vreme čekanja na ulaz, izlaz i ostale događaje. Korisničko vreme je količina procesorskog vremena utrošena na samo izvršenje procesa. Sistemsko vreme je vreme koje je kernel utrošio na opsluživanje procesa.

```
# time ls -lR /etc | sort > /dev/null
real 0m4.332s
user 0m0.580s
sys 0m0.110s
```

SLANJE SIGNALA I UNIŠTENJE PROCESA

Signali

Zavisno od implementacije u svakom UNIX sistemu je definisano 30 do 40 signala, od kojih je svaki predstavljen imenom i brojem. Slanje signala je metod komunikacije sa procesima - signali se mogu posmatrati kao kratke poruke specijalnog značenja koje se šalju procesima, koje procesi dalje prihvataju ili ignorišu.
Signali se koriste za uništenje, privremeni prekid i nastavak izvršenja procesa. Na primer, kombinacija tastera <Ctrl-C> može da uništi proces koji više nije pod kontrolom. Kada korisnik pritisne <Ctrl-C> aktivnom procesu se šalje signal prekida INT, nakon čega se uništavaju aktivni proces i svi podprocesi koje je on inicirao.

Signali se mogu klasifikovati u dve osnovne kategorije:
- signali za kontrolu procesa, koji se mogu koristiti bez obzira na trenutni komandni interpreter,
- signali za kontrolu posla, koji se mogu koristiti samo ako komandni interpreter podržava kontrolu posla.

U signale za kontrolu procesa spadaju:
- TERM Uništenje procesa (terminate) - proces može da ignoriše ovaj signal
- KILL Neopozivo uništenje procesa - proces ne može da ignoriše ili blokira ovaj signal
- HUP Uništenje procesa koji se izvršavaju u pozadini prilikom odjavljivanja korisnika sa sistema (hang up)
- INT Interaktivni signal prekida (interrupt), koji generiše INTR kontrolni karakter
- QUIT Interaktivno uništenje procesa, koje generiše QUIT kontrolni karakter

Podrazumevana akcija koja se izvršava kao posledica signala za kontrolu procesa je uništenje procesa. Procesi mogu da ignorišu sve signale osim signala KILL, koji se koristi kao poslednja mera pri uništenju procesa. Ne može se ta□no odrediti koji proces ignoriše koje signale.

U signale za kontrolu posla spadaju:

- STOP Zaustavljanje procesa - proces ne može da ignoriše ili blokira ovaj signal
- CONT Nastavi izvršenje zaustavljenog procesa - proces ne može da ignoriše ili blokira ovaj signal
- TSTP Interaktivno zaustavljanje procesa, koje generiše SUSP kontrolni karakter
- TTIN Posao u pozadini pokušava da izvrši akciju □itanja - grupa procesa je suspendovana
- TTOU Posao u pozadini pokušava da izvrši akciju upisa - grupa procesa je suspendovana

Podrazumevana akcija koja se izvršava kao posledica signala za kontrolu posla (izuzev signala CONT) je zaustavljanje, odnosno suspenzija procesa. Procesi mogu ignorisati sve signale osim signala STOP i CONT, tako da korisnik uvek može da zaustavi proces i nastavi izvršenje procesa.

Osim signala KILL i STOP, proces može da "uhvati" (catch) signal, odnosno da izvrši neku drugu akciju umesto podrazumevane. Proces koji hvata signal može da odluči koju će akciju da izvrši kao posledicu datog signala. Na primer, proces koji primi TERM signal može regularno da završi svoj rad (da najpre završi obradu podataka i upiše rezultate na disk, a zatim da prekine izvršenje). Ukoliko proces ne ignoriše ili ne hvata signal, izvršava se podrazumevana akcija.

Uništenje procesa
Izvršenje procesa koji radi u prvom planu (foreground) najlakše se može prekinuti slanjem signala prekida (INT). Podrazumevana kombinacija tastera kojom se signal INT šalje je <Ctrl-C>, što se može redefinisati komandom stty int karakter. Aktuelna kombinacija se može videti u izlazu komande

```
$ stty -a
```

Proces može ignorisati INT signal, tako da ovaj na□in uništenja procesa ne uspeva u svim slučajevima. Alternativni način uništenja je slanje signala QUIT (signal za interaktivno uništenje procesa). Podrazumevana kombinacija tastera kojom se signal **QUIT** šalje je **<Ctrl-\>**, što se može redefinisati komandom `stty quit` karakter. Proces može ignorisati i ovaj signal. Ukoliko komandni interpreter podržava kontrolu posla, proces se može suspendovati, a zatim uništiti slanjem signala pomoću komande kill. Ukoliko se proces ne može uništiti ni na jedan prethodno pomenut način, korisnik mora da se prijavi na sistem sa druge virtuelne konzole, terminal prozora ili preko mreže, a zatim iskoristi komande `ps` i `kill` za uništenje procesa.

Procesi koji se izvršavaju u pozadini mogu se uništiti iz tekućeg komandnog interpretera.

`kill`
Komanda kill se koristi za slanje signala procesima. **Signale procesima mogu da šalju vlasnici i superuser**, što znači da samo onaj korisnik koji je pokrenuo komandu kojom je proces iniciran ima pravo da taj proces zaustavi ili uništi, dok root može da zaustavi ili uništi bilo koji proces na sistemu.
Pre slanja signala neophodno je identifikovati proces komandama `ps` ili `jobs`. Komanda ps je prisutna na svim UNIX sistemima, dok je komanda jobs dostupna samo u komandnim interpreterima koji podržavaju kontrolu posla. Uništenje procesa izvršava se u tri koraka:

- određivanje PID procesa koji je potrebno uništiti komandom ps. Napomena: ukoliko se umesto PID koristi PPID procesa, biće uništeni svi procesi čiji je PPID proces roditelj;
- slanje signala procesu komandom kill. Komanda zahteva PID procesa kao argument, a dodatno se može navesti i signal koji je potrebno poslati. Signal se može navesti pomoću broja ili imena - preporučuje se navođenje putem imena, jer numeričke vrednosti variraju od sistema do sistema. Ukoliko se signal ne navede, podrazumeva se TERM;
- provera liste procesa, odnosno utvrđivanje da li je signal uništio proces ili ne.

Nekoliko signala mogu se iskoristiti za uništenje procesa, od kojih neke procesi mogu ignorisati, pa je u nekim slučajevima potrebno više puta zadati komandu kill, odnosno poslati nekoliko različitih signala procesima.
Na primer, ukoliko korisnik želi da uništi proces čiji je PID 2345, potrebno je da zada komandu

`kill -HUP 2345`, i na taj način pošalje signal HUP procesu. Ukoliko se proces ne uništi korisnik može poslati signal TERM, a u krajnjem slučaju i signal KILL (kill -KILL 2345), koji sigurno i neopozivo uništava proces.
Ukoliko neki proces koji se izvršava u pozadini konzumira mnogo procesorskog vremena, a korisnik želi da izvrši neku akciju bez uništenja procesa, potrebno je da pošalje signal STOP procesu (`kill -STOP PID`). **Signal STOP privremeno zaustavlja izvršenje procesa**, čime se procesor oslobaća, tako će se sledeća komanda koju korisnik zada brže izvršiti.
Nakon toga, procesu se može poslati signal CONT (`kill -CONT pid`) čime se nastavlja njegovo izvršenje.

Sintaksa komande kill je:

```
$ kill [-s signal ] PID
```

Dodatno, komanda kill sa parametrom -l prikazuje listu signala:

```
$ kill -l
1) SIGHUP   2) SIGINT   3) SIGQUIT  4) SIGILL
...
```

`killall`
Dodatno, korisnik može koristiti komandu `killall`, koja funkcioniše kao i komanda kill, s tim što kao argument ne zahteva PID procesa, već njegovo ime:

```
$ killall -HUP inetd
```

PROCESI KOJI SE IZVRŠAVAJU U POZADINI, POSLOVI, PRIORITETI

Procesi koji se izvršavaju u prvom planu (foreground)

Komunikacija na relaciji komandni interpreter - korisnik najšešće se odvija po principu izvršavanja jednog zadatka u jednom momentu. Korisnik čeka da program završi s radom i da shell preuzme kontrolu, odnosno da se na ekranu pojavi komandni prompt, pre nego što zada drugu komandu. Izvršenje procesa u prvom planu (**foreground processing**) je podrazumevan način izvršenja procesa za sve komandne interpretere.
Procesi koji zahtevaju interakciju sa korisnikom moraju se izvršavati u prvom planu. Procesi koji ne zahtevaju interakciju sa korisnikom (na primer, programi koji ulazne podatke čitaju iz datoteke ili ih dobijaju od drugih procesa, preko mreže ...) mogu se izvršavati i u prvom planu i u pozadini. Takvi procesi mogu poruke korisniku upisivati u neku log datoteku, ili prikazivati periodično na ekranu, što zahteva periodično izvršenje u prvom planu.

Procesi koji se izvršavaju u pozadini (background)
Korisnik može pokrenuti proces u pozadini dodavanjem znaka **& (ampersand)** na kraj komandne linije. Procesi se u pozadini izvršavaju konkurentno sa svim sistemskim i korisničkim procesima. Procesi se izvršavaju u pozadini ukoliko su vremenski zahtevni, ukoliko troše veliku količinu procesorskog vremena ili ukoliko ne zahtevaju interakciju sa korisnikom.

```
$ sort verybigfile > sortedfile &
[1] 123
```

Grupe procesa i kontrola poslova

Neki komandni interpreteri (kao što je Bourne Again Shell) uvode pojam kontrole poslova, koji je zasnovan na procesnim grupama. **Svaki put kada korisnik zada komandu (ili više komandi spregnutih u pipeline) komandni interpreter kreira jednu grupu procesa.**
Grupu procesa čine oni procesi koji su nastali kao posledica izvršenja jedne komande.
Ukoliko je komanda jednostavna, grupu obično čini jedan proces. Ukoliko se radi o komandama spregnutim u pipeline, grupu čini nekoliko procesa. Komandni interpreter procesnoj grupi dodeljuje identifikator koji je jednak identifikatoru jednog procesa iz grupe.

Posao (job) je grupa procesa koja se izvršava u pozadini. Poslovima se, kao i procesima, dodeljuju celobrojni numerički identifikatori (brojevi poslova - job number). Ukoliko korisnik

pokrene neki posao komandni interpreter na ekranu prikazuje poruku koja uključuje broj posla. Grupa procesa je jako slična poslu - jedina bitna razlika je u tome što svako pokretanje komande rezultuje procesnom grupom, dok se broj posla dodeljuje samo ako se procesna grupa suspenduje ili smesti u pozadinu. Komandni interpreteri koji podržavaju koncept kontrole poslova korisnicima nude izvestan skup komandi koje služe za upravljanje poslovima. Dodatno, postojeće komande za upravljanje procesima (poput komande kill) mogu se upotrebiti i za upravljanje poslovima.
Kontrola poslova obuhvata sledeće operacije:
- pomeranje procesa iz pozadine u prvi plan i obrnuto,
- suspendovanje i nastavak izvršenja procesa.

Za svaki posao i procesnu grupu uvodi se pojam kontrolišućeg terminala (controlling terminal). Kontrolišući terminal je konzola (ili terminal prozor) iz koga je zadata komanda koja je inicirala procesnu grupu, odnosno posao. U svakom terminalu se samo jedan proces može izvršavati u prvom planu.

```
jobs
```

Komanda jobs prikazuje listu svih poslova koji su pokrenuti iz tekućeg komandnog interpretera, bez obzira da li se izvršavaju u pozadini ili su suspendovani.

```
$ jobs
[1] Stopped vi mydoc.txt
[2] - Running sort verybigfile > sortedfile &
[3] + Stopped (tty output) summararize_log &
```

Svaka linija u izlazu komande odgovara jednoj procesnoj grupi, pri čemu je celobrojna vrednost na početku linije broj posla. Broj posla se može koristiti kao argument komande kill, pri čemu se mora koristiti prefiks %.

```
$ kill %1
```

Komandni interpreter takođe vodi evidenciju o trenutnim poslovima, koji su u izlazu komande jobs označeni znakom +, i prethodnim poslovima, koji su u izlazu komande jobs označeni znakom -. Ukoliko je korisnik pokrenuo više poslova, ostali neće imati posebne oznake.
Napomena: trenutni posao i procesna grupa koja se izvršava u prvom planu nisu ista stvar. Posao se izvršava isključivo u pozadini.

Premeštanje poslova u prvi plan (komanda fg)

U komandnim interpreterima koji ne podržavaju kontrolu poslova, proces koji je pokrenut u pozadini (pomoću sufiksa &) ostaje u pozadini dok se ne izvrši ili dok ne primi signal koji prekida rad procesa. Procesna grupa se može premestiti iz pozadine u prvi plan samo ako korisnik radi u komandnom interpreteru koji podržava kontrolu poslova. U tom slučaju, korisnik može pomoću komande fg premestiti izvršenje posla u prvi plan, nakon čega posao postaje procesna grupa.

Komandi fg se kao argument može navesti broj posla (sa prefiksom %) ili PID procesne grupe. Ako se argument ne navede, podrazumeva se trenutni posao. Rezultat izvršenja komande fg je premeštanje posla u prvi plan.

```
$ fg %1
```

ili

```
$ fg 2234
```

Suspendovanje procesne grupe

Procesna grupa se može suspendovati slanjem odgovarajućeg signala iz grupe signala za kontrolu poslova, što se može izvršiti na dva načina:
- kontrolnim karakterima za suspenziju procesa koji se izvršava u prvom planu,
- komandom kill.

Kontrolni karakter za suspenziju procesa (najčešće <Ctrl-Z>) šalje odgovarajući signal procesu koji se izvršava u prvom planu.
Sledeći primer ilustruje suspenziju procesa. Nakon pokretanja komande koja inicira izvršenje procesne grupe u prvom planu, potrebno je pritisnuti <Ctrl-Z> čime se procesna grupa suspenduje i postaje posao. Komandni interpreter na ekranu prikazuje poruku o suspenziji, zaključno sa brojem posla koji je dodeljen suspendovanoj procesnoj grupi.
Nakon toga korisnik može komandama fg i bg nastaviti izvršenje procesa u prvom planu ili u pozadini.

```
$ sort verybigfile > sortedfile
<Ctrl-Z>
[1]+ Stopped sort verybigfile > sortedfile
```

Premeštanje poslova u pozadinu (komanda bg)

Komanda bg premešta suspendovani posao u pozadinu. Posao se najćešće premešta u pozadinu nakon suspenzije, odnosno slanja kontrolnih karaktera za suspendovanje procesne grupe koja se izvršava u prvom planu. Nakon premeštanja u pozadinu posao nastavlja sa izvršenjem u pozadini. Posao dalje ostaje u pozadini dok se ne izvrši, dok ne primi neki signal od korisnika, ili ne pokuša da izvrši ulazno/izlaznu operaciju vezanu za terminal.

```
$ bg %1
[1]+ sort verybigfile > sortedfile &
```

Komanda `wait` i čekanje izvršenja poslova

Komanda `wait` inicira čekanje izvršenja jednog ili svih procesa koji radi u pozadini. Najčešće se koristi u shell programima, ali se može koristiti i interaktivno, ukoliko je potrebno sačekati izvršenje određenog posla, odnosno izvršenje svih poslova ukoliko treba pokrenuti nove.

Ukoliko se komanda wait zada bez argumenata čeka se na sve poslove. Kao argumenti se mogu navesti PID procesne grupe ili broj posla, čime se inicira čekanje na određeni posao.

```
$ job1 &
[1] 20233
$ job2 &
[2] 20234
$ job3 &
[3] 20235
$ job4 &
[4] 20237
$ wait %1
$ wait 20234
$ wait
[4] + Done job4 &
[3] + Done job3 &
$ jobs
$
```

PRIMER.
Pokretanje posla, prikazivanje broja posla, smeštanje posla u prvi plan, suspendovanje procesne grupe koja se izvršava u prvom planu, smeštanje suspendovane procesne grupe u pozadinu i nastavak izvršenja, uništenje posla.

Napomene: broj u uglastim zagradama [x] je broj posla. **Posao koji je obeležen znakom + je trenutni posao i njime se može upravljati komandama fg i bg bez eksplicitnog navođenja** broja posla ili identifikatora procesa.

```
$ sleep 500&
```

```
[1] 6989
$ jobs
[1]+ Running sleep 500 &
$ fg %1
sleep 500
<Ctrl-Z>
[1]+ Stopped sleep 500
$ bg %1
[1]+ sleep 500 &
$ jobs
[1]+ Running sleep 500 &
$ kill %1
$ jobs
[1]+ Terminated sleep 500
```

`top`

The `top` command is a very useful tool for quickly showing processes sorted by various criteria. It is an interactive diagnostic tool that updates frequently and shows information about physical and virtual memory, CPU usage, load averages, and your busy processes. The top command is usually found on *BSD, Linux, and Mac OS X systems. It can also be downloaded at www.groupsys.com/topinfo/ in versions suitable for various other Unix systems.

```
$ top –u student
```

The `/proc` File System

The `/proc` file system is a dynamically generated file system that can be used to retrieve information about processes running on your system. Linux and Solaris systems provide /proc file systems. /proc file systems are also available for other Unix systems but are usually not mounted by default.
The /proc file system contains a directory entry for active processes named after the PID. These directories contain files that provide various attributes about the process. For example, the following is the directory listing of a /proc file system for a personal shell process on a Solaris box:

```
$ ls -l /proc/$$
```

Here is the /proc entry for init on a Linux box:

```
$ sudo ls -l /proc/1
```