

# Osnove korištenja operacijskog sustava Linux

## 05. Procesi i ljuske

Dominik Barbarić

Nositelj: dr. sc. Stjepan Groš

Sveučilište u Zagrebu

Fakultet elektrotehnike i računarstva

08.11.2014

# Sadržaj

## 1 Procesi

- Ispis procesa

## 2 Signali

- Slanje signala
- Child procesi

## 3 Ljuska

- Načini rada ljuske
- Procesi u ljusci

# Proces i program

## ► Program

- Datoteka s izvršnim kodom
- Primjer: `/bin/bash` je program

## ► Proces

- Pokretanjem programa na računalu nastaje *proces*
- Za proces su vezani izvršni kod programa te svi podaci vezani uz izvršavanje programa (podaci u memoriji i u ostalim resursima)
- Svaki proces posjeduje jedinstveni identifikator - PID (*process identifier*)

# Ispis procesa

- ▶ Ispis procesa obavlja se s naredbom `ps`
  - *process status*
  - ako se pokrene bez argumenata i opcija ispisuju se procesi vezani za trenutni terminal
- ▶ `ps` prihvaća tri vrste opcija: Unix, BSD, GNU opcije
- ▶ Zadatak
  - U man stranicama proučiti opcije `ps` naredbe.

# Ispis procesa

- ▶ Ispis naredbe ps sadrži:

PID      Process ID

TTY      TTY (terminal) za koji je proces vezan

TIME     Ukupno vrijeme izvršavanja

CMD      Naredba kojom je proces pokrenut (put do programa,  
bez argumenata)

- ▶ Dodatne informacije o procesima korištenjem opcije -f

UID      Vlasnik procesa

PPID     Roditelj procesa

CMD      Naredba s argumentima

# Ispis procesa

- ▶ Korisne opcije `ps` naredbe
  - e ispis svih procesa u sustavu
  - f dodatne informacije o procesima
  - o zadavanje formata ispisa
  - sort sortiranje ispisa (po PID ako nije drugačije navedeno)

Primjer: Ispis procesa s prikazom navedenih atributa i sortiranjem

```
ps -eo pid,ppid,user,args,nice --sort user
```

# Ispis procesa

- ▶ S argumentom u dobivamo pregled resursa koje procesi zauzimaju
  - %CPU Zauzeće procesora
  - %MEM Zauzeće radne memorije
  - VSZ Veličina virtualne memorije
  - STAT Trenutno stanje procesa
  - START Vrijeme pokretanja procesa

# Stanja procesa

- Procesi mogu poprimiti sljedeća stanja:

**R** *running* - aktivni proces

**S** *sleeping* - neaktivan 20 sekundi ili manje

**I** *idle* - neaktivan više od 20 sekundi

**T** *stopped* - zaustavljen proces

**Z** *zombie* - proces koji je završio, ali je još uvijek upisan u tablicu procesa



# Signali

- ▶ *Signal* je metoda komunikacije između procesa (IPC - Inter-process communication)
- ▶ Pojava signala izaziva prekid u radu procesa
- ▶ Ovisno o vrsti primljenog signala proces ili jezgra operativnog sustava obrađuju prekid
  - Proces može za primljeni signal obraditi funkciju koja se definira kao *signal handler*. Ako nije definirana signal handler funkcija izvršava se zadana radnja (ovisno o vrsti signala)
  - Po završetku obrade proces nastavlja s normalnim izvršavanjem (osim ako u prekidu nije okončan)

# Signal handler primjer

## main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

void signal_handle(int sig_code){
    if (sig_code == SIGINT){
        printf("Primljen INT. Kraj programa!\n");
        exit(0);
    }
    else if (sig_code == SIGUSR1)
        printf("Primljen USR1\n");
}

int main(){
    if (signal(SIGINT, signal_handle) == SIG_ERR)
        exit(1);
    if (signal(SIGUSR1, signal_handle) == SIG_ERR)
        exit(1);

    while(1)
        sleep(1);

    return 0;
}
```

# Signali

- U UNIX operativnim sustavima svaka vrsta signala uz naziv ima i brojčanu oznaku

## Primjeri signala

Naziv signala	Kod	Značenje
SIGHUP	1	<i>Hangup</i> - Ugašen terminal u kojem se proces odvijao
SIGINT	2	<i>Terminal interrupt</i> - Signal s terminala. Aktivira se s CTRL+C
SIGTERM	15	<i>Terminate</i> - Programsko gašenje procesa
SIGKILL	9	<i>Kill</i> - Trenutačno gašenje procesa
SIGUSR1	30,10,16	<i>User defined 1</i>

Popis svih signala: `man 7 signal`

# Slanje signala

- ▶ Signali se u komandnoj liniji šalju naredbom `kill`  
`kill [-<kod signala>] <PID>`
  - Ako kod signala nije naveden kao argument `kill` će poslati signal `SIGTERM`
- ▶ Neke kombinacije tipki također mogu poslati signale (npr. `CTRL+C`)

# Child procesi

- ▶ Proces može pokrenuti drugi proces. Prvi proces je tada *roditelj*, a svi procesi koje je pozvao su njegova *djeca*.
- ▶ Child procesi sadrže dodatni podatak - PPID – *parent PID* koji pokazuje na njihove roditelje
- ▶ Svi procesi u sustavu su djeca nekog drugog procesa
  - Iznimka je *init* proces s  $PID = 1$  kojeg stvara kernel pokretanju OS-a
  - Proces *init* je roditelj svim ostalim procesima
- ▶ Gašenje roditeljskog procesa gasi i sve njegove child procese.

# Child procesi

## Process management funkcije

- ▶ Pokretanje novog procesa ostvaruje se pozivanjem funkcije `fork()`
- ▶ `fork()` će stvoriti novi proces iz roditelja pozivajući *isti* program
- ▶ Prilikom izvršavanja programa u novom procesu ponovno se nailazi na funkciju `fork()` koja vraća vrijednost:
  - *PID novog procesa*, ako je `fork()` pozvan iz roditeljskog procesa
  - *0*, ako je `fork()` pozvan iz child procesa

# fork() primjer

## main.c

```
#include <unistd.h>
#include <stdio.h>

int main(){
    int pid = fork();

    if (pid == 0)
        printf("Ovo je child proces!\n");
    else if (pid != -1)
        printf("Ovo je roditelj. Pozvan je child proces s PID %d\n", pid);
    else {
        printf("Fork nije uspio.\n");
        return 1;
    }

    return 0;
}
```

## Izlaz

Ovo je roditelj. Pozvan je child proces s PID 4924.

Ovo je child proces!

# Child procesi

## Process management funkcije

- ▶ Funkcija `wait()` čeka da child proces pozvan funkcijom `fork()` završi s izvođenjem.
  - `wait()` će, ako nije drugačije zadano čekati završetak prvog pozvanog djeteta
  - `wait()` u jeziku C može prihvatiti izlaznu vrijednost programa:  

```
int status; wait(&status);
```
- ▶ Nakon završetka izvođenja child procesa, on će ostati u stanju *zombie* dok ga roditeljski proces ne "uhvati" `wait()` pozivom
- ▶ `waitpid()` će čekati završetak procesa čiji PID zadajemo argumentom
- ▶ Zadatak:
  - Proučiti sva man poglavlja za `fork`, `wait` i `waitpid`.



## wait() primjer

Dogradimo prethodni primjer:

main.c

```
...
    if (pid == 0)
        printf("Ovo je child proces!\n");
    else if (pid != -1){
        printf("Ovo je roditelj. Pozvan je child proces s PID %d\n", pid);
        wait();
        printf("Child proces je završio s izvođenjem\n");
    } else {
        printf("Fork nije uspio.\n");
        return 1;
    }
...
```

Izlaz

Ovo je roditelj. Pozvan je child proces s PID 4924

Ovo je child proces!

Child proces je završio s izvođenjem

# Dodatne naredbe

## Pretraga i ispis procesa

- ▶ `pgrep` pretražuje procese po imenu i drugim atributima
- ▶ `ps tree` ispisuje stablo svih procesa na sustavu
- ▶ Naredba `top` prikazuje stanje svih procesa u real-time sučelju
  - Naprednija verzija ovog sučelja - `htop`

## Podrazumijevane vrijednosti naredbe `top`

- ▶ Osvježavanje se obavlja svake 3 sekunde
  - Učestalost osvježavanja se mijenja tipkom `i`
- ▶ Procesi su poredani po zauzeću procesora
  - Tipka `M` - sortiranje po zauzeću memorije
  - Tipka `S` - sortiranje po zauzeću procesora

# Dodatne naredbe

## Slanje signala

Za slanje signala procesima po nazivu procesa postoji nekoliko naredbi

- ▶ `killall` - šalje signal procesima pod točno navedenim nazivom
  - Signal se šalje svim procesima pod tim imenom
  - Primjer: `killall -SIGKILL dd`  
šalje SIGKILL svim dd procesima
- ▶ `pkill` - šalje signal procesima pretragom naziva po zadanom uzorku
  - Primjer: `pkill -SIGKILL dd`  
šalje SIGKILL svim procesima koji u nazivu sadrže dd (npr. proces `ddclient` bi ovdje također bio prekinut)

# Prioriteti

- ▶ Kod rezerviranja CPU vremena procesi imaju prioritet - *niceness*
  - -20 - najviši prioritet
  - +19 - najniži prioritet
  - Većina korisničkih programa ima isti niceness - 0
  - Procesi *realtime* prioriteta imaju prednost nad ostalima bez obzira na niceness
- ▶ Naredbom *nice* procesi se pokreću sa višim ili nižim prioritetom
  - *renice* mijenja prioritet postojećeg procesa ili grupe procesa

# Ljuska

- ▶ Ljuska je korisničko sučelje prema operacijskom sustavu
- ▶ Program koji se prvi pokreće paljenjem terminala
  - Ljuska tada ima ulaz i izlaz vezan na terminal
- ▶ Ljuska pokreće procese i usmjerava njihove ulaze i izlaze
- ▶ Ljuska može omogućiti automatizaciju (skripte)

# Ljuska

- ▶ Ljuske u Linuxu
  - Popis ljuski dostupnih na sustavu - `/etc/shells`
  - `sh`, `bash`, `csh`, `ksh`, ...
- ▶ Zadana ljuska za korisnika se zadaje u `/etc/passwd`
- ▶ Neke (starije) distribucije podržavaju promjenu ljuske s `chsh`

# Ljuska

## Bash

- ▶ Najraširenija ljuska na Linux sustavima - `bash`
  - *Bourne again shell*
- ▶ Nadogradnja `sh` ljuske (*Bourne shell*)
- ▶ Bash uvodi
  - upravljanje poslovima (`jobs`, `bg`, `fg`)
  - neograničenu povijest naredbi
  - pseudonime (*alias*)

# Bash

## Načini rada

- ▶ Bash ljusku je moguće pokrenuti na različite načine. Najbitniji od njih su:
  - interaktivno bez prijave
  - interaktivno sa prijavom
  - neinteraktivno
- ▶ Ovisno o odabranom načinu pokretanja, bash će na početku procesuirati različite skripte, a u radu će se drugačije ponašati



# Bash

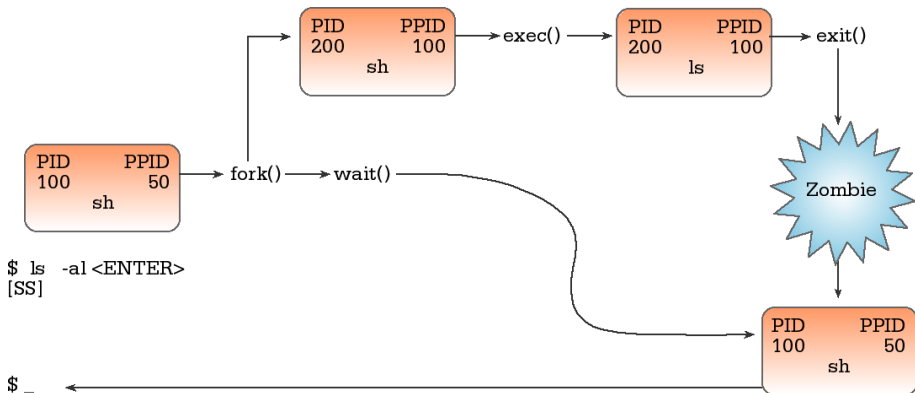
## Interaktivni način rada

- ▶ Korisnik može unositi naredbe
- ▶ Ako se korisnik pri pokretanju ljuske prijavio onda je pokrenuta *interactive login shell*. Ljuska izvršava datoteke
  - `/etc/profile`
  - Prvu koja postoji od: `~/.bash_profile`, `~/.bash_login`, `~/.profile`
- ▶ Ako se korisnik pri pokretanju ljuske nije prijavio onda je pokrenuta *interactive non-login shell*. Ljuska izvršava datoteke
  - `~/.bashrc`
  - `/etc/bash.bashrc`
- ▶ Uobičajeno je skriptu `bashrc` pozvati u `bash_profile` kako bi se ista skripta izvodila pri svakoj vrsti interaktivne prijave

# Bash

## Procesi u Bashu

### ► Pokretanje procesa u sh ljuskama



# Interpretori skripti

- ▶ U skriptama se interpreter odabire *shebang* sekvencom znakova `#!` u prvoj liniji koda

Sintaksa

```
#! path [argumenti]
```
- ▶ Shebang sekvenca se može koristiti za odabir interpretera bilo kojeg skriptnog programskog jezika
- ▶ U shell jezicima se obično navodi kako bi se skripta uvijek pokretala u ispravnoj ljusci
- ▶ Primjer: U skripti pisanoj za bash (ali ne i za, primjerice, sh) pišemo

```
#! /bin/bash
```

# Posao

- ▶ Pokretanjem procesa Ljuska stvara *posao* (job)
- ▶ Posao procesu pridjeljuje attribute kao što su
  - Terminal kojem proces (posao) pripada
  - Ulazne i izlazne uređaje (stdin, stdout, stderr)
  - ...
- ▶ Posao može objediniti više procesa koji su međusobno ovisni  
Primjer: Procesi vezani pipeom objedinjeni su u jedan posao  
`cat /etc/passwd | grep korisnik`
- ▶ Posao ima vlastiti identifikator - *Job ID* (JID)
  - Dodjeljuje ga Ljuska
  - U svakoj pokrenutoj Ljusci brojanje JID-ova počinje od 1 (u dvije Ljuske može postojati više istih JID-ova)
- ▶ Terminiranjem Ljuske terminiraju se svi poslovi u njoj

# Kontrola poslova

- ▶ Posao pokrenut u ljusci može biti u dva načina rada:
  - *Foreground* - prednji plan (fokus) u ljuski
  - *Background* - rad u pozadini
- ▶ Kod pokretanja procesa iz terminala ljuska postavlja posao u foreground
- ▶ Za pokretanje procesa u pozadini koristi se operator &  
Sintaksa: `<naredba> &`
  - Ljuska ispisuje JID i PID procesa

Primjer: `vim &`

`[1] 4132`

# Kontrola poslova

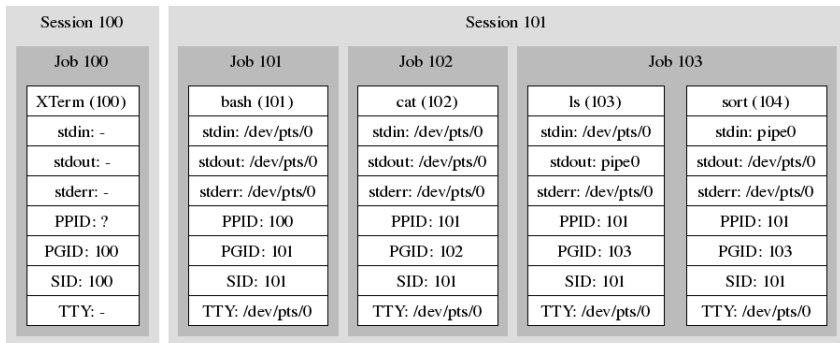
- ▶ Proces koji je trenutno u *foreground*-u može se *suspendirati* (signal SIGTSTP) kombinacijom tipki *CTRL + Z*
- ▶ Ljuska ispisuje JID suspendiranog procesa
- ▶ Proces se može nastaviti naredbama:
  - *fg* - u foregroundu
  - *bg* - u backgroundu
- ▶ Popis poslova koji se izvode u trenutnoj ljusci dobije se naredbom *jobs*

# Procesi u Linuxu

- ▶ Koncept poslova u ljusci na kernelskoj je razini izveden *grupama procesa*
  - Grupe procesa su koncept podržan od samog operativnog sustava
  - Grupe procesa omogućuju istovremeno slanje signala svim međusobno ovisnim procesima (unutar iste grupe)
  - U modernim ljuskama ova dva pojma se praktički poistovjećuju
- ▶ Grupe procesa imaju identifikator PGID (*Process group ID*)
  - PGID je jednak PID-u prvog pokrenutog procesa u grupi
- ▶ Osiromašeni (*orphaned*) proces
  - Proces koji je ostao pokrenut u grupi procesa nakon što je glavni proces terminiran

# Procesi u Linuxu

- ▶ Više grupa procesa dio su jedne *sjednice* (engl. *session*)
  - Jedna sjednica obuhvaća sve procese vezane za isti terminal
  - Prijavom korisnika na sustav započinje se *login session*





# Literatura

- ▶ <http://www.linux.com/archive/feature/125977>
- ▶ <http://www.linux-tutorial.info/modules.php?name=MContent&pageid=84>
- ▶ <http://www.linuxhq.com/guides/SAG/x1826.html>
- ▶ <http://www.win.tue.nl/~aeb/linux/lk/lk-10.html>
- ▶ <http://www.linusakesson.net/programming/tty/index.php>
- ▶ <http://acms.ucsd.edu/info/jobctrl.html>

man stranice: ps, 7 signal, fork (sva poglavlja), wait (sva poglavlja), jobs