

Labo Besturingssystemen 2

Gorik De Samblanx — Ann Philips — Jeroen Van Aken

1 Multi-User operating system: LINUX

Leerdoelen

- De basis structuur van het LINUX operating system begrijpen
- bestanden en folders kunnen zoeken en manipuleren via de Linux terminal
- programma's of taken kunnen starten, stoppen en opvolgen via de terminal

Linux heeft zijn wortels in het Unix operating System. Linux is eenvoudig gezegd een implementatie van Unix voor desktop PC's. Unix is een multi-user multi-tasking besturingssysteem dat in eerste instantie bedoeld was voor mainframes, werkstations en servers. Hoewel ontworpen en geïmplementeerd in de jaren '70, wordt het nog steeds gebruikt voor bepaalde toepassingen. Gebaseerd op de statistieken op Wikipedia ¹, beheerst Windows OS de desktop/laptop markt, maar het grootste deel van het internet draait op Unix/Linux en bijna elke supercomputer draait Linux. De gebruikersbasis is zelfs groter dan ooit en groeit nog steeds omdat Unix/Linux vaak het OS is van embedded apparaten, smartphones en tablets (Android draait op een Linux kernel!) en zelfs je auto.

Het Linux OS bestaat uit verschillende componenten, waaronder:

- Een bootloader: beheert het opstarten (booten) van het OS;
- De Linux kernel;
- Apparaatstuurprogramma's: software om de hardwareapparaten te besturen en te interfacen; - Daemons: diensten (printen, internet, ...) die op de achtergrond draaien;
- De shell: een op tekst gebaseerde interface waarmee je instructies kunt geven aan de kernel, programma's kunt uitvoeren, enz;
- Een window manager voor Xorg: een systeem dat alle grafische elementen (vensters, knoppen, schuifbalken, enz.) afhandelt die op het scherm worden weergegeven; Er zijn verschillende populaire window managers in gebruik: IceWM, xfwm, KWin, openbox, ...;
- Een desktopomgeving: een GUI-gebaseerde interface voor controle en interactie met het systeem, tegenwoordig populairder dan de shell op desktop/laptop-pc's. Een desktopomgeving definieert de look-and-feel van het bureaublad, maar heeft een windowmanager nodig voor het tekenen van alle grafische onderdelen.
- Applicaties

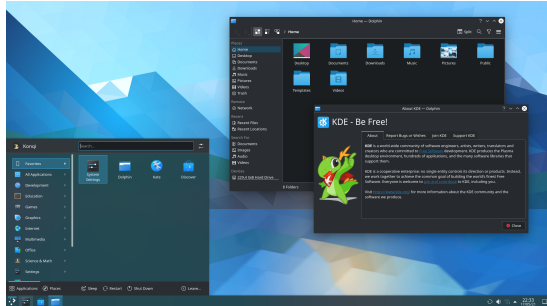
Een van de dingen waar Linux-starters mee worstelen is dat er niet zoiets bestaat als één Linux-systeem: er bestaan meer dan honderd Linux-distributies (kortweg distro's). Distro's verschillen voornamelijk in de desktopomgeving, de voorgeïnstalleerde applicaties en de softwarerepository (cf. app store) die ze aanbieden. Er bestaan verschillende populaire bureaubladomgevingen, waaronder :

- KDE: een bureaubladomgeving die zich richt op een intuïtieve, gebruiksvriendelijke en zeer aanpasbare werkplek; meer informatie is te vinden op <https://www.kde.org> (figuur 1a)
- Gnome: richt zich sinds Gnome 2 op gebruiksvriendelijkheid, cognitieve ergonomie en toegankelijkheid;

¹https://en.wikipedia.org/wiki/Usage_share_of_operating_systems

geschreven in C / C++ / Python / Vala / Javascript; meer informatie is te vinden op <https://www.gnome.org> (figuur 1b)

- XFCE: heeft als doel om snel en lichtgewicht te zijn en toch visueel aantrekkelijk en gemakkelijk te gebruiken; geschreven in C en gebaseerd op GTK+ toolkit; meer informatie is te vinden op <https://www.xfce.org> (figuur 1d)
- LXDE: een zeer lichtgewicht X11 desktop met een lage (laagste?) geheugenvoetafdruk en energieverbruik, meestal gebruikt op systemen met beperkte bronnen (bijv. Raspberry Pi) of oudere hardware; geschreven in C/C++ en gebaseerd op GTK+ toolkit; meer informatie is te vinden op <https://www.lxde.org>



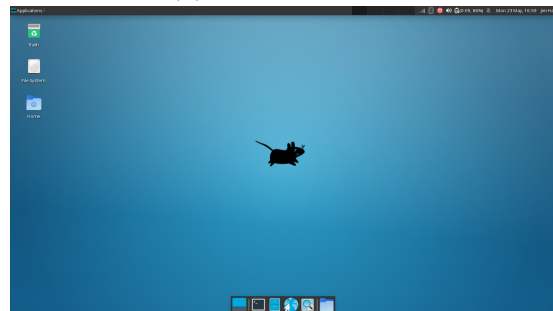
(a) KDE desktop



(b) Gnome desktop



(c) Unity desktop



(d) XFCE desktop

Figuur 1: Linux Desktop environments

Linux distro's zijn er in vele smaken, sommige ontworpen voor algemeen gebruik, andere voor specifieke taken. Hieronder worden slechts enkele distro's besproken om je wat voorbeelden en referenties te geven. Een eenvoudige zoektocht op het internet (bijv. <https://distrowatch.com/>) onthult vele andere en er bestaan zelfs documenten met richtlijnen om je te helpen de juiste distro te kiezen.



Red Hat Linux : Een van de oorspronkelijke Linux-distributies. De commerciële, niet-vrije versie is Red Hat Enterprise Linux, die gericht is op grote bedrijven die Linux-servers en -desktops op grote schaal gebruiken. De gratis variant voor desktop systemen van deze distributie heet: Fedora Linux.



SuSE Linux : Deze distributie is voornamelijk beschikbaar tegen betaling omdat het veel commerciële programma's bevat, hoewel er ook een uitgekilde gratis versie is die je kunt downloaden.



Ubuntu Linux : gebaseerd op Debian, ontworpen voor regelmatige releases, consistente gebruikerservaring en commerciële ondersteuning op zowel desktop als server. Er zijn veel meer officiële distributies van op Ubuntu gebaseerde distro's: Kubuntu (Ubuntu met een plasma-desktop), Lubuntu (Ubuntu met een LXDE-desktop), Xubuntu (Ubuntu met een XFCE-desktop), ...



Linux Mint : gebaseerd op Debian en Ubuntu, met een Cinnamon of MATE desktop, met een mix van open source en propriëtaire softwaretoepassingen;



Gentoo Linux : is ontworpen als een zeer draagbaar, flexibel en modulier systeem dat wordt gedistribueerd als open-source code die zelf moet worden gecompileerd en geoptimaliseerd op het lokale systeem; dit zou moeten resulteren in machinespecifieke optimalisaties en snelheidsverbeteringen;



Arch Linux : volgt het KISS-principe (keep it simple, stupid) en richt zich op eenvoud, minimalisme en correctheid van de code; het volgt een 'rolling release model' wat betekent dat alleen regelmatige systeemupdates nodig zijn om altijd de laatste release te draaien; Arch verwacht dat gebruikers een zeker begrip hebben van de werking van het systeem;



Kodi : een Linux distro voor mediaspelers en thuisbioscoopssystemen;



OpenWrt : Linux versie voor routers;



Kali : een niche Linux gericht op computerbeveiliging, digitaal forensisch onderzoek en penetratietesten;

1.1 Linux terminal

Zoals vermeld in de inleiding is Unix/Linux een multi-user systeem. Vroeger (voor de revolutie van de desktop PC) was computerkracht schaars en duur. Daarom werden krachtige werkstations en mainframe computers gebouwd die door veel gebruikers gedeeld konden worden. Deze gebruikers maakten verbinding met de Unix computer met een terminal (toetsenbord, scherm, seriële interface). De terminal werd alleen gebruikt voor het verkrijgen van invoer, het weergeven van uitvoer en fouten (stdin, stdout, stderr!) en het echte rekenwerk werd gedaan op de Unix computer.

Tegenwoordig draait Linux zelfs op de meest bescheiden desktop computer en gebruiken we geen echte terminals meer om toegang te krijgen. Toch is het idee van een terminal nog steeds geldig, omdat je nu een geëmuleerde of virtuele terminal opent om toegang te krijgen tot de commandoregel. En je kunt er veel tegelijk openen. Virtuele terminals worden vaak aangeduid met pts/0, pts/1, etc. waarbij pts staat voor pseudo terminal. Pseudo terminals worden typisch gebruikt door grafische terminal implementaties zoals xterm en bij inloggen op afstand, zoals telnet of ssh. Native terminals, seriële device terminals, hardware of kernel geëmuleerde terminals worden daarentegen typisch aangeduid met tty/0,



`tty/1`, etc. waarbij `tty` staat voor ‘tele-typewriter’.

1.2 Bestanden structuur

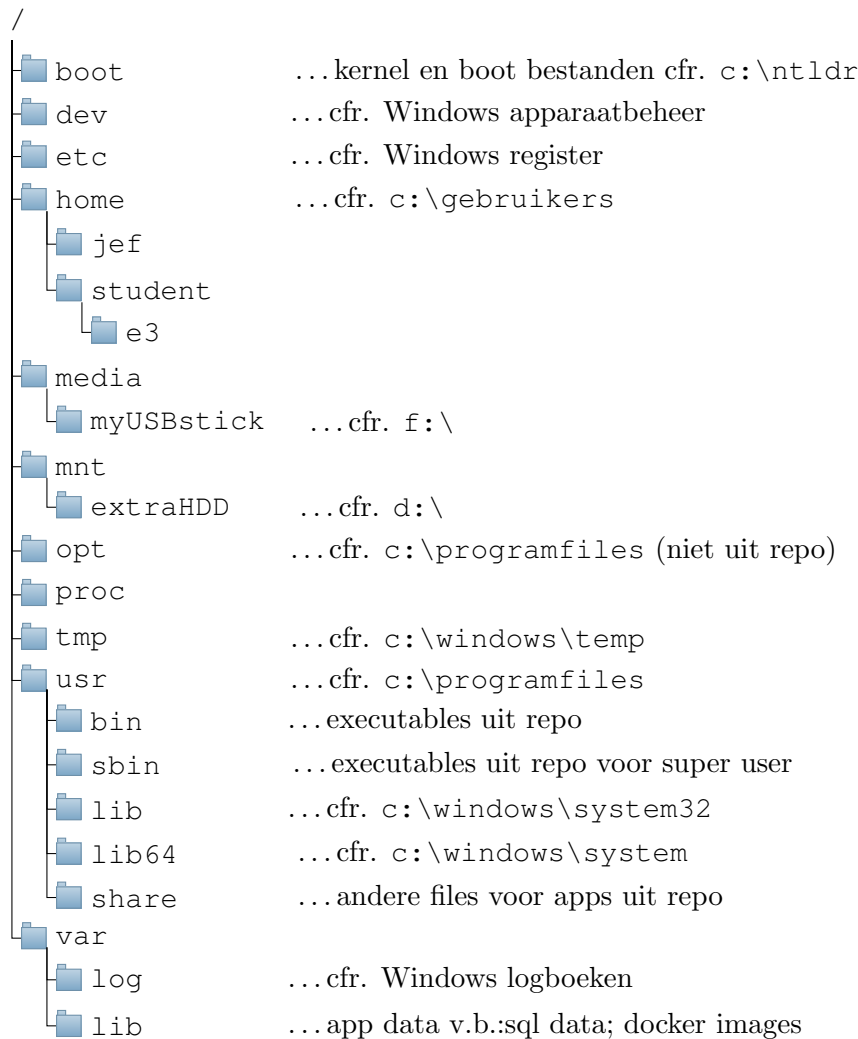
Er is een gezegde over Unix/Linux dat als volgt gaat: “In Unix/Linux is alles een bestand; als iets geen bestand is, is het een proces.” Wat een proces precies is, wordt later besproken. We richten ons eerst op bestanden.

Een bestand is een kernonderdeel van het Linux systeem. Bestanden kunnen leesbare ASCII-tekst of binaire gegevens bevatten die documenten, afbeeldingen, ... voorstellen, maar ook het toetsenbord en de muis, printers, enz. worden voorgesteld als een soort bestand. Bestanden kunnen worden opgeslagen op elk medium zoals de harde schijf, SSD, SD, CD-ROM, ... Linux organiseert alle bestanden - waar ze ook zijn opgeslagen - in een boomstructuur: het bestandssysteem. Afbeelding 2 hieronder illustreert dit idee.

Alles begint bij de wortel (‘root’) van de boom, aangeduid als `/`. Bestanden kunnen worden georganiseerd in ‘directories’, wat eigenlijk ‘speciale’ bestanden zijn die lijsten en informatie van andere bestanden en directories bevatten. Het maakt niet uit waar bestanden fysiek zijn opgeslagen - op een harde schijf, SD of zelfs op het netwerk - ze worden allemaal weergegeven in mappen van de root-bestandsstructuur.

Er bestaan verschillende soorten bestanden. Gewone bestanden bevatten normale gegevens, bijvoorbeeld tekst, SQL-gegevens of uitvoerbare binaire code. Directories zijn een ander type bestanden dat al eerder is uitgelegd. Naast reguliere bestanden en mappen zijn er een aantal speciale bestanden. Bijvoorbeeld, speciale bestanden in `/dev` die gebruikt worden voor invoer en uitvoer, en realtime statusinformatie over de kernel, processen, I/O, etc. sockets zijn ook speciale bestanden die gebruikt worden voor inter-proces netwerkmachtige communicatie met bescherming van het bestandssysteem voor toegangscontrole. Named pipes bieden een andere manier voor inter-proces communicatie zonder de netwerk-stijl van communicatie. Meer details over deze speciale bestanden volgen later.

<code>/bin</code>	bevat de binaire bestanden van algemene programma's, gedeeld door het systeem, de systeem-beheerder en gebruikers; (Op een Windows systeem komt deze map min of meer overeen met de map <code>c:\windows</code> .)
<code>/boot</code>	bevat de opstartbestanden die nodig zijn voor het opstartproces, de kernel (<code>vmlinuz</code>), ...; (Op een Windows systeem komt deze map min of meer overeen met de map <code>c:\boot</code> .)
<code>/dev</code>	bevat verwijzingen naar alle CPU perifere hardware, harde schijf partities, etc., die worden weergegeven als bestanden met speciale eigenschappen. Let op de links naar <code>stdin</code> , <code>stdout</code> en <code>stderr</code> . Een andere curiositeit is ‘null’, ‘zero’ en ‘random’. Schrijven naar <code>/dev/zero</code> is hetzelfde als schrijven naar <code>/dev/null</code> : alle gegevens gaan verloren. Lezen van <code>/dev/zero</code> genereert evenveel null-chars als er gelezen worden. <code>/dev/random</code> genereert willekeurige data, dit kan zowel afdrukbaar als niet-afdrukbaar zijn. Probeer <code>head /dev/random</code> om het te controleren.
<code>/etc</code>	bevat de belangrijkste systeemconfiguratiebestanden (d.w.z. gegevens vergelijkbaar met die in het Configuratiescherm in Windows); bijvoorbeeld <code>/etc/passwd</code> bevat alle gebruikersinformatie, <code>/etc/group</code> bevat alle groepsinformatie en <code>/etc/shells</code> bevat de namen van beschikbare shells; gebruik <code>cat</code> om het te controleren; (Op een Windows systeem wordt dit soort informatie min of meer opgeslagen in het Windows register).
<code>/home</code>	bevat de thuismappen van de gewone gebruikers; elke thuismap is alleen toegankelijk voor de corresponderende gebruiker en de systeembeheerder; (Op een Windows systeem komt deze map min of meer overeen met de map <code>c:\users</code>).
<code>/lib</code>	bevat kernel modules en gedeelde bibliotheekbestanden voor allerlei programma's die nodig zijn voor het systeem en de gebruikers (in <code>/bin</code> en <code>/sbin</code>); (Op een Windows systeem komt deze map min of meer overeen met de map <code>c:\windows\system</code> met <code>dll</code> bestanden).
<code>/mnt</code>	standaard koppelpunt voor externe bestandssystemen, bijvoorbeeld een CD-ROM of een digitale camera; (op een Windows systeem komt deze map ongeveer overeen met <code>c:\</code> , <code>d:\</code> , <code>f:\</code> , enz.)



Figuur 2: Linux directory structuur

/opt	een map gereserveerd voor uitbreidingspakketten en software van derden, die geen deel uitmaakt van de standaard installatie of pakketbeheerder; (Op een Windows systeem komt deze map ongeveer overeen met de map c:\programfiles).
/proc	is een virtueel bestandssysteem dat door de kernel wordt gebruikt om informatie over systeembronnen (kernel datastructuren) te sturen naar processen die informatie bevatten; typ bijvoorbeeld <code>cat /proc/loadavg</code> om het huidige belastingsgemiddelde op je Linux systeem te zien;
/root	de thuishap van de root (systeembeheerder); let op het verschil tussen <code>/</code> , de hoofdmap en <code>/root</code> , de thuishap van de root gebruiker; (Op een Windows systeem komt deze map ongeveer overeen met de map c:\users\administrator).
/run	een tijdelijk bestandssysteem dat gebruikt wordt tijdens het opstarten;
/sbin	bevat programma's voor gebruik door het systeem en de systeembeheerder; (Op een Windows systeem komt deze map ongeveer overeen met een map als c:\windows\system32.)
/sys	is een virtueel bestandssysteem dat het beeld van de kernel van het systeem weergeeft;
/tmp	een directory gereserveerd voor tijdelijke ruimte voor gebruik door het systeem en applicaties; wordt vaak schoongemaakt bij het opnieuw opstarten, dus gebruik deze niet om werk op te slaan; (Op een Windows systeem komt deze directory min of meer overeen met de map c:\windows\temp.)
/var	bevat variabele bestanden zoals logbestanden, de mail wachtrij, de print spooler ruimte, ruimte voor tijdelijke opslag van bestanden gedownload van het internet,... die blijven bestaan tussen twee opstart sequenties; mind <code>/var/log</code> bevat onder andere de Linux systeem logs; bijvoorbeeld,

/var/cache/apt/archives/ bevat installatie pakketten geïnstalleerd door apt-get.

Standaard start je in je thuismap wanneer je een nieuwe terminalsessie opent. Linux is een multi-user systeem. Linux maakt voor elke gebruiker een eigen thuismap aan. In je thuismap heb je alle rechten om bestanden en mappen aan te maken, te hernoemen en te verwijderen. Buiten je thuismap heb je deze rechten misschien niet.

1.2.1 Bestandsnavigatie

De basiscommando's om door de boomstructuur van het bestandssysteem te navigeren en deze te bewerken/wijzigen, staan hieronder.

pwd drukt het pad van de actuele directory af;

ls toont de inhoud van een directory;

`ls -l` : zoals `ls` maar toont meer gedetailleerde informatie, zoals te zien is in het voorbeeld hieronder. De `-l` is een commandoregeloptie of -argument. De meeste commando's hebben veel mogelijke opties die meestal worden uitgelegd in de 'info' of 'man' pagina's van het commando. Daarover binnenkort meer.

```
$ ls -l
total 79284
drwxrwxr-x  3 jef jef      4096 jan 12 11:45 Documents
drwxrwxr-x 16 jef jef     36864 jan  9 10:45 Downloads
-rw-rw-r--  1 jef jef      1195 jan  9 08:40 demo.txt
```

Merk op dat in de eerste kolom informatie als `drwxrwxr-x` wordt afgedrukt. Het eerste teken geeft het bestandstype aan (d voor directory, - voor bestand, l voor link, ...).

cd <dir> de actuele directory wijzigen naar <dir>.

cd .. werkdirectory wijzigen in de bovenliggende directory van de huidige directory (in de boomstructuur van het bestandssysteem);

Opmerking: <dir> moet de locatie van de map in de boomstructuur van het bestandssysteem aangeven. Deze locatie kan worden opgegeven als een 'relatief' of een 'absoluut' pad. Bijvoorbeeld, /home/jef/Documenten is een absoluut pad: het geeft precies de locatie van de map aan, onafhankelijk van de huidige locatie. Het pad ../Documenten daarentegen is een relatief pad: de doelmap wordt gevonden door eerst één niveau omhoog te gaan in de boomstructuur van het bestandssysteem (naar het bovenliggende niveau) en dan de map 'Documenten' te selecteren. Natuurlijk is de uiteindelijke bestemming afhankelijk van de huidige map, dus relatief ten opzichte van de huidige map. Merk op dat '.' een steno is om de bovenliggende map aan te geven en '..' een steno om de huidige map aan te geven, bijvoorbeeld ./main.c geeft het bestand main.c in de huidige werkdirectory aan.

cd ~ Wijzig de werkmap naar de thuismap van de gebruiker, ongeacht de huidige locatie;

mkdir <dir>
een nieuwe map <dir> aanmaken in de huidige map;

cp <src-file> <dest-file>
kopieert een bestand van een bron (kan het pad naar het bestand bevatten) naar een doelbestand;

Opmerking 1: let op het gebruik van een mapnaam met een / achteraan, het resultaat van `cp lab1 lab2` is bijvoorbeeld anders dan `cp lab1 lab2/`. Probeer dit uit!

Opmerking 2: de opdrachtregeloctie `-R` wordt gebruikt om een map 'recursief' te kopiëren, wat betekent dat alle inhoud - bestanden en submappen - wordt gekopieerd.

mv <src-file> <dest-file>
een bestand van een bron naar een bestemming verplaatsen;

rm <file> verwijdt een bestand uit een directory;

Opmerking en waarschuwing: ook hier kan de opdrachtregeloctie '-R' gebruikt worden om 'recursief' bestanden en mappen te verwijderen, maar wees voorzichtig met deze optie omdat het veel meer kan verwijderen dan je van plan was en er is geen manier om verwijderde bestanden te 'recyclen'!

rmdir <dir>

verwijdert een directory, wanneer deze geen files of subdirectories meer bevat;

tree

geeft de inhoud van mappen weer in een boomstructuur. Wees voorzichtig met dit commando omdat het erg lange lijsten kan genereren, bijvoorbeeld als de huidige directory de root dir is.

Oefening

Ga naar je 'home' folder en maak voor dit vak een subdirectory met folder voor elke opgave in dit labo. Als output van het `tree` commando kom je dan op onderstaande weergave.

```
bsys2/
|-- opgave1
|-- opgave2
'-- opgave3

3 directories, 0 files
```

1.2.2 Programmalocatie

Er zijn een aantal commando's die helpen bij het vinden van de locatie van commando's en programma's:

locate bestanden op naam vinden door in een indexdatabase te zoeken (deze database bevat mogelijk niet de meest bijgewerkte informatie);

which een bestand vinden, maar kijkt alleen in het zoekpad van de gebruiker; het zoekpad van de gebruiker wordt opgeslagen in de omgevingsvariabele `PATH` en kan worden afgedrukt met `echo $PATH`.

Opmerking: Het is de moeite waard om hier het `echo` commando te introduceren. Het `echo` commando lijkt niet veel waarde te hebben omdat het alleen de tekst weergeeft die volgt op het commando, bijvoorbeeld `echo hello world!` stuurt de tekst 'hello world!' naar 'stdout'. Als je de man page leest, zul je ontdekken dat je enkele opmaakopties kunt toevoegen, maar dat is eigenlijk alles. Desondanks is het `echo` commando een veelgebruikt Linux commando. Het `echo` commando wordt meestal gebruikt in shell scripts - die buiten het bereik van deze cursus vallen - om informele berichten aan de gebruiker te tonen tijdens het uitvoeren van het script.

Je kunt `echo` hier ook gebruiken om de waarden van zogenaamde shell- en omgevingsvariabelen af te drukken. Je kunt shellvariabelen heel eenvoudig als volgt definiëren en instellen: `<variabele>=<waarde>`. Bijvoorbeeld: `mijnTekst='hello world!'`. Let hierbij dat er geen spatie staat voor of na het = teken. Gebruik `$` om de waarde van een shellvariabele te krijgen, bijvoorbeeld `$mijnTekst`. Het uitprinten van de waarden met `echo` is eenvoudig, bijv: `echo $mijnTekst`

whereis zoek de binaire, bron- en manpagina's voor een commando in een lijst met standaard Linux-plaatsen.

1.2.3 Bestandsinformatie

Verder zijn er ook een aantal commando's die je meer info kunnen geven over een bestand of de indeling en gebruik van het bestandssysteem.

stat geeft gedetailleerde informatie over een bestand (grootte, I-node, links, tijdstempels van wijzigingen, ...);

du	schijfgebruik schatten (wat meestal anders is dan de bestandsgrootte) van bestanden of mappen; gebruik <code>du -h</code> om door mensen leesbare grootte in KB, MB, enzovoort te produceren;
df	rapporteert het gebruik van beschikbare schijfruimte; gebruik <code>df -h</code> om door mensen leesbare grootte in KB, MB, enzovoort te produceren;
fdisk	de partitietabel van een schijf oplijsten of wijzigen (wees voorzichtig!);
lsblk	geeft informatie over alle blokapparaten; gebruik <code>lsblk -f</code> om meer gedetailleerde informatie over alle bestandssystemen te krijgen;

Oefening

Hoeveel ‘fysieke’ schijven zijn er binnen je Linux systeem? Check ook hoeveel schijfruimte er nog vrij is op het filesysteem waar je ‘home’ folder op te vinden is.

tip: gebruik de `-h` optie om getallen in een ‘leesbaar’ formaat weer te geven

1.3 Man pages

Soms weet je de exacte naam van een commando niet meer of ben je de betekenis van een opdrachtregeloctie vergeten. Een aantal informatiehulpmiddelen kan je hierbij helpen.

apropos <keyword>

geeft een lijst met commando's die <keyword> bevatten;

man <keyword>

toont de man(ual) pagina's van <keyword>;

Opmerking 1: je kunt hulp krijgen bij man zelf door `man man` in te typen.








Opmerking 2: de man pagina's zijn verdeeld in 9 secties zoals systeemoproepen, bibliotheekoproepen, shell commando's, enz. Zie `man man` voor meer informatie. Een sleutelwoord kan in meer dan één sectie voorkomen, bijvoorbeeld het sleutelwoord `write`. In dat geval moet je het sectienummer vermelden om de juiste man pagina te selecteren, bijvoorbeeld `man 1 write` of `man 2 write`.

Opmerking 3: je kunt ook vooruit en achteruit zoeken binnen een man pagina met respectievelijk `/<pattern>` en `?<pattern>`. Gebruik de helpfunctie ‘h’ in een manpagina voor meer informatie.

1.4 Bash shell functies

Wanneer je een terminal sessie opent, wordt er een shell gestart waarmee je kunt communiceren met het Linux systeem door commando's in te voeren of scripts en programma's uit te voeren. In tegenstelling tot GUI-gestuurde (grafische gebruikersinterface) systemen zoals Windows, Gnome of KDE, is een shell een op CLI (command line interface) gebaseerde interactieoplossing. Er bestaan verschillende soorten shells in Linux: 'sh' (Bourne shell), 'ksh' (Korn shell), 'csh' (C shell), 'Bash' (Bourne again shell - combineert Korn en C shell functies en is vaak de standaard shell in Linux), ...

Als de standaard toegewezen shell je niet bevalt, gebruik dan het `chsh` (change shell) commando om over te schakelen naar een andere shell. We gaan ervan uit dat je Bash gebruikt in deze cursus. Een aantal toetscombinaties zijn de moeite waard om te onthouden om je efficiëntie te verhogen.

	commando/bestandsnaam voltooien;
 	toon alle mogelijkheden om commando's en bestandsnamen aan te vullen;
 + 	een lopend programma beëindigen;
 + 	maak de terminal leeg;

- history** commando dat de opdrachtgeschiedenis toont;
- ! <num>** het commando met nummer `num` in de commandogeschiedenis uitvoeren;
- !!** de eerste opdracht in de opdrachtgeschiedenis uitvoeren;
- [ctrl]+[r]** doorzoek de opdrachtgeschiedenis (herhaal **[ctrl]+[r]** om door de geschiedenis te bladeren);
- [↑] of [↓]** alternatief voor **[ctrl]+[r]** en waarschijnlijk veel gemakkelijker;

1.4.1 Jokertekens

Stel je voor dat je alleen alle bestanden wilt weergeven die eindigen op `.c`. Dit kun je doen met `ls` met jokertekens, dus `ls -l *.c`. Jokertekens kunnen worden gebruikt in strings voor het matchen van patronen. Patroonherkenning wordt meestal gebruikt in lijsten, het zoeken en manipuleren van bestanden, het filteren van gegevens, enz. Daarom kunnen veel commando's gebruik maken van wildcards. Om een ander voorbeeld te geven: `rm *.o` verwijdert alle objectcode (`.o`) bestanden. Een tekenreeks patroon met jokertekens wordt uitgebreid tot een lijst van tekenreeksen die overeenkomen met het patroon. Enkele voorbeelden:

- ?** komt overeen met elk afzonderlijk teken, bijvoorbeeld `pts/3` voldoet aan het patroon `pts/?`;
- *** komt overeen met een willekeurig aantal tekens (nul of meer), bijvoorbeeld `'ma*.c'` komt overeen met alle `c`-bestanden die beginnen met `'ma'`;
- []** komt overeen met elk teken dat is opgegeven met **[]**, bijvoorbeeld `m[a,o,e]t` zal uitbreiden naar `'man'`, `'mot'` en `'met'`; `-` wordt gebruikt om een bereik aan te geven, bijvoorbeeld `0-9` of `a-z`;
- [!]** logisch niet van **[]** : komt overeen met elk teken dat niet voorkomt in **[]**;
- { }** somt verschillende patronen op waarvan er ten minste aan één moet worden voldaan, bijv. `{*.odt, *.doc}` komt overeen met alle `.odt`- of `.doc`-bestanden.

Merk op dat `\` wordt gebruikt als een 'escape'-karakter om speciale karakters zoals `?`, `,`, `!`, etc. te beschermen. Als je bijvoorbeeld de bestandsnaam `ma*p` wilt aangeven in een tekenreeks patroon waarbij de `*` niet geïnterpreteerd moet worden als het jokerteken `*`, dan moet je `map` gebruiken. Meer details zijn te vinden in de `man` pagina's met `man 7 glob`.

Herinner je het `echo` commando uit 1.2.2? Het `echo` commando biedt ook een interessante functie met betrekking tot jokertekens. Je kunt `echo` gebruiken om af te drukken wat er uitgevoerd zou zijn als een commando wildcards gebruikt. Bijvoorbeeld, `rm -R *` is een gevaarlijk commando; voordat je het uitvoert, zou je `echo rm -R *` moeten gebruiken om een beter idee te krijgen wat precies overeenkomt met het `*` jokerteken.

Oefening

Hoeveel gebruikers staan er beschreven in het `/etc/passwd` bestand? Gebruik hiervoor het `wc` command en raadpleeg de manpage voor het gebruik van de juiste opties.

Ga naar je home-folder en vergelijk het verschil in output tussen `ls *` en `ls .*`

Ga naar `/etc`. Gebruik `file` om het bestandstype van de volgende subset van bestanden te achterhalen. Vergeet niet om `echo` te gebruiken om de wildcarduitbreiding te inspecteren.

- Alle bestanden die eindigen op `~`
- Alle bestanden die beginnen met het teken `'x'` of `'y'`
- Alle bestanden die de tekst `-n` bevatten met `n` een getal van 1 cijfer

1.4.2 Opdracht aliases

Het is mogelijk om snelkoppelingen of synoniemen voor bestaande commando's te maken. Deze worden 'aliases' genoemd. Om een overzicht te krijgen van welke aliases zijn ingesteld in je shell, typ je `alias`. Je kunt een soortgelijke uitvoer krijgen als hieronder. Je kunt zien dat een alias `ll` is gedefinieerd als een snelkoppeling voor het commando `ls` met de opties `alF`. Dit betekent dat je `ll` kunt gebruiken als een ingebouwd commando in de shell. De shell zal `ll` uitbreiden tot `ls -alF` en deze instructie uitvoeren.

Bv. het commando: `alias cdlab='cd ~/bsys2/oefening1'` creëert een alias `cdlab` om gemakkelijk van waar dan ook in de bestandsstructuur naar je oefenmap te springen. Je kunt `unalias cdlab` gebruiken om de alias te vernietigen. Er is één nadeel: als je je login sessie afsluit, gaan je aliases verloren. Als je je aliases permanent wilt bewaren, moet je ze opslaan in een login scriptbestand, bijvoorbeeld het verborgen bestand `.bash_aliases` in je homedirectory. Dit scriptbestand wordt elke keer uitgevoerd als je inlogt op een nieuwe sessie en daardoor worden je aliases weer geactiveerd.

1.5 I/O redirection

De meeste programma's hebben invoer nodig en produceren uitvoer. Linux commando's doen hetzelfde. Standaard wordt invoer gegeven met het toetsenbord en uitvoer wordt weergegeven op het scherm. De invoerbron van een opdrachtregelprogramma heet standaard invoer (`stdin`). De standaard uitvoerbron van een opdrachtregelprogramma heet standaard uitvoer (`stdout`). Om volledig te zijn, noemen we ook de standaard foutuitvoer (`stderr`) waar standaard foutmeldingen van programma's naartoe worden geschreven. Zoals gezegd is `stdin` standaard het toetsenbord en `stdout` het scherm, maar dit kan veranderd worden indien nodig. Met I/O redirection kun je eenvoudig de invoer of uitvoer van een programma omleiden naar een andere bron dan 'stdin', 'stdout' of 'stderr'.

De meest eenvoudige, maar zeer krachtige, I/O-omleidingsoperatoren zijn:

>	uitvoer (<code>stdout</code>) omleiden naar bestand;
1>	zelfde als >
2>	<code>stderr</code> omleiden naar bestand;
&>	<code>stderr</code> en <code>stdout</code> omleiden naar bestand;
>>	toevoegen aan bestand;
<	invoer (<code>stdin</code>) van bestand omleiden;
 	pipe de uitvoer (<code>stdout</code>) van het linker programma naar de invoer (<code>stdin</code>) van het rechterprogramma

De volgende voorbeelden illustreren deze operatoren. Voer deze commando's in een terminal sessie en bekijk zelf het resultaat.

pwd > list.txt

pwd drukt het resultaat af naar het bestand 'list.txt' in plaats van `stdout`;

ls -l >> list.txt

de uitvoer van `ls -l` wordt toegevoegd aan het bestand 'list.txt';

cat < list.txt

cat gebruikt 'list.txt' als invoer in plaats van `stdin`;

tree | more

de uitvoer van `tree` wordt gegeven als invoer voor `more` (controleer dit in een map waar de uitvoer van `tree` uit meerdere terminalpagina's bestaat).

Opmerking: `/dev/null` wordt vaak gebruikt om uitvoer te verwijderen. Bijvoorbeeld, `ls > /dev/null` zal alle uitvoer van `ls` niet op het scherm tonen of opslaan.

Voer, vanuit je home-folder `ls -R /etc > lsout.txt` uit om recursief een lijst te maken van alle inhoud in de `/etc` map en de uitvoer om te leiden naar het bestand `lsout.txt`. Bekijk het bestand met `cat`, `less` en `tail` en merk de verschillen tussen deze commando's.

Split nu dit bestand in kleinere bestanden van telken 10KB, met telkens een andere naam: `lsout1`, `lsout2`, `lesout3`, ...

1.6 Zoeken doorheen het bestandssysteem

Een van de belangrijkste zoekprogramma's in een Linux terminal is `find`. Het is een krachtig hulpmiddel om bestanden te vinden. Het vertrouwt niet op een database zoals `locate`, maar doorzoekt het bestandssysteem op zoek naar bestanden en mappen die aan de opgegeven criteria voldoen. Nogmaals, raadpleeg de manpagina's voor een gedetailleerd overzicht van alle mogelijke opties. Het gebruik wordt hieronder geïllustreerd met een paar voorbeelden.

`find <path> -name <bestandsnaam>`

doorzoek het onderliggende bestandssysteem, beginnend bij de map op locatie `<path>` voor de gegeven `<bestandsnaam>`;

`find <path> -size +5000K`

doorzoek het onderliggende bestandssysteem, beginnend bij `<path>` voor bestanden groter dan 5000KB;

`find <path> -perm -a+r`

doorzoek het bestandssysteem voor bestanden die door iedereen kunnen worden gelezen (`a = all`);

`find <path> -type d`

doorzoek het bestandssysteem voor bestanden van het type directory (`d = directory`, `f = file`, `s = socket`, `p = pipe`, `l = link`, ...);

De bestandsnaam waarnaar gezocht moet worden, kan gedefinieerd worden met 'wildcards', bijvoorbeeld: `find -name main*.c`.

Opmerking: als je het commando `find` niet als root uitvoert, geeft `find` een foutmelding op 'stderr' voor elke map waarvoor je geen leesrechten hebt. Dit kan veel gegevens op je scherm genereren zodat je de effectieve bestandsmatches over het hoofd ziet. Natuurlijk kun je het `find` commando uitvoeren met root rechten (`sudo`) of, als dat niet mogelijk is, alle foutmeldingen negeren door 'stderr' als volgt om te leiden: `find <path> <opties> 2>/dev/null`

De meeste Linux distro's bevatten een woordenboek, wat gewoon een leesbaar bestand is met woorden op elke regel. Meestal wordt dit woordenboek geopend met `words`. `words` is niet echt een bestand, maar een link naar een bestand met de echte woordenlijst, zoals de bestanden `american-english` of `british-english`. Waar bevinden deze bestanden zich? En is `words` gekoppeld aan Amerikaans of Brits Engels?

Tip: beperk de zoekopdracht tot `/usr/share` voor een snel antwoord.

Duik verder in de manpage van `find` en zoek nu naar alle bestanden in de `/etc` folder die de voorbije maand zijn aangepast.

Daar waar `find` enkel op naam en eigenschappen van bestanden, zou je ook kunnen zoeken op inhoud van (tekst) bestanden. Hiervoor bestaat er `grep`. Het is een krachtig hulpmiddel om in een bestand of bestanden te zoeken naar regels die overeenkomen met het gegeven patroon. We beperken onze uitleg tot het basisgebruik van `grep -E` of `egrep` waarbij reguliere expressies worden gebruikt om het patroon te bouwen. Reguliere expressies worden in meer detail besproken in het volgende topic. Voorlopig werken

we alleen met tekenreeksen die ‘letterlijk’ in het bestand moeten voorkomen - er worden nog geen speciale tekens gebruikt. Egrep kan ook zoeken in invoer van ‘stdin’ of een pipe. Een paar voorbeelden verduidelijken het gebruik van egrep:

egrep <regex> <bestand(en)>

zoek alle regels met <regex>, een reguliere expressie die definieert waarnaar je zoekt, in de invoerbestanden <bestand(en)>;

egrep 'if' main.c

zoek alle regels met ‘if’ in main.c;

egrep 'if' *.c

zoek alle regels met ‘if’ in .c-bestanden in de huidige map;

cat *.c | egrep 'if'

vergelijkbaar met het bovenstaande (maar zonder bestandsindicatie)

egrep -i 'if' *.c

zoek alle regels met ‘if’ in .c-bestanden in de huidige map en negeer hoofdlettergevoeligheid;

Ook hier kan je voor al deze methoden meer info vinden in de manpages.

1.6.1 Reguliere expressies

Wildcards zijn een begin om zoekpatronen te definiëren, maar reguliere expressies gaan veel verder dan wildcards en zijn veel krachtiger. Zonder volledig te willen zijn, leggen we hier de basisprincipes van reguliere expressies uit.

Reguliere expressies bevatten ‘letterlijk’ en ‘speciale’ tekens. Letterttekens specificeren precies de tekens die teken-voor-teken moeten worden geëvenaard. Sommige ‘speciale’ tekens hebben dezelfde betekenis als in jokertekens, maar de meeste hebben een andere betekenis en zijn veel krachtiger.

\	escape character;
[]	komt overeen met elk teken dat is opgegeven met []; – wordt gebruikt om een bereik aan te geven, bijvoorbeeld 0–9 of a–z;
[^]	logisch niet van [] : komt overeen met elk teken dat niet voorkomt in []; (merk het verschil met wildcards!)
^	geeft een anker aan dat aangeeft dat een match aan het begin van de regel moet staan
\$	geeft een anker aan dat aangeeft dat een match aan het einde van de regel moet staan
.	komt overeen met elk afzonderlijk teken, behalve einde van regel (bij jokertekens: ?);
+	komt overeen met <i>een of meerdere</i> matches van de voorgaande bouwstenen;
*	komt overeen met <i>nul of meerdere</i> matches van de voorgaande bouwstenen;
?	komt overeen met <i>nul of één</i> match van de voorgaande bouwstenen;
{x,y}	komt overeen met <i>x tot y keer</i> een match van de voorgaande bouwstenen;
{x}	komt overeen met <i>exact x keer</i> een match van de voorgaande bouwstenen;
{x,}	komt overeen met <i>x keer of meer</i> een match van de voorgaande bouwstenen;
{,y}	komt overeen met <i>y keer of minder</i> een match van de voorgaande bouwstenen;

Ter verduidelijking een aantal voorbeelden :

^[A–Z] komt overeen met elke regel in een bestand dat begint met een hoofdletter;

[+-]?[0–9]
komt overeen met gehele getallen, met of zonder teken;

main.\.c komt overeen met alle bestanden zoals main1.c, mainb.c, main+.c, enz;

r[0-9]{7} komt overeen met een sequentie startende met **r** gevolgd door 7 cijfers. Bv. r0367934 (cfr. je studentennummer)

Meer details zijn te vinden in de man pages met `man 7 regex`. Om verder te oefenen met reguliere expressies of om bepaalde patronen snel te testen kan je gebruik maken van de gratis online omgeving <https://regex101.com/>

1.7 Bestands- en tekstmanipulatie

Voor het vervangen of weglaten van tekens bestaan er een aantal eenvoudige commando's. Zo heb je `tr` (translate) voor het vervangen van een teken of tekenreeks door een andere set van tekens. `cut` kan knippen in een tekenreeks. De positie van dit knippen kan op basis van de index in de string of op basis van scheidingstekens, zoals een spatie of een punt-komma. Ter illustratie hieronder een paar voorbeelden, maar voor de volledigheid kan je ook hier best de manpage van deze commando's nalezen.

cut -d: -f1 /etc/group

splitst elke lijn in de `/etc/group` file op basis van de `:` en print het eerste element, in dit geval de groepnaam;

grep jef /etc/passwd | cut -d: -f6

Zoek naar de lijn met het woord *jef* in het bestand `/etc/passwd`, knip het vervolgens met als scheidingsteken `:` en druk het zesde element op het scherm. In dit geval de home-folder voor de gebruiker Jef.

cat demo.csv | tr , ;

stuurt de inhoud van het `demo.csv` (comma-separated-value) bestand, naar `tr` en vervang elke komma door een punt-komma.

We kunnen deze paragraaf niet afsluiten zonder `awk` te vermelden. `Awk` is een volledige taal voor het scannen en verwerken van tekstpatronen, gemaakt door Aho, Weinberger & Kernighan (vandaar de naam). Het kan behoorlijk geavanceerd zijn, we geven je hier enkel een voorproefje van wat `awk` kan. Ondanks zijn complexiteit kan je ook al met een aantal eenvoudige commando's, krachtige dingen verwezenlijken. Ook hier is er weer de manpage voor meer uitleg. Maar een online tutorial is hier ook zeker aan te raden.

`awk` neemt als argument een string met een scan en/of een verwerk opdracht. De scanopdracht staat tussen `/` `/` en de verwerking geef je aan met `.`. Standaard knipt hij elke gevonden regel in elementen op basis van witruimte, maar je kan ook een ander scheidingsteken kiezen zoals bij het `cut` commando. Eenvoudig kan je stellen dat `awk` een samensmelting is van `grep` en `cut`, maar met 'super powers'.

Ook hier illustreren we aan de hand van een paar voorbeelden de werking en het gebruik van `awk`.

awk '/if/' main.c

zoek alle regels met 'if' in `main.c`;

awk -F: '{print \$1}' /etc/group

splitst elke lijn in de `/etc/group` file op basis van de `:` en print het eerste element, in dit geval de groepnaam;

awk -F: '/jef/ {print \$6}' /etc/passwd

Zoek naar de lijn met het woord *jef* in het bestand `/etc/passwd`, knip het vervolgens met als scheidingsteken `:` en druk het zesde element op het scherm. In dit geval de home-folder voor de gebruiker Jef;

awk -F: '/bash/ {if (\$3 > 100) print \$1}' /etc/passwd

Zoek naar de lijn met het woord *bash* in het bestand `/etc/passwd`, knip het vervolgens met als scheidingsteken `:` en druk het eerste element (de gebruikersnaam) op het scherm, op voorwaarde dat het derde element (userID) groter is dan 100;

Gebruik de output van `df` om de naam en de vrije ruimte op elk device beginnende met `/dev` op het scherm te printen met een pijltje tussen de twee records. Een mogelijke output zie je hieronder.

```
/dev/sda1 --> 84679976
/dev/sdb1 --> 456777760
```

Tip: naast `print` bestaat er ook een `printf` functie bij `awk`

1.8 Super user (root)

De superuser of root is een speciale gebruiker die alle rechten en permissies heeft in alle modi. De superuser wordt meestal gebruikt voor systeembeheertaken. Twee commando's zijn de moeite waard om te ontdekken (zie man pagina's voor details):

su wisselen van gebruiker / superuser worden;
sudo een commando uitvoeren als een andere gebruiker / superuser.

Vooralsudo is erg interessant om te gebruiken als je een enkel commando wilt uitvoeren zonder de juiste rechten te hebben. Maar pas op voor de gevolgen! De applicatie zal starten als super user en alle bestanden die worden aangemaakt of bewerkt zullen alleen toegankelijk zijn met super user rechten.

Gebruik dit commando daarom alleen als je er zeker van bent dat dit geen verdere problemen oplevert. Bv. een commando uitvoeren met sudo dat bestanden of mappen in je homemap zal aanmaken/wijzigen is nooit een goed idee. Dit zal leiden tot situaties waarin je als gewone gebruiker geen toegang hebt tot de bestanden in je eigen homedirectory.

1.9 Machtigingen voor bestanden

Elk bestand heeft drie toegangsrechten: lezen (r), schrijven (w) en uitvoeren (x). Programma's moeten de rechten voor uitvoeren hebben om te kunnen starten. Deze rechten kunnen worden ingesteld voor drie groepen gebruikers: de gebruiker(s), de groep en anderen. De gebruiker is meestal de eigenaar van het bestand; gebruikers in dezelfde groep als de eigenaar van het bestand krijgen de groepspermissies; alle anderen krijgen de anderen-permissies. Het concept 'groep' wordt hieronder uitgelegd.

De rechten van een bestand kunnen eenvoudig gecontroleerd worden met `ls -l`. Stel dat de rechten voor een bestand `rw-rw-r--` zijn. De eerste drie rechten geven aan dat de gebruiker lees-, schrijf- en uitvoerrechten heeft op het bestand, de volgende drie rechten geven aan dat de groep het bestand alleen kan lezen en schrijven en de laatste drie rechten geven aan dat alle anderen het bestand alleen kunnen lezen.

Samengevat is het protectie systeem gebaseerd op drie klassen van gebruikers:

- eigenaar (*owner*): gewoonlijk de gebruiker die het bestand creëerde;
- groep (*group*): de groep waartoe de eigenaar behoort;
- anderen (*others*): al de rest van de gebruikers

Per klasse kunnen drie toegangsprivileges gegeven worden:

- r : het bestand mag gelezen worden
- w : het bestand mag gewijzigd of verwijderd worden
- x : het bestand mag uitgevoerd worden of de directory mag doorzocht worden

De rechten van bestanden en mappen kunnen veranderd worden met `chmod`. Laten we dit commando illustreren met een voorbeeld: `chmod u+x, o=wx demo.dat`

Met deze instructie worden uitvoerbare rechten toegevoegd (+) aan de 'gebruiker' en schrijf- en uitvoerbare rechten afgenomen (-) van 'anderen' van 'demo.dat'. Gebruik de man pages voor meer informatie over

chmod.

Opmerking: de permissies voor gebruikers, groepen en anderen worden soms binair gedefinieerd. Elke permissie (r, w, x) voor elk type gebruiker (gebruiker, groep, anderen) kan worden ingesteld met 1 bit aan informatie. In totaal zijn er 3 sets van 3 bits nodig. Elke groep van 3 bits wordt gegeven als een decimaal getal, variërend van 0 tot 7. Bijvoorbeeld `chmod 764 demo.dat`. Een overzicht van deze codering kan je vinden in tabel 1

decimaal	binair	privilege	decimaal	binair	privilege
0	000	---	4	100	r--
1	001	--x	5	101	r-x
2	010	-w-	6	110	rw-
3	011	-wx	7	111	rwX

Tabel 1: decimale codering van toegangsmachtigingen

Oefening

Ga naar `~/bsys2/oefening1` en maak een `main.c` bestand dat 'Hello World!' op de terminal print. Gebruik `gcc main.c` om dit bestand te compileren. Open indien nodig `main.c` met `vi` en herstel de fouten. Zodra `main.c` is gecompileerd, moet er een uitvoerbaar bestand genaamd `a.out` worden gemaakt. Controleer of dit bestand uitvoerbare rechten heeft en zo ja, voer het programma dan uit.

Gebruik `chmod` om de uitvoerbare rechten van de gebruiker van `a.out` te verwijderen en/of terug in te stellen. Voer het programma uit om het verschil te merken.

Gebruik `chmod` om de schrijfrechten van de gebruiker van `a.out` te verwijderen en/of terug in te stellen. Bewerk het programma en compileer opnieuw om het verschil te zien.

1.10 Gebruikers en groepen

Groepen in Linux worden gebruikt voor een betere toegangscontrole. Om dat te begrijpen, kun je het volgende voorbeeld bekijken. Een bestand 'demo.dat' mag niet toegankelijk zijn voor elke gebruiker op het systeem. Om de toegang tot 'demo.dat' beter te controleren, maak je een groep 'abc' aan en voeg je alle gebruikers die toegang hebben tot 'demo.dat' toe aan de groep 'abc'. Stel tenslotte de bestandsrechten van 'demo.dat' in op iets als `rwXrwx---` zodat de eigenaar en de groep lees-/schrijf-/uitvoerrechten hebben, maar andere gebruikers helemaal geen rechten. Het bestand `/etc/group` bevat een overzicht van alle bestaande groepen op het systeem. Druk dit bestand af om alle groepen op je systeem te krijgen. Het commando `id` of `groups` kan gebruikt worden om alle groepen waartoe je behoort op te vragen. Probeer dit eens.

Met het commando `chown` is het mogelijk om de eigenaar en groep van een bestaand bestand of map te wijzigen. Met de optie `-R` kun je zelfs recursief de wijzigingen toepassen op een hele mapstructuur. Als je alleen de groep wilt wijzigen, kun je ook het commando `chgrp` gebruiken.

Hoewel dit buiten het bereik van deze handleiding valt, vermelden we de volgende commando's om gebruikers en groepen te beheren:

useradd / groupadd

een gebruikersaccount (groep) aanmaken of bijwerken;

userdel / groupdel

een gebruikersaccount (groep) en gerelateerde bestanden verwijderen;

usermod / groupmod

een gebruikersaccount (groep) aanpassen;

passwd

het gebruikerswachtwoord wijzigen;

chfn info over de gebruiker wijzigen (echte naam en andere info, meestal gebruikt door *finger*).

Gedetailleerde informatie is uiteraard te vinden op de info/man pagina's.


1.11 De vim-editor

Tekstbestanden kunnen gecreëerd en gewijzigd worden met vim. Deze editor heeft twee basis modes:

- commando mode
- tekst invoer mode

Wanneer men vim opstart komt men in commando mode. Overgaan naar tekst invoer mode kan met volgende commando's gebeuren:

i	invoeren van tekst voor de cursor positie
a	invoeren van tekst na de cursor positie
o	invoeren van tekst op de volgende lijn
O	invoeren van tekst op de vorige lijn
R	overschrijven van tekst

Tijdens tekst invoer mode kan alleen met de  toets naar links bewogen worden. Terugkeren naar commando mode gebeurt m.b.v. de **Esc** toets. In deze mode kan met de pijltjes over de tekst bewogen worden. Ook kunnen **NEXT** en **PREV** gebruikt worden om de cursor over 24 lijnen naar onder of naar boven te bewegen. Andere positioneringen:

<i>lijn</i> nr G	naar de lijn met lijnnummer <i>lijn</i> nr
G	naar het einde van het bestand;
^	naar het begin van een lijn
\$	naar het einde van een lijn
/zoekarg	naar het eerst volgende voorkomen van <i>zoekarg</i>
n	naar het volgende voorkomen van <i>zoekarg</i>
?	naar het vorige voorkomen van <i>zoekarg</i>

Tekst verwijderen kan op verschillende manieren gebeuren:

dd	de lijn waarop de cursor staat
dw	het woord waarop de cursor staat
D	vanaf de cursor tot op het einde van de lijn
x	het teken op de positie waar de cursor staat
r	overschrijven van één teken (geen verwijdering)

Tekst kopiëren:

yy	de lijn waarop de cursor staat wordt in een buffer geplaatst
p	inhoud van de buffer wordt toegevoegd na de cursorlijn
P	inhoud van de buffer wordt toegevoegd voor de cursorlijn
J	samenvoegen van twee opeenvolgende lijnen

Het *yy* en *dd* kan voorafgegaan worden door een getal. In dat geval wordt de bewerking uitgevoerd op het gespecificeerde aantal lijnen vanaf de lijn waarop de cursor staat.

Algemene commando's:

ZZ	bewaar de veranderingen en verlaat de editor
:wq	bewaar de veranderingen en verlaat de editor
:q!	verlaat de editor zonder de veranderingen te bewaren
.	herhaal vorige actie
u	maak vorige actie ongedaan
U	herstel de lijn in haar vorige toestand

1.12 Taakcontrole

Als je een programma (ook wel een ‘taak’ genoemd) voor een lange tijd uitvoert of als je meerdere programma’s tegelijk wilt starten zonder veel terminals te openen, moet je deze programma’s op de achtergrond uitvoeren. Verschillende commando’s en sneltoetsen kunnen je helpen om deze taken te organiseren in voorgrond en achtergrond.

<prog> & de & geeft aan dat het programma moet starten en op de achtergrond moet draaien;

ctrl+z Dit forceert het huidige lopende proces naar de suspendmodus; merk op dat **ctrl+c** een programma beëindigt - niet opschort; als het programma eenmaal is opgeschort, kun je het bg of het fg commando gebruiken;

bg het laatst opgeschorte programma begint op de achtergrond te draaien;

fg Het laatst opgeschorte programma begint op de voorgrond te draaien; vandaar dat je met **ctrl+z** en fg programma’s kunt pauzeren en hervatten;

jobs toont alle taken die op de achtergrond draaien in de terminal;

fg %<job> de opdracht krijgen met opdracht <job> (het nummer in de rangschikking van de opdracht jobs) op de voorgrond;

kill %<job>
beëindig de opdracht <job> (het nummer in de rangschikking van de opdracht jobs);

timeout <time> <prog>
start <prog> en als het programma na <time> nog steeds draait, wordt het gestopt; Bv.
timeout 10s <prog> : laat het programma maximaal 10 seconden draaien (s = seconden, m = minuten, h = uren, d = dagen);

watch <prog>
<prog> periodiek uitvoeren (standaard 2 seconden);

watch -n 10 <prog>
<prog> elke 10 seconden uitvoeren;

wait %<job>
wacht tot opdracht <job> klaar is; Dit commando kan ook een opsomming van opdrachten accepteren, in dat geval wacht het tot alle opdrachten in de lijst klaar zijn.

Zoals gebruikelijk kun je meer informatie over het gebruik van deze commando’s vinden in de man pages. Man pagina’s over de commando’s jobs, bg en fg kunnen echter ontbreken. Dit zijn Bash commando’s, wat betekent dat je er meer over kunt vinden in de Bash documentatie (bijvoorbeeld man bash).

Bovenstaande commando’s kunnen veelal enkel gebruikt worden voor taken die je zelf als gebruiker gestart hebt. Om meer informatie te vinden over taken van het systeem of andere gebruikers kan je ook één van onderstaande commando’s gebruiken.

top interactief hulpmiddel om een Linux procesoverzicht te tonen en te verversen; dit commando implementeert een enorme hoeveelheid opties en mogelijkheden (opmaak, sorteren, filteren, ...); Binnen de applicatie kan je met behulp van een aantal toetsen extra functies oproepen Bv.:

- **h** voor help;
- **d** om het vernieuwingsinterval te wijzigen;

- `V` om een boomstructuur van de processen te krijgen;
- `q` om af te sluiten;

ps geeft informatie weer over een momentopname van alle of een selectie van de actieve processen; zonder argumenten toont dit alleen processen die geassocieerd zijn met dezelfde terminal;

ps -ef info weergeven over alle processen op het systeem;

ps -u <user>
filterprocessen op <user>;

pgrep <string>
vindt alle processen die <string> in hun naam bevatten (niet in het procespad!);

kill <pid>
het proces met PID <pid> beëindigen; Als argument kan ook een signaalnummer of -naam worden doorgegeven; een lijst van beschikbare signalen kan worden gegeven met `kill -l`; bijvoorbeeld `kill -9 <pid>` of `kill -SIGKILL <pid>` zou dezelfde actie tot gevolg hebben;

1.13 Bash scripting

Tot nu toe hebben we alleen opdrachten rechtstreeks in de terminal getypt. We kunnen ook een reeks opdrachten in een tekstbestand schrijven en deze opdrachten opeenvolgend door een Bash-terminal laten uitvoeren. Het bestand moet uitvoerrechten hebben voor de gebruiker en de eerste regel van het bestand moet een *Bash Shebang* bevatten. Dit zijn de 2 karakters `#!` gevolgd door de interpreter voor de volgende regels in het bestand. Voor een bash-script is dit dus `#!/bin/bash`. Merk op dat als je Python-code in het bestand zou zetten, dit ook kan worden uitgevoerd door `#!/usr/bin/python` als eerste regel in het bestand te plaatsen. Alle andere regels die met een `#` beginnen, worden beschouwd als commentaar.

In het bestand kun je alle opdrachten gebruiken zoals je die in een gewone terminal zou gebruiken. Maar je kunt ook variabelen definiëren en deze door het script heen verder gebruiken.

1.13.1 Variabelen

Er zijn twee soorten variabelen. Enerzijds shell variabelen. Deze zijn lokaal in het shell proces, ze worden niet doorgegeven naar child-processen. Voor deze namen gebruikt men kleine letters. Bij de declaratie van een variabele moet deze onmiddellijk geïnitieerd worden. Bijvoorbeeld `demo=4` (let op: er mogen geen spaties rond het `=` teken staan!)

Expansie van de variabele gebeurt met behulp van het `$` teken. Bijvoorbeeld `echo $demo` bij een gewone variabele. Maar variabelen kunnen ook lijsten bevatten, waar op basis van index elementen kunnen uit geselecteerd worden. Voorbeelden van expansie van variabelen zie je hieronder.

\$varnaam / \${varnaam}
geëxpandeerd naar de inhoud van de variabele. De `{}` symbolen zijn optioneel en dienen enkel om de naam te scheiden van andere woorden;

\${varnaam[i]}
Indien varnaam een lijst van woorden bevat: selecteer element met index `i` uit de lijst; index in de lijst start bij 0;

\${varnaam[@]}
Indien varnaam een lijst van woorden bevat: geeft alle elementen in de lijst weer;

\${#varnaam[@]}
Indien varnaam een lijst van woorden bevat: geeft het aantal elementen in de lijst weer;

Anderzijds zijn er de omgevingsvariabelen, deze zijn globaal in het shell proces. Voor deze namen gebruikt men hoofdletters. Zo heb je bijvoorbeeld `$PWD` dat het huidige pad bevat of `$HOME` die de home-folder van de huidige gebruiker bevat. Het commando `printenv` of `env` geeft een lijst van alle omgevingsvariabelen.

De argumenten die meegegeven worden bij het starten van het script worden ook ondergebracht in lokale variabelen. Zo is \$1 is het eerste argument, \$2 is het tweede, ... Alle argumenten kunnen opgevraagd worden met \$* en met \$# krijg je het aantal argumenten. \$0 bevat dan weer de naam van het bash script.

1.13.2 Expressies

Binnen een bash script gebeurt de evaluatie van een numerieke expressie binnen \$((...)). Voor numerieke variabelen zijn volgende operatoren mogelijk :

|| && | ^ & -le -ge -lt -gt -ne -eq << >> + - * / % ! ()

Tests op strings of status van bestanden worden enkel geëvalueerd door het test commando of door [[]]. Voor strings kan je gelijkheid (=) testen, maar ook alfabetische volgorde (< of >). In een expressie kan ook een status van een file opgevraagd worden: -l filenaam waarbij de letter l één van de onderstaande kan zijn. Een voorbeeld kan je vinden in figuur 3

d	is de filenaam een directory ?	r	heb ik lees toegang tot deze file ?
e	bestaat de filenaam ?	w	heb ik schrijf toegang tot deze file ?
f	is de filenaam een gewone file ?	x	kan ik deze file uitvoeren ?
o	ben ik eigenaar van de file ?	z	bevat deze file 0 bytes ?

```

1  #!/bin/bash
2  a=2
3  b=3
4  c=$(( $a + $b ))
5  echo "de som van 2 en 3 is $c"
6  [[ -e "demo.txt" ]] && echo "demo.txt bestaat" || echo "demo.txt bestaat niet"
7  test -d "/etc" && echo "/etc is een folder" || echo "/etc is geen folder"

```

Figuur 3: Voorbeeld bash script met expressies

1.13.3 Controle structuren

Binnen een bash script zijn een aantal controle structuren (of loops) mogelijk. Hieronder kan je een kort overzicht vinden van de belangrijkste structuren. Bekijk zeker ook man bash voor uitgebreide documentatie. Een voorbeeld van deze structuren kan je vinden in figuur 4

for i in 1..5 ; do <...> done

For lus voor het overlopen van een lijst. De lijst kan uit een variable komen, of uit een sequentie zoals in dit voorbeeld. De waarde wordt telkens in de variabele \$i geladen;

while <expr> ; do <...> done

While lus waarbij elke keer <expr> zal geëvalueerd worden;

if <expr> ; then <...> else <...> fi

If structuur voor het testen van <expr>

```

1  #!/bin/bash
2  for i in {1..5} ; do
3      echo "demo: ping -c1 192.168.0.$i"
4  done
5
6  x=1
7  while [[ $x -le 10 ]] ; do
8      echo $x
9      x=$(( $x + 3 ))
10 done
11
12 if [[ -e "demo.txt" ]] ; then
13     echo "demo.txt bestaat"
14 else
15     echo "demo.txt bestaat niet"
16 fi

```

Figuur 4: Voorbeeld bash script met controle structuren

Maak bash-shell programma met een bestandsnaam als argument. Indien het argument overeenkomt met een tekstbestand, wordt de inhoud van het bestand op het scherm getoond. Indien het argument overeenkomt met een directory, wordt een lijst met de in deze directory aanwezige bestanden eindigend op `.sh`, aan de gebruiker getoond.

Maak een shell programma om te zoeken in de group file naar groepen waarvan de group-id in een gegeven range ligt, display de bijhorende groepnaam en een overzicht van de leden waarbij je aangeeft of de home-folder van elk groepslid door jouw gebruiker doorzoekbaar is (optie `-x`). De range wordt ingegeven via argumenten bij het commando, bijvoorbeeld `./zoekleden.sh 200 470`.

2 Proces beheer

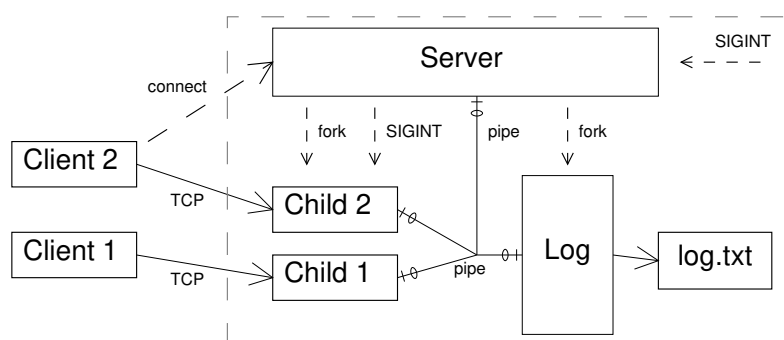
Leerdoelen

- Concepten van multi-process applications inoefenen
- Principe van inter-process-communication en file-descriptors binnen verschillende processen begrijpen
- Signals kunnen onderscheppen en sturen vanuit c-code

Om de concepten rond procesbeheer, signals en inter-process-communication (IPC) wat beter in de vingers te krijgen, implementeren we in deze oefening een Multi-Process-TCP-Echo-Server met log-functionaliteit. We noemen dit een echo-server omdat het de ontvangen tekstbericht van de client terugstuurt (echo) naar de client.

De basis bouwsteen van dit programma is het serverproces, dit implementeert de eigenlijke server voor het accepteren van de verschillende client-TCP connecties. Voor elke client die verbinding maakt met de server, wordt een nieuw proces aangemaakt via `fork()` om de requests van de client af te handelen. Het aanmaken van een proces moet dynamisch gebeuren want je weet van tevoren niet hoeveel clients er gaan verbinden.

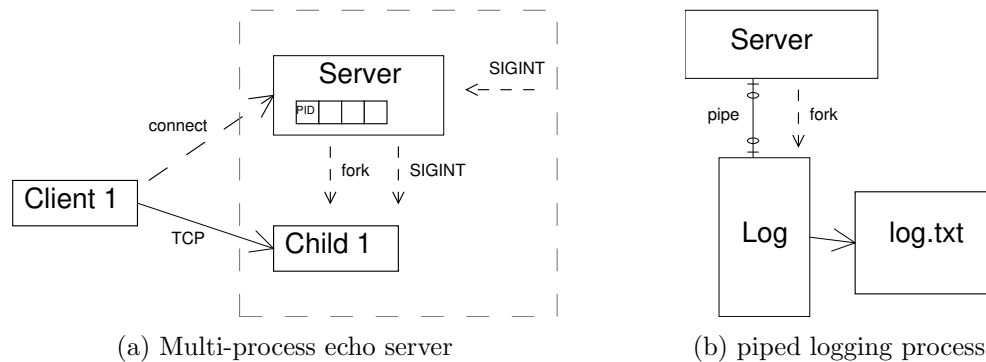
Verder is er een centraal log-proces dat via IPC, in dit geval `pipe()`, berichten krijgt van zowel het hoofdproces als de verschillende child-processen. Alle ontvangen berichten worden netjes genummerd en weggeschreven in een bestand `log.txt`. Het overzicht van de algemene werking van deze server zie je in figuur 5. Het deel binnen de stippellijn is te implementeren, de code voor de clients is aangeleverd.



Figuur 5: Echo server setup

De werking en implementatie van deze server kan je in twee blokken verdelen. Enerzijds is er de server voor het accepteren en afhandelen van de verschillende connecties. Zoals in figuur 6a. En anderzijds is er het log-proces dat berichten kan ontvangen via een `pipe()` en deze zal wegschrijven naar een log-bestand. Bij het ontvangen van een bericht zal dit proces telkens een index (teller) verhogen en deze samen met het bericht wegschrijven. Zodoende heeft elke lijn in het log-bestand een uniek nummer. Het prinsipschema zie je in figuur 6b. Een mogelijke opmaak voor dit logbestand zie je in figuur 7.

Voor de implementatie van de echo-server worden TCP-sockets gebruikt voor de communicatie met de clients. In de `/lib` folder bij de voorbeeldcode, kan je een volledige implementatie vinden voor het gebruik



Figuur 6: Echo server bouwstenen

van TCP-sockets. Bekijk `tcpsock.h` voor de documentatie van de functionaliteit die reeds aangeleverd is. We verwijzen ook graag naar de cursus ‘Computernetwerken’ voor meer informatie over TCP en socket calls. Voor testdoeleinden wordt de voorkeur gegeven om gebruik te maken van het lokale loop-back-netwerk met IP 127.0.0.1. De TCP poort voor deze connectie is vrij te kiezen en kan ingegeven worden via een `#DEFINE` in de code.

De code voor een eenvoudige client die periodiek een stukje tekst stuurt, is gegeven samen met voorbeeldcode van een eenvoudige server. In de gegeven servercode wordt echter slechts één cliëntverbinding tegelijkertijd ondersteund. Deze code moet je uitbreiden zodat na het accepteren van een connectie het parent-proces een nieuw child-proces start door middel van `fork()` om de data komende van deze connectie verder af te handelen. De server zal een dynamische array (gebruik hiervoor `malloc()` en `realloc()`) bijhouden van het PID voor elk child. Zodoende kan de server bij het ontvangen van een `SIGINT` (`ctrl`+`c`) dit signaal eerst delegeren naar alle child-processen en wachten tot deze afgesloten zijn (met `waitpid()`) vooraleer zelf af te sluiten. Telkens wanneer de server een nieuwe connectie (en een child-proces zal starten) accepteert, neemt hij ook de tijd om de dynamische lijst met PID’s te controleren. PID’s van child-processen die niet meer bestaan, omdat bijvoorbeeld de client zelf reeds afgesloten heeft, worden verwijderd uit de lijst.

Wanneer een child-proces een `SIGINT` ontvangt, zal deze eerst nog de tekst ‘closing connection’ naar de client sturen via de TCP-connectie voordat zelf af te sluiten.

Het log-proces start onmiddellijk bij aanvang van de server, zelfs voor het initialiseren van de TCP-sockets, en deelt een communicatiekanaal via `pipe()`. Bij het wegschrijven naar het `log.txt` bestand krijgt elke lijn een uniek nummer. Een aantal gebeurtenissen krijgen een record in het log-bestand, maar ook alle data die ontvangen is van elke client komt in dit bestand. Voor dit laatste zal niet enkel het parent-proces naar de pipe schrijven, maar zal ook elk child gebruik maken van de pipe. Een mogelijk voorbeeld van dit bestand kan je zien in figuur 7

```

1 001 Server started, accepting connections
2 002 client connected, child forked with pid 236
3 003 client connected, child forked with pid 268
4 004 received: Hello World!
5 005 received: My name is Bond...
6 006 client closed connection
7 007 server received SIGTERM, closing child connection
8 008 client closed connection
9 009 no more children, shutting down server

```

Figuur 7: voorbeeld output voor log.txt

Merk op dat voor het afsluiten van het log-proces je ook het `SIGINT` signaal kan delegeren, maar je kan ook de pipe gebruiken om het afsluiten te initiëren (zie man `pipe`).

2.1 Analyse

Probeer de werking van deze server zo goed mogelijk te begrijpen voordat je begint met de implementatie. Maak een sequence-diagram van de werking van de verschillende bouw stenen. Giet ook de verschillende

stappen tijdens het opstarten en afsluiten van de server in een sequence-diagram.

Bestudeer de voorbeeldcode en maak een plan hoe je je code zal organiseren. Hoe zal je de functionaliteit logisch verdelen over verschillende `.c` en `.h` bestanden? Hoe verloopt de communicatie via de pipe; gebruik je hiervoor een eigen protocol? Kortom, een hele boel vragen, voor je kan starten aan de eigenlijke implementatie.

2.2 Implementatie

Download de voorbeeldcode van Toledo. Begin met het uitbreiden van de `makefile` zodat `tcpsock.c` kan gebruikt worden als een dynamische library voor zowel een client als het serverprogramma. Merk op dat je een eigen client kan implementeren voor het zenden van berichtjes naar de echo-server, maar met de Linux tools zoals `socat` of `netcat` zou je ook kunnen verbinden aan je server.

Start bij de implementatie met één van de bouwstenen zoals aangegeven in figuur 6. Test dit onderdeel voor je start aan de tweede bouwsteen. Voor de echo-server zoals in figuur 6a schrijf je de ontvangen berichten naar `STDERR`, na het integreren van de twee bouwblokken kan je dit stuk dan vervangen door het schrijven naar de pipe.

3 Multithreading applicatie

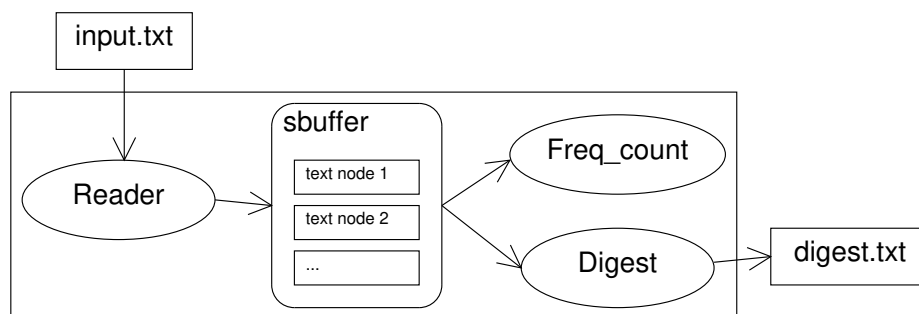
Leerdoelen

- Concepten van multi-threading applications inoefenen
- Principes en valkuilen van shared data tussen verschillende threads begrijpen
- Synchronisatie-structuren voor threads kunnen toepassen

In deze oefening bestuderen we een multi-threading producer-consumer opstelling, waarbij we starten één producer en meerdere onafhankelijke consumers. Elk onderdeel loopt in een afzonderlijke thread en zal de nodige data doorgeven via een geheugenstructuur die we hier een ‘shared buffer’ zullen noemen. (Dit principe wordt soms ook gerefereerd als ‘message queue’)

In figuur 8 zie je een overzicht van de applicatie. Het omvat 1 process waarin 3 threads gestart worden, elk van de drie threads heeft toegang tot een *gesynchroniseerde* geheugenstructuur (op de gezamenlijke heap).

De ‘reader’ thread zal tekst lijn-per-lijn inlezen uit een `input.txt` bestand en elke lijn toevoegen aan de ‘shared buffer’. De twee overige threads `Freq_count` en `Digest` lezen van de buffer en verwerken de tekst-nodes volgens hun eigen functionaliteit. Als een element verwerkt is, mag deze verwijderd worden uit de buffer. Merk op dat een element in de buffer pas verwijderd mag worden als deze verwerkt is door zowel de `Freq_count` thread als de `Digest` thread, alle data moet door de twee threads afzonderlijk verwerkt worden.



Figuur 8: Multi-threading applicatie-overzicht

De `Freq_count` thread zal nodes lezen van de ‘shared buffer’ en intern een frequentietabel bijhouden van de letters a–z doorheen alle nodes die hij verwerkt. Elke ontvangen tekst wordt hiervoor eerst omgezet naar kleine letters (lowercase) en op basis van deze tekst zal een interne array met frequenties voor elke letter aangepast worden.

De `Digest` thread zal een SHA1 digest berekenen van elk gelezen element uit de ‘shared buffer’ en deze, telkens op een nieuwe lijn, wegschrijven naar het bestand `digest.txt`.

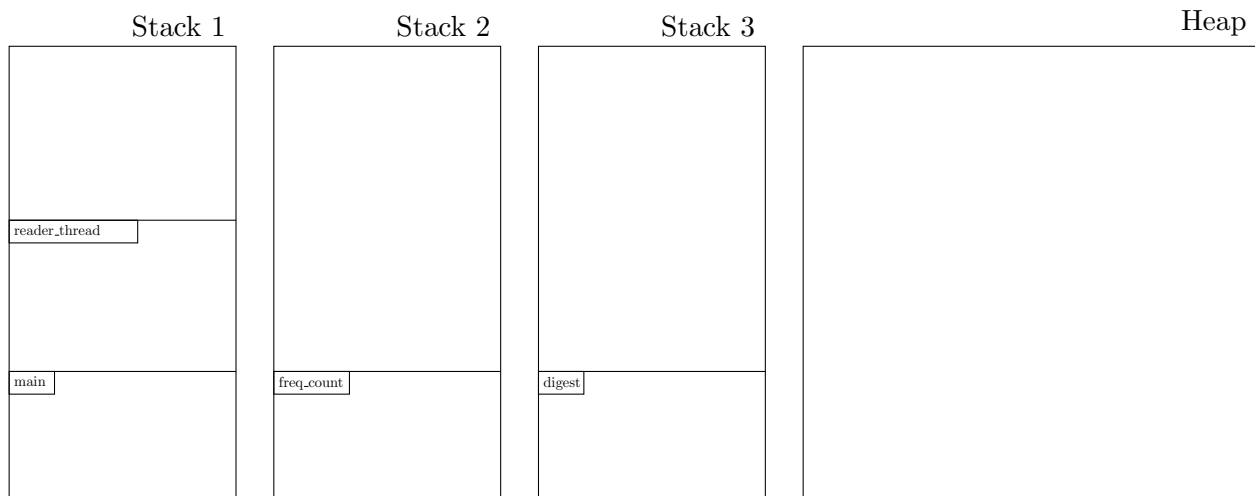
3.1 Analyse

Probeer de werking van deze oefening zo goed mogelijk te begrijpen voordat je begint met de implementatie. Om je hierbij te helpen bekijk je eerst de voorbeeldcode voor de ‘shared buffer’ die je kan vinden op Toledo. Dit geeft je een beeld welke functionaliteiten verwacht worden bij de werking van de buffer. Het geeft je ook een beeld van de datastructuur voor het opslaan van de verschillende nodes in de buffer.

Let wel dat je deze voorbeeld implementatie verder moet uitbreiden, er zijn bijvoorbeeld nog geen synchronisatie mechanismen aanwezig voor een ‘thread-safe’ werking van de buffer. Ook eventuele logica voor het conditioneel verwijderen (enkel wanneer alle consumers de data gelezen hebben) is nog niet geïmplementeerd.

Om meer inzicht te krijgen teken je de geheugen-layout voor een bepaald scenario. Hiervoor kan je je baseren op figuur 9, deze geeft een mogelijke layout voor een applicatie met drie threads. Merk op dat elke thread zijn eigen stack krijgt, maar dat ze allen een gezamenlijke heap delen.

Teken nu de geheugen-layout voor het volgende scenario: De `reader` thread heeft reeds 5 lijnen van `input.txt` gelezen en deze toegevoegd aan de buffer via `sbuffer_insert`. De `freq_count` thread, heeft de 4 eerste reeds afgewerkt en zijn frequentietabel aangepast. De `digest` thread heeft nog slechts 2 nodes afgewerkt en zal net starten aan zijn derde element. De nodes die beide threads reeds hebben afgewerkt zijn reeds verwijderd uit de buffer.



Figuur 9: Voorbeeld Stack - Heap tekening

Probeer bij het maken van de tekening een zicht te krijgen hoe je de eventuele logica voor het conditioneel verwijderen van nodes in de buffer gaat implementeren.

Maak als laatste een plan hoe je je code zal organiseren. Hoe zal je de functionaliteit en threads logisch verdelen over verschillende `.c` en `.h` bestanden?

3.2 Implementatie

Download de voorbeeldcode van Toledo. Begin met het uitbreiden van de `makefile` zodat al je mogelijke bronbestanden worden gecompileerd.

Voorzie in `sbuffer.c` de nodige synchronisatie structuren zodat deze code in een multithreading applicatie kan gebruikt worden. Denk hierbij bv aan een mutex, semaphore, rw-locks of condition-variable. Welke zijn nuttig en toepasbaar in deze situatie?

Pas alle regels voor ‘clean code’ toe en refactor wanneer nodig. Denk hierbij bijvoorbeeld aan:

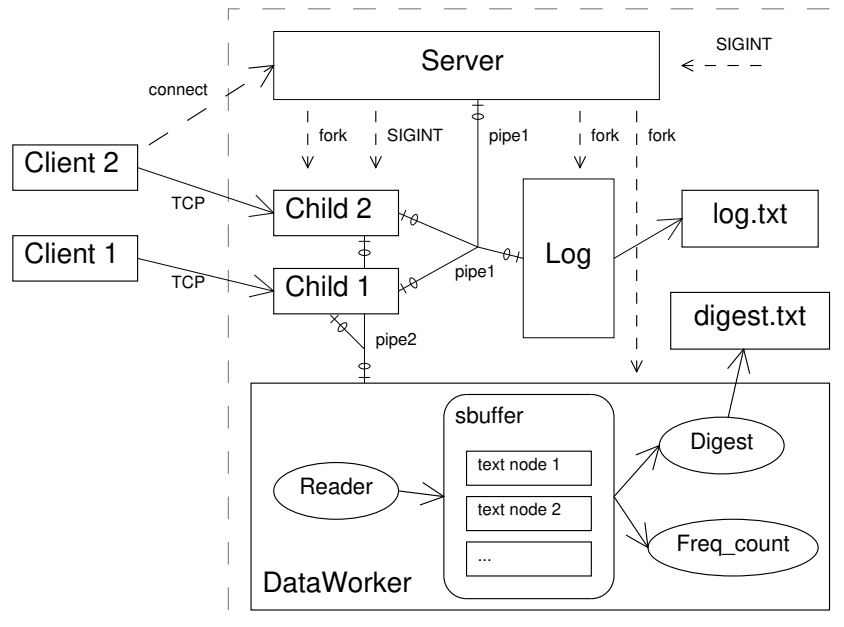
- probeer code duplication te vermijden
- let op indentatie
- gebruik begrijpbare en consistente benamingen voor methoden en variabelen
- probeer het gebruik van globale variabelen te vermijden waar mogelijk.
- ...

4 Multithreading server

Als laatste stap kunnen we nu de code van opgave 2 en 3 samenvoegen om zo te komen tot een multifunctionele server. De code van opgave 3 zal via een `fork` een afzonderlijk process vormen binnen de code van oefening 2.

Het lezen van het `input.txt` bestand in oefening 3 zal nu vervangen worden door het lezen van `pipe2` waar alle ‘childs’ hun ontvangen data in dumpen.

Een overzicht van de totale structuur zie je in figuur 10. Hier herken je het bovenste blok als de structuur zoals geïmplementeerd in opgave 2, en in het onderste gedeelte herken je de structuur zoals in opgave 3



Figuur 10: Multi-threading applicatie-overzicht

Merk op dat data nu met tussenpozen kan ontvangen worden, in tegenstelling tot het lezen van een bestand dat een continue stream van data leverde. De `freq_count` thread en de `digest` thread moeten nu kunnen ‘wachten’ op nieuwe data in de buffer. Deze situatie vereist een uitgebreidere synchronisatie, moest je deze nog niet geïmplementeerd hebben, met behulp van bijvoorbeeld een conditional variable of semaphore.