Naloga 3

Naloga 3 - mysh

Tokratna navodila so res malce daljša, kar pa še ne pomeni, da je naloga tudi tako obsežna. Večino ukazov zlahka spišete v nekaj minutah. Preden se lotite programiranja, razmislite, kako boste zasnovali celoten program. Težji del naloge s skriva v pravilni izvedbi izvajanja v ozadju, preusmerjanja in cevovodov. Veliko uspeha.

V programskem jeziku C napišite ukazno lupino. Program naj omogoča vnašanje ukazov v ukazno vrstico (ena vrstica en ukaz), po pritisku tipke enter, pa naj se ukaz izvede. Podprite spodaj naštete notranje ukaze. Če ukaz ni notranji, naj se izvede ustrezen zunanji ukaz (preko `fork` & `exec`). Podprite tudi izvajanje v ozadju, preprosto preusmerjanje in izvedbo cevovoda. Lupina naj podpira tako interaktivni kot neinteraktivni način delovanja. Pri razpoznavanju ukaznih vrstic ni potrebno preverjati za morebitne napačne vnose - predpostavite lahko, da uporabnik vedno vtipka sintaktično pravilno vrstico. Ne pa nujno semantično pravilno, zato naj bo preverjanje napak le minimalno - pri sistemskih klicih (na primer `chdir()`, `mkdir()`, `opendir()`, `open()`) preverite le njihov status in po potrebi izpišite obvestilo o napaki s `perror()`.

Uporaba funkcije `system()` in podobnih ni dovoljena. Uporabljajte le funkcije, ki smo jih omenili na vajah - večina so to ovojne funkcije sistemskih klicev, z izjemo `printf()`, `scanf()`, ipd.

Glejte tudi priloženi primer izvajanja.

REPL - read, eval & print loop

Med interaktivnim (ročno vnašanje ukazov) in neinteraktivnim (skriptni način) načinom lahko ločimo s pomočjo funkcije `isatty(1)`. Če program zaženemo s presumeritvijo vhoda, gre za neinteraktivni način, npr. `./mysh <skripta.sh`, sicer pa za interaktivnega. Ukazno lupino sprogramiramo po principu repl:

- v interaktivnem načinu izpišemo pozivnik - ime lupine in "> ", npr "`mysh`> ",
- preberemo eno vrstico s standardnega vhoda (read),
- jo izvedemo (eval),
- izpišemo rezultat na standardni izhod (print) in
- ponavljamo (loop).

Format ukaznih vrstic je kar se da preprost:

- vsaka vrstica vsebuje kvečjemu en ukaz,
- prazne vrstice oz. vrstice, ki vsebujejo le bele znake (funkcija `isspace()`) ignorirajte,
- vrstice, katerih prvi ne-beli znak je # ignorirajte (komentarji),
- sicer pa vrstico razbijemo na simbole (angl. token),

    - simbol je zaporedje ne-belih znakov, npr. `ls`, `cd`, `echo`, `/etc/passwd`, `&`, `>out.txt`, `<in.txt`
    - prav tako je simbol tudi niz obdan z narekovajema, npr. "`echo`", "`ls`", "`vsebujem presledek`"
    - med dvema simboloma je vedno vsaj en beli znak - tako zaporedja kot je npr. `a+b` smatramo kot en simbol, `a + b`, pa kot tri simbole,
- prvi simbol je ime ukaza, ostali simboli so njegovi parametri,
- zadnji simboli lahko podajajo preusmeritev vhoda, preusmeritev izhoda ali izvajanje v ozadju, prisotnost teh simbolov je opcijska, njihovo **zaporedje je vedno** sledeče:
    1. morebitna preusmeritev vhoda v obliki <*datoteka*,
    2. morebitna preusmeritev izhoda v obliki >*datoteka*,
    3. morebitno izvajanje v ozadju z znakom `&`.
- predpostavite lahko, da so vrstice vedno v pravilni obliki:
    - vsi potrebni parametri ukaza so podani,
    - sintaksa ustreza zgornjemu opisu,
    - ni pa nujno, da so parametri smiselni, npr. preusmeritev vhoda na neobstoječo datoteko, ustvarjanje imenika, ki že obstaja itd.

Preprosti vgrajeni ukazi

- `name` *ime* - nastavi ime lupine, če imena ne podamo, izpiše ime lupine (privzeto ime je `mysh`),
- `help` - izpiše spisek podprtih ukazov, format izpisa je po vaši želji - se ne preverja avtomatsko,
- `status` - izpiše izhodni status zadnjega (v ospredju) izvedenega ukaza,
- `exit` *status* - konča lupino s podanim izhodnim statusom,
- `print` *args*... - izpiše podane argumente na standardni izhod (brez končnega skoka v novo vrstico),
- `echo` *args*... - kot print, le da izpiše še skok v novo vrstico,
- `pid` - izpiše pid procesa (kot $BASHPID),
- `ppid` - izpiše pid starša.

Vgrajeni ukazi za delo z imeniki

- `dirchange` *imenik* - zamenjava trenutnega delovnega imenika, če imenika ne podamo, skoči na /,
- `dirwhere` - izpis trenutnega delovnega imenika,
- `dirmake` *imenik* - ustvarjanje podanega imenika,
- `dirremove` *imenik* - brisanje podanega imenika,
- `dirlist` *imenik* - preprost izpis vsebine imenika (le imena datotek, ločena z dvema presledkoma), če imena ne podamo, se privzame trenutni delovni imenik.

Ostali vgrajeni ukazi za delo z datotekami

- `linkhard` *cilj ime* - ustvarjanje trde povezave na cilj,
- `linksoft` *cilj ime* - ustvarjanje simbolične povezave na cilj,
- `linkread` *ime* - izpis cilja podane simbolične povezave,
- `linklist` *ime* - izpiše vse trde povezave na datoteko z imenom ime,

- `unlink` *ime* - brisanje datoteke,
- `rename` *izvor ponor* - preimenovanje datoteke,
- `cpcat` *izvor ponor* - ukaz cp in cat združena (glej Izziv 5 - cpcat.c).

Cevovod

Cevovod podamo na naslednji način:

- `pipes` "stopnja 1" "stopnja 2" "stopnja 3" ...
  Primer:

  ```
  pipes "cat /etc/passwd" "cut -d: -f7" "sort" "uniq -c"
  ```

  je enako kot v bashu naslednji ukaz:

  ```
  cat /etc/passwd | cut -d: -f7 | sort | uniq -c
  ```

- prisotni bosta vsaj dve stopnji.

Izvajanje v ozadju

Če je zadnji simbol vrstice enak &, potem naj se podani ukaz izvede v ozadju. Pri tem upoštevajte nasledje:

- Pri izvajanju zunanjih ukazov je v vsakem primeru potrebno naredit `fork()` & `exec()`. Izvajanje v ospredju zahteva še `waitpid()`, pri izvajanju v ozadju pa `waitpid()` izpustite. Potrebno pa je seveda imeti tudi rokovalnik za pokopavanje zombijev.
- Pri izvajanju vgrajenih ukazov pa je potrebno malce pazljivosti. Razmislite sami, kdaj potrebujete `fork()`.
- Ne pozabite na rokovalnik signala SIGCHLD. Napotke kako ga pravilno implementirati dobite tukaj.

Preusmerjanje

Preusmerjanje izvedite, kadar je na koncu ukaza <*vhod* ali >*izhod* ali oboje. Temu lahko sledi še &.

Pri zunanjih ukazih je preusmerjanje preprosto, ker gre za **podproces** je dovolj le kombinacija klicev `open()` in `dup2()`. Podobno je z notranjimi ukazi, ki se izvajajo v ozadju. Razmislite zakaj?

Precej več pazljivosti pa je potrebno z notranjimi ukazi (v ospredju). Potrebno je shraniti trenutno stanje deskriptorjev, ki jih preusmerjamo, nato izvedemo preusmeritev in notranji ukaz, na koncu pa je potrebno obnoviti staro stanje deskriptorjev - sicer se vam lahko zgodi, da vam "izgine" vhod/izhod same lupine. Če uporabljate višje nivojske (buffered IO) funkcije za izpis, npr. `printf()`, je potrebno pred preusmeritvijo/obnovitvijo izhoda izvesti še `fflush(stdout)` da izpraznete medpomnilnik. V nasprotnem primeru gre lahko izpis popolnoma drugam kot programer pričakuje, na primer kadar je originalni izhod že preusmerjen, kot je na primer pri avtomatskem testiranju.

Dodatni namigi

Berite po svoji želji ali pa tud ne.

Splošni namigi:

- Namige lahko upoštevate ali pa tudi ne. Če ste si program zamislili drugače, ni nič narobe.
- Program čim bolj poenostavite. Uporaba globalnih spremenljivk lahko precej poenostavi program.
- Nizom in tabelam statično določite kapaciteto. Npr. za ime lupine definirajte `char name[64]`, podobno za ukazno vrstico itd. Izberite smiselno velike kapacitete.
- Uporabi `malloc()` in njegovih sorodnikov se tako lahko izognete.

Branje vrstice in razpoznavanje simbolov:

- Razpoznavanje sprogramirajte v eni funkciji, npr. `int tokenize(char * line)`. S kazalcem na char potujte po nizu `line` in z razpoznanimi simboli polnite globalno tabelo `char* tokens[MAX_TOKENS]` - vsakič ko razpoznate en simbol, ga shranite v tabelo. Poleg tega imejte še spremenljivko `int token_count`, ki pove kako dolga je tabela. `MAX_TOKENS` naj bo večji od 20.
- Ko se premikamo po nizu `line`, lahko za vsakim razpoznanim simbolom postavimo bajt 0 (konec niza). V tabelo `tokens` pa shranimo le kazalec na začetek simbola. Tako nam ni potrebno rezervirati dodatnega prostora in kopirati nizov.
- Vsak notranji ukaz implementirajte v svoji funkciji. Morebitne parametre ukaza enostavno preberete iz globalne tabele `tokens`.

Izvajanje v ozadju in preusmeritve:

- Ugotavljanje ali gre za izvajanje v ozadju ali preusmerjanje naredite po razpoznavanju simbolov.
- Preprosto preverite zaporedoma simbole od zadnjega proti prvemu: najprej ozadje, nato preusmeritev izhoda, nato preusmeritev vhoda.
- Preverite zadnji simbol, če gre za ozadje, si to zapomnite in zmanjšate `token_count`, nato preverite, če gre za preusmeritev izhoda, si zapomnite ime datoteke in zmanjšate `token_count`. In po istem principu še preusmeritev vhoda.
- Pripravite si funkcije za stvaritev začetne stopnje cevovoda, vmesne stopnje in končne stopnje. V vsaki stopnji poskrbite za ustrezno cev, stvaritev podprocesa in ustrezne preusmeritve.
- Ko je določena stopnja pripravljena lahko izvedete ustrezen ukaz - podan kot en simbol (v narekovajih) - izkoristite že napisano kodo.

Status oddaje naloge

Status oddaje naloge	Pri tej nalogi vam ni treba oddati ničesar.
	Pri tej nalogi ne morete oddati prispevkov.
Stanje ocen	Neocenjeno
Rok za oddajo	ponedeljek, 4. junij 2018, 23:55
Preostali čas	Rok za oddajo naloge je potekel
Zadnja sprememba	-
Komentar oddaje	▶ Komentarji (0)

Odziv

Ocena	7,00 / 12,00
Ocenjeno v	četrtek, 7. junij 2018, 14:45

NAVIGACIJA

Pregledna plošča
- Prva stran

Strani spletnega mesta

Trenutni predmet

OS

Sodelujoči

Priznanja

Splošno

O predmetu
Potek predmeta
Samostojno delo
Viri
Dostop do Linux sistemov
Obvestila
Forum
Opravljanje domačih nalog
Naloga 0 - Preizkus oddaje
Naloga 1
Naloga 2
proc-demo
Naloga 3
Primeri izvajanja
Točke iz izzivov
Dodatne točke
Primer izpita: os-1112-1
Primer izpita 10. 9. 2014
Primer izpita: 22.6.2017
Rezultati 1. rok - teorija
Rezultati 1. rok - praksa
Rezultati 2. rok - teorija
Rezultati 2. rok - praksa
Rezultati 3. rok - teorija

Rezultati 3. rok - praksa

5. marec - 11. marec

12. marec - 18. marec

19. marec - 25. marec

26. marec - 1. april

2. april - 8. april

9. april - 15. april

16. april - 22. april

23. april - 29. april

30. april - 6. maj

7. maj - 13. maj

14. maj - 20. maj

21. maj - 27. maj

Moji predmeti

Predmeti

NASTAVITVE

Skrbništvo predmeta