Enunt

Sa se implementeze un shell simplu, care suporta executia de comenzi externe cu argumente multiple, comenzi interne, redirectari, pipe-uri. Shell-ul trebuie sa suporte executia de comenzi compuse, cu oricati operatori.

Shell-ul trebuie sa suporte urmatorii operatori de executie:

- operatorul de secventiere "`;`", care va fi folosit pentru a executa comenzile "pe rand"; de exemplu, `expr1; expr2` va avea ca efect executia comenzilor `expr1` mai intai, si dupa terminarea executiei acestora se va continua cu executia comenzilor `expr2`
- operatorul de paralelism "`&`", care va fi folosit pentru a executa comenzile in paralel; de exemplu, `expr1 & expr2` va avea ca efect executia comenzilor `expr1` si a comenzilor`expr2` in paralel
- operatorul "`|`" care va fi folosit pentru a executa comenzile legate de acesta cu stdout-ul redirectat in stdin; de exemplu, `expr1 | expr2` va avea ca efect executia comezilor`expr1` cu stdout-ul redirectat in stdin-ul comenzilor `expr2`
- operatorii de executie conditionata "`&&`" si "`||`", care vor fi folositi pentru a executa comenzile in functie de codul de eroare; astfel `expr1 && expr2` va avea ca efect executia comenzilor `expr2` doar daca comenzile `expr1` au ca rezultat un cod de eroare 0, iar `expr1 || expr2` va avea ca efect executia comenzilor `expr2` doar daca comenzile `expr1` au ca rezultat un cod de eroare diferit de zero

Prioritatea operatorilor de executie, in ordine crescatoare, este:

- operatorul de secventiere
- operatorul de paralelism
- operatorii de executie conditionata
- operatorul |

Shell-ul trebuie de asemenea sa suporte si urmatorii operatori de redirectare:

- "`> fisier`" pentru redirectarea stdout-ului in fisier
- "`2> fisier`" pentru redirectarea stderr-ului in fisier
- "`&> fisier`" pentru redirectarea stdout-ului si a stderr-ului in fisier
- "`< fisier` pentru redirectarea stdin-ului din fisier

In fine, shell-ul trebuie sa suporte urmatoarele comenzi interne:

- `exit` si `quit` pentru terminarea shell-ului
- `cd director` pentru schimbarea directorului curent

Shell-ul trebuie sa suporte si variabile de mediu in forma $VARIABILA_DE_MEDIU.

Precizari generale

Mai jos sunt prezentate cateva exemple de comenzi si outputul generat de acestea:

```
> date && sleep 1; echo date
Sun Oct 24 10:41:43 EEST 2004
date

> date && sleep 1; date
Sun Oct 24 10:41:51 EEST 2004
Sun Oct 24 10:41:52 EEST 2004

> date && sleep 1 & date
Sun Oct 24 10:42:02 EEST 2004
Sun Oct 24 10:42:02 EEST 2004

> true && date
Sun Oct 24 10:42:27 EEST 2004

> false && date

> false || date
Sun Oct 24 10:43:37 EEST 2004

> cat /etc/services | grep telnet
telnet          23/tcp
rtelnet         107/tcp                         # Remote Telnet
rtelnet         107/udp
telnets         992/tcp                         # Telnet over SSL
telnets         992/udp
tfido           60177/tcp                       # fidonet EMSI over telnet

>

> gcc > tmp; echo sep; cat tmp
gcc: no input files
sep

> gcc 2> tmp; echo sep; cat tmp
sep
gcc: no input files

> cat < tmp
gcc: no input files

> pwd; cd $HOME; pwd
/home/tavi/cursuri/so/tema1
/home/tavi
```

```
> exit

tavi@pixie:~/cursuri/so/tema1$
```

Pentru a simplifica implementarea temei, puteti folosi parserul implementat de noi. Pentru detalii despre parser, cititi README.

Promptul afisat de shell este impus pentru a facilita testarea automata si este "\n> " (adica se va afisa linie noua inaintea lui, apoi caracterul '>' urmat de un spatiu). Numele executabilului temei trebuie sa fie 'tema1'. Pentru a trece testul 09, este obligatoriu sa respectati formatul mesajului de eroare impus. Mesajul de eroare trebuie scris la stderr si trebuie sa fie identic cu cel asteptat de teste (vezi "test_09_ref.txt" din teste).

Din cauza diferentei intre Windows si Linux la crearea de noi procese (CreateProcess vs. fork + exec*) s-ar putea sa nu puteti folosi acelasi tip de parcurgere a arborelui sintactic si pe Windows si pe Linux. Daca vreti sa reutilizati concepte/cod de pe Linux pe Windows, concepeti parcurgerea sa functioneze si cu CreateProcess de pe Windows.

Precizari Windows

Tema se va rezolva folosind doar functii Win32. Se pot folosi de asemenea si functiile de formatare `printf`, `scanf`, functiile de alocare de memorie `malloc`, `free` si functiile de manipulare a sirurilor de caractere (`strcat`, `strdup`, etc.)

Pentru partea de I/O si procese se vor folosi doar functii Win32. De exemplu, functiile `open`, `read`, `write`, `close` nu trebuie folosite, in locul acestor trebuind sa folositi `CreateFile`,`ReadFile`, `WriteFile`, `CloseHandle`.

Precizari Linux

Tema se va rezolva folosind doar functii POSIX. Se pot folosi de asemenea si functiile de formatare `printf`, `scanf`, functiile de alocare de memorie `malloc`, `free` si functiile de manipulare a sirurilor de caractere (`strcat`, `strdup`, etc.)

Pentru partea de I/O si procese se vor folosi doar functii POSIX. De exemplu, functiile `fopen`, `fread`, `fwrite`, `fclose` nu trebuie folosite, in locul acestor trebuind sa folositi `open`,`read`, `write`, `close`.

Testare

Pentru simplificarea procesului de corectare al temelor, dar si pentru a reduce greselile temelor trimise, corectarea temelor se va face automat cu ajutorul unor teste publice.

O tema care trece cele 9 teste automate va lua 10 puncte din 10 (daca nu triseaza folosind API interzis, cum ar fi functia system() in cazul temei 1, caz in care nu va fi punctata). Un test valoreaza un punct si se acorda suplimentar un punct din oficiu. Din aceste puncte se vor scadea automat puncte pentru intarzieri si pentru warning-uri. Se mai poate scadea suplimentar cand tema va fi revazuta din cauza nerespectarii criteriilor scrise la sectiunea reguli. Astfel:

-0.1 pentru makefile incorect (de exemplu compileaza de fiecare data totul)
-0.2 pentru README necorespunzator
-0.2 surse prost comentate
-0.4 neverificarea conditiilor de eroare sau/si neeliberarea de resurse
-0.1 diverse alte probleme constatate in implementare

In cazuri exceptionale se poate scadea mai mult decat este mentionat mai sus.

FAQ

Q: Cum pot sa citesc arhivele listei de discutii?
A: O varianta este cu Opera: File -> Import and export -> Import mail -> Import generic mbox file -> alegeti fisierele respective.

Q: Temele se pot face in C++?
A: Da.

Q: Am voie sa folosesc functii POSIX pe Windows?
A: La toate temele de SO pe Windows se va folosi Win32 API, nu POSIX. Oricum, fork si exec* nu sunt suportate pe Windows XP decat dupa instalarea "Unix Services for Windows".

Q: La temele de Windows, trebuie sa folosesc functiile de API in versiunea Unicode, ANSI, sau generica?
A: Nu este impus sa folositi o versiune anume. Aveti grija totusi sa folositi corect functiile (ele primesc parametri de tip CHAR pentru versiunea ANSI, WCHAR pentru Unicode si TCHAR in versiunea generica). Pentru detalii vezi Unicode in the Windows API.

Q: Ce fac daca am intalnit un caz limita al carui comportament nu este precizat in enunt?
A: In general la temele de SO, pentru cazuri limita ce nu apar in testele publice sau in enunt, se accepta orice comportament documentat in README. Un exemplu este comportamentul pentru "command | cd /something". Daca nu sunteti siguri, intrebati pe lista de discutii.

Q: Trebuie optimizat numarul de fork-uri? De exemplu daca am comanda: a|b|c trebuie sa am 3 forkuri sau pot sa am 4 sau 5?
A: Nu este obligatoriu sa optimizati numarul de fork-uri. Totusi, in general e bine sa aveti in vedere eficientizarea consumului de resurse.

Q: Shell-ul trebuie sa se comporte ca un shell adevarat (sh, bash) cand ... ?

A: Functionalitatea minima necesara este cea din enuntul temei. Daca implementati ceva in plus, precizati in README. Exemple de functionalitate care nu este ceruta: updatarea unor variabile de mediu (gen $OLDPWD si $PWD), command prompt cu detalii in genul user curent, masina si director curent, multe altele... (vezi man bash pentru o idee despre functionalitatea unui shell complet :-))

Q: Am voie sa nu folosesc parser-ul din enunt daca vreau sa scriu eu altul echivalent?

A: Da.