Tema 2 Mini-shell

Dată publicare: **16 Martie 2017**

Deadline: **5 Aprilie 2017, ora 23:55**

Deadline hard: **12 Aprilie 2017, ora 23:55**

Enunț

Să se implementeze un shell simplu, care suportă execuția de comenzi externe cu argumente multiple, comenzi interne, redirectări, pipe-uri. Shell-ul trebuie să suporte execuția de comenzi compuse, cu oricâți operatori.

Shell-ul trebuie să suporte următorii operatori de execuție:

- operatorul de secvențiere ”;”
  - va fi folosit pentru a executa comenzile “pe rând”;
  - de exemplu, `expr1; expr2` va avea ca efect mai întâi execuția comenzilor `expr1` și, după terminarea execuției acestora, execuția comenzilor `expr2`;
- operatorul de paralelism ”&”
  - va fi folosit pentru a executa comenzile în paralel;
  - de exemplu, `expr1 & expr2` va avea ca efect execuția comenzilor `expr1` și a comenzilor `expr2` în paralel;
  - în implementare **NU** aveți voie să vă reapelați singuri executabilul.

```
execv("./my_homework", "command");
```

```
CreateProcess(
      NULL,
      "my_homework.exe"
      NULL,
      NULL,
      FALSE,
      0,
      NULL,
      NULL,
      STARTUPINFO_ptr,
      PROCESS_INFORMATION_ptr
);
```

- operatorul ”|” (pipe)
  - va fi folosit pentru înlănțuirea comenzilor;
  - de exemplu, `expr1 | expr2` va avea ca efect execuția comezilor `expr1` cu stdout-ul redirectat în stdin-ul comenzilor `expr2`;
- operatorii de execuție condiționată ”&&” și ”||”
  - vor fi folosiți pentru a executa comenzile în funcție de codul de eroare;
  - `expr1 && expr2` va avea ca efect execuția comenzilor `expr2` doar în cazul în care comenzile `expr1` au ca rezultat un cod de eroare `0`;
  - `expr1 || expr2` va avea ca efect execuția comenzilor `expr2` doar în cazul în care comenzile `expr1` au ca rezultat un cod de eroare diferit de zero.

Prioritatea operatorilor de execuție este, de la cel mai prioritar la cel mai puțin prioritar:

Search

Informații generale SO

- Documentație și alte resurse
- Feed RSS
- Hall of SO
- Listă de discuții
- Mașini virtuale
- Trimitere teme

Informații SO 2016-2017

- **Examen**
  - Examen CA/CC 2012-2013
  - Examen CA/CC 2013-2014
  - Examen CA/CC 2014-2015
  - Examen CA/CC 2015-2016
- **Reguli generale și notare**
  - Notare CA/CB/CC
- Anunțuri
- Calendar
- Catalog
- Echivalări teme
- Karma Awards
- SO Need to Know
- Orar și împărțire pe semigrupe

Laboratoare

- **Resurse**
  - C/SO Tips
  - Macro-ul DIE
  - GDB
  - Resurse
  - Function Hooking and Windows Dll Injection
  - Oprofile
  - Recapitulare
  - Thread-uri - Extra
  - Visual Studio Tips and Tricks
  - windows-video
- Laborator 01 - Introducere
- Laborator 02 - Operații I/O simple
- Laborator 03 - Procese
- Laborator 04 - Semnale
- Laborator 05 - Gestiunea memoriei

1. operatorul |
2. operatorii de execuție condiționată
3. operatorul de paralelism
4. operatorul de secvențiere

Shell-ul trebuie, de asemenea, să suporte și următorii operatori de redirectare:

- "`< nume_fisier`" pentru redirectarea intrării standard din fișierul `nume_fisier`;
- "`> nume_fiser`" pentru redirectarea ieșirii standard în fișierul `nume_fisier`;
- "`2> nume_fisier`" pentru redirectarea ieșirii de eroare standard în fișierul `nume_fisier`;
- "`&> nume_fisier`" pentru redirectarea ieșirii standard și ieșirii de eroare standard în fișierul `nume_fisier`;
- "`>> nume_fisier`" pentru redirectarea ieșirii standard în fișierul `nume_fisier` în modul "append";
- "`2>> nume_fisier`" pentru redirectarea ieșirii de eroare standard în fișierul `nume_fisier` în modul "append".

În fine, shell-ul trebuie să suporte următoarele comenzi interne:

- `exit` și `quit` pentru terminarea shell-ului
- `cd director` pentru schimbarea directorului curent
  - **ATENȚIE**: Mini-shell-ul trebuie să funcționeze în continuare la introducerea comenzii `cd` fără parametru. Nu se impune implementarea comportamentului din `bash` (schimbarea directorului curent cu directorul *home* al utilizatorului curent) și nici o valoare de exit pentru acest scenariu (practic, `cd` fără parametru poate să nu facă nimic, dar să nu rămână cu comportament nedefinit care să ducă la erori).

Shell-ul trebuie să suporte variabile de mediu:

- formatul de utilizare este `$VARIABILA_DE_MEDIU` identic pe Linux și pe Windows;
- variabilele de mediu sunt moștenite de la shell-ul părinte (Bash) sau sunt definite în mini-shell;
- definirea variabilelor se face sub forma `NUME_VARIABILA=valoare`;
- nu trebuie tratat cazul în care `valoare` conține referiri la alte variabile de mediu.
- dacă variabila de mediu nu există, aceasta are valoarea șirul vid (**Atenție** șirul vid este diferit de NULL)

Precizări generale

- Pentru a simplifica implementarea temei, puteți folosi parserul implementat de noi. Pentru detalii despre parser, citiți fișierul `README` din arhivă. Exemple de utilizare a parserului găsiți în sursele `UseParser.cpp` și `CUseParser.c`.
- Vă recomandăm să începeți implementarea temei pornind de la cele două schelete de cod puse la dispoziție.
- Promptul afișat de shell este impus pentru a facilita testarea automată și este ">" (adică se va afișa caracterul `>` urmat de un spațiu).
- Numele executabilului temei trebuie să fie `mini-shell` pe Linux, respectiv `mini-shell.exe` pe Windows.
- Din cauza diferenței între Windows și Linux la crearea de noi procese (`CreateProcess` vs. `fork` + `exec`), s-ar putea să nu puteți folosi același tip de parcurgere a arborelui sintactic și pe Windows și pe Linux. Dacă vreți să reutilizați concepte/cod de pe Linux pe Windows, concepeți parcurgerea să funcționeze și cu funcția `CreateProcess` de pe Windows.
- Recomandăm rezolvarea și testarea din aproape în aproape a temei, după pași:
  - rularea de comenzi simple
  - rularea de comenzi interne (`cd`, `exit`, `quit`)
  - implementarea redirectărilor (operatorii `<`, `>`, `2>`, `&>`, `>>`, `2>>`)
  - variabile de mediu

- Laborator 06 - Memoria virtuală
- Laborator 07 - Profiling & Debugging
- Laborator 08 - Thread-uri Linux
- Laborator 09 - Thread-uri Windows
- Laborator 10 - Operații IO avansate - Windows
- Laborator 11 - Operații IO avansate - Linux
- Laborator 12 - Implementarea sistemelor de fișiere

Cursuri

- **Curs 01 - Introducere**
- **Curs 02 - Sistemul de fișiere**
- **Curs 03 - Procese**
- **Curs 04 - Planificarea execuției. IPC**
- **Curs 05 - Gestiunea memoriei**
- **Curs 06 - Memoria virtuală**
- **Curs 07 - Securitatea memoriei**
- **Curs 08 - Fire de execuție**
- **Curs 09 - Sincronizare**
- **Curs 10 - Dispozitive de intrare/ieșire**
- **Curs 11 - Networking în sistemul de operare**
- **Curs 12 - Implementarea sistemelor de fișiere**
- **Curs 13 - Securitatea sistemului**
- **Quizz-uri curs**
- Curs extra - Android
- Curs extra - Virtualizare
- Curs extra - Sincronizarea proceselor
- Note de curs

Teme

- **Tema Asistenți - Guardian process**
- Contestații
- Git. Indicații folosire GitLab

- secvențierea comenzilor (operatorii `&&`, `||`, `;`)
- implementarea operatorilor `&` (paralel) și `|` (pipe)

- Aveți mai jos câteva exemple de comenzi și rezultatul generat de acestea:

- Indicații generale teme
- Tema 1 Multi-platform Development
- Tema 2 Mini-shell
- Tema 3 Memorie virtuală
- Tema 4 Planificator de threaduri
- Tema 5 Server web asincron

Table of Contents

```
> ls
Makefile   README.checker  mini-shell mini-shell.o  parser
Makefile.checker  inputs   mini-shell.c outputs       tags

> uname -a ; ps
Linux bogdan-desktop 2.6.31-19-generic #56-Ubuntu SMP Thu Jan 28
  PID TTY          TIME CMD
 6078 pts/0    00:00:00 bash
 6190 pts/0    00:00:00 mini-shell
 6200 pts/0    00:00:00 ps

> date && sleep 1 ; echo date
Mon Feb  8 13:40:25 EET 2010
date

> date && sleep 1; date
Mon Feb  8 13:40:49 EET 2010
Mon Feb  8 13:40:50 EET 2010

> true && date
Mon Feb  8 13:41:16 EET 2010

> false && cat mini-shell.c

> false || date
Mon Feb  8 13:42:36 EET 2010

> cat /et/services
cat: /et/services: No such file or directory

> cta /etc/services
Execution failed for 'cta'

> 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

> strace -e trace=read ls 2> strace.out
Makefile   README.checker  mini-shell mini-shell.o  parser   tag
Makefile.checker  inputs   mini-shell.c outputs       strace.out

> head -1 strace.out
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0@#\0\0\0\

> pwd; cd Teme; pwd
/home/bogdan/Documents/SO/Solutii
/home/bogdan/Documents/SO/Solutii/Teme

> LETTER=alfa && echo $LETTER
alfa

> echo a > test ; echo b >> test && cat test
a
b

> exit
```

Precizări Windows

- Tema se va rezolva folosind doar funcții Win32. Se pot folosi, de asemenea, și funcțiile de formatare `printf`, `scanf`, funcțiile de alocare de memorie `malloc`, `free` și funcțiile de lucru cu șiruri de caractere (`strcat`, `strcmp`, etc.)
- Pentru partea de I/O și procese se vor folosi doar funcții Win32. De exemplu, funcțiile `fopen`, `fread`, `fwrite`, `fclose` nu trebuie folosite, în locul acestora trebuind să folosiți `CreateFile`, `ReadFile`, `WriteFile`, `CloseHandle`.
- Pentru a permite transmiterea de caractere speciale în argumente (spre exemplu `echo 'int main() { return 0; }'`) vă recomandăm să "îngrădiți" argumentele liniei de comandă a `CreateProcess` cu apostrofuri.
- Dacă folosiți Visual Studio dezactivați Unicode (clic dreapta pe proiect → Properties → Character Set → Not Set)

Pentru testarea temei cu testele publice se va folosi cygwin. Compilarea surselor se face cu compilatorul specific Windows, `cl` (în Visual Studio console).

Instalare Cygwin

- Puteți descărca Cygwin de aici.
- Asigurați-vă că la instalare ați selectat pachetele `make` și `gcc`.

Mașina virtuală de Windows pusă la dispoziție are deja Cygwin instalat.

Precizări Linux

- Tema se va rezolva folosind doar funcții POSIX. Se pot folosi de asemenea și funcțiile de formatare `printf`, `scanf`, funcțiile de alocare de memorie `malloc`, `free` și funcțiile de de lucru cu șiruri de caractere (`strcat`, `strdup`, etc.)
- Pentru partea de I/O și procese se vor folosi doar funcții POSIX. De exemplu, funcțiile `fopen`, `fread`, `fwrite`, `fclose` nu trebuie folosite, în locul acestor trebuind să folosiți `open`, `read`, `write`, `close`.

Testare

- Pentru simplificarea procesului de corectare a temelor, dar și pentru a reduce greșelile temelor trimise, corectarea se va realiza automat cu ajutorul testelor publice indicate în secțiunea de materiale ajutătoare.
- Există 18 teste. Se pot obține maxim 9.5 puncte prin trecerea testelor. Se acordă 0.5 puncte din oficiu.
- Pentru a trece testul 18, este obligatoriu să respectați formatul mesajului de eroare impus. Mesajul de eroare trebuie scris la `stderr` și trebuie să fie identic cu cel așteptat de teste (urmăriți `test_18_ref.txt` din teste).
- O temă care trece toate testele automate va obține 10 puncte din 10 (daca nu trișează folosind API interzis, cum ar fi funcția `system()`, caz în care nu va fi punctată).
- **Testul 0** din cadrul checker-ului temei verifică automat coding style-ul surselor voastre folosind stilul de coding din kernelul Linux. Acest test valorează **5 puncte** din totalul de 100. Pentru mai multe informații despre un cod de calitate citiți pagina de recomandări.
- Din punctajul temei se vor scădea automat puncte pentru întârzieri și pentru warning-uri. La revizia temei, se poate scădea suplimentar pentru nerespectarea criteriilor scrise la secțiunea de depunctări ale temelor. Astfel:
  - `-0.1` pentru fișier Makefile incorect (de exemplu compilează de fiecare dată totul)

- `-0.2` pentru fișier README necorespunzător
- `-0.2` surse necorespunzător comentate
- `-0.2` pentru neverificarea condițiilor de eroare
- `-0.2` pentru neeliberarea de resurse: nu se eliberează memoria alocată (*memory leaks*)
- `-0.2` pentru neeliberarea de resurse: nu se închid descriptorii de fișiere (Linux), respectiv handlerele (Windows) după utilizare
- `-0.2` diverse alte probleme constatate în implementare

- În cazuri excepționale se poate scădea mai mult decât este menționat mai sus.

Înainte de a uploada tema, asigurați-vă că implementarea voastră trece testele pe mașinile virtuale. Dacă apar probleme în rezultatele testelor, acestea se vor reproduce și pe vmchecker.

Pentru a inspecta diferențele între output-ul mini-shell-ului și cel al binarului de referință folosit de checker setați `DO_CLEANUP=no` în scriptul run_test.sh.

Una dintre depunctări este pentru leak-uri de memorie. În Linux pentru identificarea lor puteți folosi utilitarul valgrind. Vă puteți folosi de suportul checker-ului pentru a verifica leak-urile de memorie setând `USE_VALGRIND=yes` în scriptul run_test.sh. Pentru mai multe detalii cercetați conținutul fișierului README al checker-ului.

Materiale ajutătoare

Cursuri utile:

- Curs 1
- Curs 2
- Curs 3

Laboratoare utile:

- Laborator 1
- Laborator 2
- Laborator 3

Resurse:

- Parserul pentru comenzi, scheletele de cod și testele sunt disponibile în directorul 2-minishell din repo-ul de pe Github.

Pagina de Upload:

- vmchecker

Repo-ul de pe Github conține și un script Bash care vă ajută să vă creați un repository privat pe instanța de Gitlab a facultății, unde aveți la dispoziție 5 repository-uri private utile pentru teme. Urmăriți indicațiile din README și de pe wiki-ul SO.

În plus, responsabilii de teme se pot uita mai rapid pe Gitlab la temele voastre pentru a vă ajuta în cazul în care întâmpinați probleme/bug-uri. Este mai ușor să primiți suport în rezolvarea problemelor implementării voastre dacă le oferiți responsabililor de teme acces la codul sursă pe Gitlab.

Dacă ați folosit Gitlab pentru realizarea temei, indicați în README link-ul către repository. Asigurați-vă că responsabilii de teme au drepturi de citire asupra repo-

ului vostru.

FAQ

- **Q:** Tema 2 se poate face în C++?
  - **A:** Nu.
- **Q:** Cum pot să citesc arhivele listei de discuții?
  - **A:** O variantă este cu Opera: File → Import and export → Import mail → Import generic mbox file → alegeți fișierele respective.
- **Q:** Am voie sa folosesc funcții POSIX pe Windows?
  - **A:** La toate temele de SO pe Windows se va folosi Win32 API, nu POSIX. Oricum, `fork` și `exec` nu sunt suportate pe Windows 7 decât după instalarea Interix/SUA.
- **Q:** La temele de Windows trebuie să folosesc funcțiile de API în versiunea Unicode, ANSI, sau generică?
  - **A:** Nu este impus să folosiți o versiune anume. Aveți, totuși, grijă să folosiți corect funcțiile (acestea primesc parametri de tip CHAR pentru versiunea ANSI, WCHAR pentru Unicode și TCHAR în versiunea generică). Pentru detalii consultați Unicode in the Windows API.
- **Q:** Ce fac dacă am intâlnit un caz limită al carui comportament nu este precizat în enunț?
  - **A:** În general la temele de SO, pentru cazuri limită ce nu apar în testele publice sau în enunt, se acceptă orice comportament documentat în README. Un exemplu este comportamentul pentru "command | cd /something". Daca nu sunteți siguri, întrebați pe lista de discuții.
- **Q:** Trebuie optimizat numărul de fork-uri? Spre exemplu, în cazul comenzii `a|b|c` trebuie sa am 3 forkuri sau pot să am 4 sau 5?
  - **A:** Nu este obligatoriu să optimizați numărul de fork-uri. Totuși, în general este bine să aveți în vedere eficientizarea consumului de resurse.
- **Q:** Shell-ul trebuie să se comporte ca un shell adevărat (sh, bash) în situația ... ?
  - **A:** Funcționalitatea minimă necesară este cea din enunțul temei. Daca implementați ceva în plus, precizați în README. Exemple de funcționalitate care nu este cerută: updatarea unor variabile de mediu (gen `$OLDPWD` și `$PWD`), history, multe altele ... (vezi man bash pentru o idee despre funcționalitatea unui shell complet)
- **Q:** Am voie să nu folosesc parserul din enunț dacă doresc să scriu eu altul echivalent?
  - **A:** Da.
- **Q:** Avem voie sa folosim:

```
const char *argv[] = {"/bin/bash", "-c", command, NULL};
execv("/bin/bash", (char *const *)argv);
```

  - **A:** Nu.
- **Q:** Am voie să fac `execv` pe tema mea pentru a executa o parte din arbore independent?
  - **A:** Nu.
- **Q:** Am întâmpinat o problemă la testul X. Local testul este *passed*, însă pe vmchecker testul este *failed*. Ce este de făcut?
  - **A:** Execută testele în mașina virtuală. Problema va apărea și acolo și o vei rezolva depanând (aka *debugging*) programul tău folosind gdb și valgrind în mașina virtuală.

Suport, întrebări și clarificări

Pentru întrebări sau nelămuriri legate de temă folosiți lista de discuții sau

canalul de IRC.