Tema 1 Multi-platform Development

- Dată publicare: **28.02.2019**
- **Deadline**: **13.03.2019, ora 23:55**
- **Deadline hard**: **20.03.2019, ora 23:55**

Scopul temei

- Recapitularea lucrului cu funcțiile din biblioteca standard C:
  - lucrul cu fișiere
  - alocare dinamică de memorie
  - folosirea pointerilor
- Încapsularea datelor într-o formă abstractă pentru o structură de date
- Folosirea bibliotecilor dinamice (și linkarea cu acestea)
- Realizarea unui Makefile pentru platformele Linux (folosind `gcc`) și Windows (folosind `cl`)

Dezvoltarea temei

Dezvoltarea trebuie făcută exclusiv pe mașinile virtuale SO.

Nu rulați testele *"local"* (pe calculatoarele voastre sau în mașinile voastre virtuale). Veți avea diferențe față de vmchecker, iar echipa de SO nu va depana testele care merg "local", dar pe vmchecker nu merg. Pe vmchecker sunt aceleași mașinile virtuale ca cele de pe wiki.

Este încurajat ca lucrul la tema să se desfășoare folosind `git`. Indicați în README link-ul către repository dacă ați folosit git. **Asigurați-vă că responsabilii de teme au drepturi de citire asupra repo-ului vostru**.

Ca să vă creați un repo de gitlab în instanța facultății: în repository-ul so-assignments [https://github.com/systems-cs-pub-ro/so-assignments] de pe Github se află 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.

Motivul pentru care încurajăm acest lucru este că responsabilii de teme se pot uita mai rapid pe Gitlab [https://gitlab.cs.pub.ro] 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 [https://gitlab.cs.pub.ro].

Crash-course practic de git puteți găsi aici: git-immersion [http://gitimmersion.com/lab_01.html#main_content]

Atât semnătura pentru funcția de comparare (veți vedea mai jos pentru ce vă trebuie), cât și testele publice care se rulează pe vmchecker se găsesc pe repo-ul so-assignments de pe Github [https://github.com/systems-cs-pub-ro/so-assignments/tree/master/1-multi/]:

```
student@so:~$ git clone https://github.com/systems-cs-pub-ro/so-assignments.git
student@so:~$ cd so-assignments/1-multi
```

În repository-ul de pe Github se vor găsi și scheletele pentru temele viitoare, care vor fi actualizate și se vor putea descărca pe viitor folosind comanda:

```
student@so:~$ git pull
```

Tot prin comanda de mai sus se pot obține toate actualizările făcute în cadrul temei 1.

Enunț

Să se implementeze **în C** o coadă de priorități ce va conține cuvinte. Operațiile ce trebuie implementate pentru coadă sunt următoarele:

Operația	Descrierea operație

insert <cuvânt> <prioritate>	adaugă cuvântul în coadă cu prioritatea respectivă
top	întoarce primul cuvânt din coadă - cel cu prioritate maximă
pop	elimină primul element din coadă

Aceste comenzi se vor regăsi una per linie.

Programul poate primi o serie de argumente din linia de comandă reprezentând fișiere de intrare din care se face citirea. Dacă nu există niciun argument citirea se face de la intrarea standard (`STDIN`). Afișarea se va face la ieșirea standard (`STDOUT`)

Atenție: în cazul în care sunt specificate mai multe fișiere de intrare, toate operațiile se aplică **aceleiași** cozi de priorități. Ordinea în care se aplică operațiile este dată de ordinea de transmitere a fișierelor din linia de comandă. Dacă un fișier **nu există**, acesta va fi **ignorat** (se trece la urmatorul fișier).

Exemplu:

```
# Citire din fișiere:

student@so-vm:~$ cat pq1.in
# Unde pq1.in poate conține:
insert ozy 30
insert mishu 50
top
pop
top
insert oreo 50
top
pop

student@so-vm:~$ ./tema1 pq1.in pq2.in pq3.in

# Citire de la stdin:
student@so-vm:~$ ./tema1 < pq.in
```

Format de iesire este câte un rezultat pe linie:

```
student@so-vm:~$ cat  pq.in
insert ana 20
insert simona 42
top
pop
inserare
top
pop
topor
top
insert boss 30
popor
comanda_care_nu_exista
sudo comanda
top
student@so-vm:~$ ./tema1 < pq.in
student@so-vm:~$
simona
ana
boss
```

Pentru primul exemplu, se vor aplica toate operațiile din fișierul *pq1.in*, după care *pq2.in* și *pq3.in*. în cazul în care *pq2.in* nu există, se va încerca deschiderea și citirea comenzilor din fișierul *pq3.in*.

Citirea de la `stdin` se face până la întâlnirea caracterului `EOF` (CTRL+D)

Coada de priorități va trebui să folosească în implementare funcția *compare* (header-ul funcției este pus pe Github). Acestă funcție întoarce:

- *< 0*: dacă primul parametru este mai mic decât al doilea
- *== 0*: dacă parametrii sunt egali
- *> 0*: dacă primul parametru este mai mare decât al doilea

Coada de priorități o puteți implementa cum doriți, atât timp cât țineți cont de următoarele precizări.

Precizări generale

Indicațiile și precizările generale pentru teme sunt valabile și aici. Vă rugăm să le parcurgeți și să țineți cont de ele înainte de a vă apuca de temă și respectiv înainte de submisia finală.

- Valorile introduse în coada de priorități sunt cuvinte urmând regex-ul `[a-z]+`, iar prioritățile vor putea fi memorate pe 4 octeți.
- Cuvintele inserate, cât și prioritățile sunt **unice**.

În general valorile priorităților într-o astfel de coadă nu sunt unice, gândiți-vă la algoritmii de planificare a proceselor într-un sistem în care 80% din procese au aceeași prioritate. Această restricție e impusă deoarce dorim să vă facem implementarea și testarea ușoare. Apreciem implementările care țin cont de priorități care nu sunt unice (și documentate cu teste)

- Antetul funcției de compare ce **trebuie** folosită (în întreaga temă) este declarat în **`compare.h`**. Definiția funcției se găsește într-o bibliotecă dinamică (`libcompare.so` pentru Linux, respectiv `compare.dll` pentru Windows).

Va trebui să linkați în tema voastră aceste biblioteci. Este interzisă folosirea unei alte funcții de comparare scrisă prin surse proprii. Nu includeți bibliotecile în arhiva finală a temei. IMPORTANT: bibliotecile au fost compilate folosind mașinile virtuale, care sunt cu arhitectura pe 64-bit - mașina virtuala de SO:

```
student@so-vm:~$ uname -a
Linux vagrant 4.15.0-29-generic #31-Ubuntu SMP Tue Jul 17 15:39:52 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
```

- Dacă dezvoltați pe mașina locală e posibil să vă apară erori de incompatibiltate.

Nu submiteți vreo sursă a `compare.c` și vă bazați implementarea pe acesta. Veți fi depunctați (atât de checker, cât și la corectură dacă checkerul are false positive).

Makefile-ul pentru Windows trebuie să compileze sursele utilizând flag-ul **/MD**.

- Pentru Windows, compilarea se va realiza din PowerShell, iar rularea se va face folosind Cygwin.
- Programul trebuie să execute comenzile în ordine, așa cum au fost primite sau citite din fișier(e).
- Liniile goale din fișierul de intrare trebuie ignorate (programul nu face nimic și trece la linia următoare)
- Funcțiile care manipulează coada de priorități trebuie să funcționeze pe o coadă de priorități primită ca parametru, nu direct pe una care este definită ca variabilă globală.
- Executabilul generat va purta numele **`tema1`** pe Linux și **`tema1.exe`** pe Windows.
- Dimensiunea maximă a unei comenzi este de **20000 de caractere**.
- Buffer-ul folosit pentru citirea comenzilor poate fi declarat cu dimensiune statică.
- Coada de priorități **NU** poate fi implementată folosind vectori alocați static.
- **Verificați valorile întoarse de funcțiile `malloc/calloc/realloc` (în funcție de implementarea aleasă)**. În cazul în care una dintre aceste funcții eșuează, trebuie întors codul de eroare `12` (este codul de eroare pentru `ENOMEM`). Acest cod de eroare trebuie propagat și returnat până la ieșirea din program. Valoarea erorii este pozitivă.
  - exemplu: din `main` se apelează `f1`, iar `f1` apelează `f2`: dacă eroarea apare în momentul apelului unui `malloc` în funcția `f2`, atunci codul de eroare (valorea `12`) va fi întors în `f1`, din `f1` va trebui întors tot `12`, iar din main se va ieși cu același cod de eroare.
- Operațiile de *insert, top, pop* care nu respectă formatul specificat vor fi ignorate.
- Operația *top* apelată pe o coadă fără elemente va trebui să afișeze un newline
- De asemenea, operațiile care nu există vor fi ignorate (e.g. `add gigel`).

Precizări VMChecker

Arhiva temei va fi încărcată de două ori pe vmchecker [https://vmchecker.cs.pub.ro/ui/#SO] (Linux și Windows). Arhiva trimisă trebuie să fie aceeași pe ambele platforme (se vor compara cele două arhive trimise).

Insistăm, dacă mesajul cu roșu nu a fost clar: arhiva care se trimite pe vmchecker trebuie să fie **identică** pe ambele platforme. Puteți folosi `md5sum` sau `sha1sum` (sau comenzi similare) asupra arhivelor voastre dacă ați dezvoltat în locuri diferite.

Temele trimise pe o singură platformă sau cu arhive diferite nu vor fi punctate și vor fi notate cu 0.

Arhivele trebuie să conțină sursele temei, `README` și două fișiere Makefile care conțin target-urile `build` și `clean`:

- **Linux:** Fișierul Makefile se va numi `GNUmakefile`.
  - **ATENȚIE**: GNUmakefile (cu `m` mic).
- **Windows:** Fișierul Makefile se va numi `Makefile`.
- Regula de `build` trebuie să fie cea principală (executată atunci când se dă `make` fără parametrii)

Executabilul rezultat din operația de compilare și linking se va numi `tema1` pe Linux și `tema1.exe` pe Windows.

Nu e nevoie să includeți bibliotecile dinamice în arhivă.

Punctare

- Tema va fi punctată cu minimul punctajelor obținute pe cele două platforme. Nu aveți voie să folosiți directive de preprocesare de forma:

```
#ifdef __linux__
[...]
#ifdef _WIN32
[...]
```

Cu alte cuvinte: exact același cod trebuie să ruleze pe ambele platforme. Tema **NU TREBUIE** să conțină surse specifice unuia sau altuia dintre sistemele de operare Linux/Windows. Veți avea două fișiere makefile (`GNUmakefile` pentru Linux și `Makefile` pentru Windows, cum e precizat mai sus) iar checker-ul va ști, în funcție de sistemul lui de operare, ce makefile să folosească.

Nota mai poate fi modificată prin depunctări suplimentare:

- Lista generală de depunctări
- `-1` folosirea unei funcții proprii de comparare, alta decât cea declarată în header-ul `compare.h` din schelet și a cărei definiție se găsește în `libcompare.so` și `compare.dll`. Este deja test pe vmchecker [https://vmchecker.cs.pub.ro/ui/#SO], însă în cazul în care considerăm că a trecut din cauza unui bug în vmchecker [https://vmchecker.cs.pub.ro/ui/#SO] aplicăm manual depunctarea
- `-2` implementare netransparentă a cozii de priorități; coada de priorități ar trebui să fie abstractizată cu un singur obiect (în C: structură de date), iar operațiile pe coadă trebuie făcute pe obiectul respectiv. Puteți folosi definiții proprii pentru elementele cozii.
- `-4` alocare statică coadă de priorități
- se pot scădea oricâte puncte pentru teme care conțin erori grave/vizibile de coding style sau de funcționare care pot să nu fie pe lista generală de depunctări

Testul 0 din cadrul checker-ului temei verifică automat coding style-ul surselor voastre folosind stilul de coding din kernelul Linux [https://www.kernel.org/doc/Documentation/process/coding-style.rst]. Acest test valorează **5 puncte** din totalul de 100. Pentru mai multe informații despre un cod de calitate citiți pagina de recomandări.

Pentru investigarea problemelor de tip *Segmentation Fault* sau comportament incorect al aplicației la unul din teste, pentru debugging, se recomandă folosirea gdb în Linux.

Una dintre depunctări este pentru leak-uri de memorie [https://en.wikipedia.org/wiki/Memory_leak]. În Linux pentru identificarea lor puteți folosi utilitarul valgrind [http://valgrind.org/docs/manual/quick-start.html].

Pentru instalarea `gdb` și `valgrind`, pe o distribuție Ubuntu se poate folosi comanda:

```
student@so:~$ sudo apt-get install gdb valgrind
```

Pentru debugging și detectarea leak-urilor de memorie este necesar să ștergeți toate optimizările de la flag-urile de compilare (e.g. `-O3`) și trebuie să compilați doar cu flag-urile `-Wall -g` (sau cele care mai activează alte warning-uri, e.g. `-Wextra`).

Nu trebuie la fiecare eroare considerată fatală să eliberați fiecare pointer alocat dinamic. În cadrul corecturii temei principala verificare pentru memory leaks va fi pe o funcționare corecta/normală, fără input invalid. Rețineți că memory leak-ul apare atunci când programul vostru nu poate returna sistemului de operare memoria folosită! Concentrați-vă pe folosirea `valgrind` pe teste care trec și care dau input valid într-o primă fază.

Precizări Makefile

Makefile-ul trebuie să respecte următoarea structură: pentru fiecare fișier `.c` generat trebuie să se obțină un fișier obiect. La final trebuie să faceți linkarea între sursa principală (să zicem `main.c` din care se obține `main.o`), celelalte fișiere obiect obținute din celelalte surse ale voastre și biblioteca `libcompare.so` pe Linux, respectiv `compare.dll` pe Windows.

Porniți de la exemplele de Makefile atât pentru Linux cât și pentru Windows oferite în laboratorul 1. Un alt exemplu puteți găsi aici [https://www.gnu.org/software/make/manual/html_node/Simple-Makefile.html].

Nu uitați: Makefile-ul pentru Windows trebuie să compileze toate sursele voastre utilizând flag-ul **/MD**.

Denumirile lor trebuie să fie:

- **Linux:** Fișierul Makefile se va numi `GNUmakefile`.
  - **ATENȚIE**: GNUmakefile (cu `m` mic).
- **Windows:** Fișierul Makefile se va numi `Makefile`.

Resurse necesare realizării temei

Semnătura funcției de comparație (compare.h [https://github.com/systems-cs-pub-ro/so-assignments/blob/master/1-multi/util/compare.h]) este:

```
#ifndef __COMPARE_H_
#define __COMPARE_H_

/**
  * Function to compare two integer values (priorities) for inserting
  * in a priority queue
  * @param prio1 - first priority (first value)
  * @param prio2 - second priority (second value)
  * @return Value higher than 0 if prio1 > prio2,
  *              lower than 0 if prio1 < prio2
  *              equal to 0 if prio1 == prio2
  */
int compare(int prio1, int prio2);

#endif
```

Biblioteca ce conține funcția de compare (pre-compilată pentru mașinile virtuale de Linux-64bit și Windows 7 în format DLL) dar și întregul checker se găsesc pe github-ul SO [https://github.com/systems-cs-pub-ro/so-assignments/tree/master/1-multi]

Pentru a clona repo-ul și a accesa resursele temei 1:

```
student@so-vm:~$ git clone https://github.com/systems-cs-pub-ro/so-assignments.git
student@so-vm:~$ cd so-assignments
student@so-vm:~/so-assignments$ cd 1-multi
```

- Referințe utile:
  - ANSI C reference [https://en.cppreference.com/w/c]
  - Data Structures Visualization [http://www.cs.usfca.edu/~galles/visualization/Algorithms.html]

FAQ

- **Q:** Tema 1 se poate face în C++?
  - **A:** Nu.
- **Q:** "Valorile introduse în coada de priorități sunt cuvinte `[a-z]+`" - trebuie verificate cuvintele la introducere?
  - **A:** Nu.
- **Q:** Se pot folosi directive de preprocesare de tipul `#define`?
  - **A:** Da. Singurele directive de preprocesare interzise sunt cele care introduc cod condițional în funcție de OS-ul folosit (e.g. `ifdef __linux__`)
- **Q:** Pentru citire/scriere din fișier/consolă putem folosi `freopen`?
  - **A:** Da, e ok. Puteți folosi orice funcție din categoria `fopen`, `fread`, `fwrite`, `fclose`.
- **Q:** Se poate folosi `realloc`?
  - **A:** Da.
- **Q:** Se pot folosi funcțiile `fgets`, `fscanf`, `printf`, `fprintf`?
  - **A:** Da. Atenție să nu folosiți `gets`!
- **Q:** Pe Windows, folosind `cl.exe` nu mi se compilează același cod care mi se compila pe Linux. De ce?
  - **A:** Cel mai probabil cauza este următoarea: pe Linux este folosit `C99` ca standard la `gcc`, care printre altele acceptă să declari variabile în mijlocul codului. Pe Windows, compilatorul `cl` folosește standardul `C89`, care forțează declararea variabilelor **doar** la început (un exemplu de problema).
- **Q:** Văd că pentru coding style iau 0, ce pot face în această situație?
  - **A:** Descărcați cu `wget` checkpatch.pl de aici [https://raw.githubusercontent.com/torvalds/linux/master/scripts/checkpatch.pl], îl puneți în `PATH` și apoi rulați checker-ul de Linux (pașii sunt mai jos). Alternativ, vă puteți folosi de acest wrapper [https://raw.githubusercontent.com/systems-cs-pub-ro/so-assignments/master/checkpatch_wrapper.sh] peste checkpatch.pl a verifica sursele folosind criteriile considerate în evaluarea temelor.

```
student@so:~$ wget https://raw.githubusercontent.com/torvalds/linux/master/scripts/checkpatch.pl
student@so:~$ export PATH=$PATH:/path/to/dir/with-checkpatch
student@so:~$ cd /path/to/lin/checker && ./run_all.sh
```

Suport, întrebări și clarificări

Pentru întrebări sau nelămuriri legate de temă folosiți lista de discuții sau canalul de IRC.

Orice intrebare pe mailing list e recomandat să aibă subiectul de forma `[Tema1][Platforma] Titlul problemei`. Exemple de așa da:

- [Tema1][Linux] Memory leaks detected, desi am facut free
- [Tema1][Windows] No makefile found
- [Tema1][General] Neclaritate enunt: functie compare

Exemple de așa nu:

- Problema la tema 1
- eroare tema 1
- eroare la compare

Evident, și în cel de-al doilea caz veți primi răspunsuri, dar e posibil să le primiți mai greu. În conținutul emailului, în caz de probleme mai specifice dați cât mai multe detalii despre ce ați încercat, mesaje de eroare, faceți attach la log-uri de execuție, output-uri de comenzi respectiv ce comenzi ați rulat etc.

Revedeți și secțiunea de guidelines pentru lista de discuții SO

so/teme/tema-1.txt · Last modified: 2019/03/14 12:41 by luca.istrate