# Tema 1: Sistemul de fișiere
# Varianta: 45784
# Student: Bogdan Bud

## 1 Descrierea temei

Se dă următorul format de fișier binar, pe care îl vom numi în continuare formatul **SF** (i.e. "**section file**"). Un SF este compus din două părți: **header** și **body**. Structura generală a unui SF este ilustrată mai jos. Se poate observa că **header**-ul este situat la începutul fișierului, înainte **body**.

SF FILE STRUCTURE

```
HEADER
BODY
```

Header-ul unui SF conține informații care identifică formatul acestuia și deasemenea descrie modul în care body-ul trebuie citit. Aceste informații sunt organizate ca o secvență de câmpuri, fiecare câmp având un anumit număr de octeți și o anumită semnificație. Structura header-ului este ilustrată în caseta HEADER de mai jos, specificând pentru fiecare câmp numele acestuia și numărul de octeți pe care îl ocupă (separate prin ": "). Unele câmpuri sunt simple numere (ex. MAGIC, HEADER_SIZE, VERSION și NR_OF_SECTIONS), în timp ce altele (i.e. SECTION_HEADERS și SECTION_HEADER) au o structură mai complexă, având propriile sub-câmpuri. Astfel, SECTION_HEADERS este compus dintr-o secvență de elemente de tip SECTION_HEADER, fiecare dintre acestea având la rândul lui sub-câmpurile: SECT_NAME, SECT_TYPE, SECT_OFFSET și SECT_SIZE.

HEADER

```
MAGIC: 2
HEADER_SIZE: 2
VERSION: 4
NO_OF_SECTIONS: 1
SECTION_HEADERS: NO_OF_SECTIONS * sizeof(SECTION_HEADER)
    SECTION_HEADER: 7 + 1 + 4 + 4
        SECT_NAME: 7
        SECT_TYPE: 1
        SECT_OFFSET: 4
        SECT_SIZE: 4
```

Semnificația fiecărui câmp este următoarea:

- Câmpul MAGIC identifică fișierele SF. Valoarea acestuia este specificată mai jos.
- Câmpul HEADER_SIZE indică dimensiunea header-ului fișierului SF.
- Câmpul VERSION identifică versiunea formatului SF, presupunând că acesta se poate schimba de la o versiune la alta (deși nu este cazul la această temă).
- Câmpul NO_OF_SECTIONS specifică numărul de elemente de tip SECTION_HEADER, care vor fi acoperite mai jos.
- Sub-câmpurile unui SECTION_HEADER fie au nume explicite, fie sunt descrise mai jos.

Body-ul unui SF, practic o secvență de octeți, este organizat ca o colecție de secțiuni. O secțiune e formată dintr-o secvență de SECT_SIZE octeți consecutivi, începând de la octetul SECT_OFFSET, în cadrul fișierului. Secțiunile consecutive nu vor fi neapărat una după alta. Cu alte cuvinte, între două secțiuni consecutive pot exista octeți ce nu aparțin nici unei secțiuni. O secțiune dintr-un SF conține caractere afișabile și caractere de final de linie. Se poate spune că sunt de fapt secțiuni de tip text. Octeți dintre secțiuni pot avea orice valoare, dar aceștia nu au nici o relevanță în interpretarea conținutului unui SF. Caseta de mai jos ilustrează două secțiuni și header-ele acestora într-un SF posibil.

PARTIAL SF FILE's HEADER AND BODY

```
Offset  Bytes
0000:   ...
...     ...
xxxx:   SECT_1 TYPE_1 1000 10
yyyy:   SECT_2 TYPE_2 2000 10
...     ...
1000:   1234567890
...     ...
...     ...
2000:   1234567890
...     ...
```

Următorele restricții se aplică anumitor câmpuri din SF:

- Valoarea câmpului MAGIC este "0t".
- Valoarea câmpului VERSION e un număr între 73 și 193, inclusiv.
- Câmpul NO_OF_SECTIONS are valoarea între 2 și 14, inclusiv.
- Valorile valide pentru SECT_TYPE sunt: 18 37 21 14 65 70 96 .
- Liniile dintr-o secțiune sunt separate de următoarea secvență de octeți (valorile sunt în hexazecimal): `0D 0A`.

# 2 Cerințele temei

Trebuie să scrieți un program C, numit "*a1.c*" care implementează cerințele ce urmează.

## 2.1 Compilarea și rularea

Programul trebuie să poată fi **compilat fără erori**, pentru a fi acceptat. Comanda de compilare va fi cea de mai jos:

Compilation Command

```
gcc -Wall a1.c -o a1
```

Warning-uri în procesul de compilare vor aduce o penalizare de 10% din scorul final.

Atunci când rulează, executabilul vostru (îl vom numi "*a1*") **trebuie să ofere funcționalitatea minimală și rezultatele așteptate**, pentru a fi acceptat. Vom defini mai jos ce înseamnă funcționalitatea minimală. Următoarea casetă ilustrează modul în care programul va fi rulat. Opțiunile și parametrii pe care programul trebuie să îi accepte, împreună cu semnificația acestora se vor detalia în următoarele secțiuni.

Running Command

```
./a1 [OPTIONS] [PARAMETERS]
```

## 2.2 Afișarea variantei

Fiecare student primește o variantă puțin modificată a formatului SF și a cerințelor. Prima sarcină din temă va fi afișarea variantei de temă primită, atunci când programul este rulat precum mai jos.

Display Variant Command

```
./a1 variant
```

Output Sample for Display Variant Command

```
45784
```

## 2.3 Afișarea conținutului folderelor

Atunci când se dă opțiunea "`list`", programul trebuie să afișeze pe ecran numele anumitor elemente (ex. fișiere, subfoldere) din folderul a cărui cale este specificată prin opțiunea "`path`", conform exemplului din caseta de mai jos.

List Directory Command

```
./a1 list [recursive] <filtering_options> path=<dir_path>
```

Opțiunile din linia de comandă se pot da în orice ordine, de exemplu opțiunea "recursive" poate apărea înainte sau după alte opțiuni.

Elementele care trebuie afișate vor fi determinate pe baza criteriilor de filtrare menționate în continuare. Fiecare nume de element trebuie afișat pe o linie separată, fără linii libere între nume, în forma ilustrată în caseta de mai jos (numele fiecărui element trebuie să înceapă cu calea acestuia, specificată prin opțiunea "path"). Dacă nu se găsește nici un element care să corespundă criteriilor de căutare, nu trebuie afișat decât string-ul "SUCCESS".

Sample Output for List Directory Command (Success Case)

```
SUCCESS
test_root/test_dir/file_name_1
test_root/test_dir/file_name_2
test_root/test_dir/file_name_3
...
```

Dacă se folosește opțiunea "`recursive`", programul vostru trebuie să traverseze întregul sub-arbore, începând din orice folder dat, intrând recursiv în toate sub-folderele. Elementele găsite trebuie de asemenea să conțină calea cu tot cu directorul specificat prin opțiunea "path". Cele două casete de mai jos arată cum se poate rula programul cu opțiunea "`recursive`" și un output posibil.

Sample List Directory Command

```
./a1 list recursive path=test_root/test_dir/
```

Sample Output for List Directory Command (Success Case)

```
SUCCESS
test_root/test_dir/file_name_1
test_root/test_dir/file_name_2
test_root/test_dir/subdir_1/file_name_1
test_root/test_dir/subdir_1/subdir_1_1/file_name_2
test_root/test_dir/subdir_2/file_name_3
...
```

În cazul în care se întâlnește o eroare, trebuie afișat un mesaj de eorare, urmat de o explicație, conform casetei următoare.

Output for List Directory Command (Error Case)

```
ERROR
invalid directory path
```

Criteriile de filtrare, utilizate pentru a selecta elementele ce vor fi afișate sunt următoarele:

- Numele elementului trebuie să se termine cu secvența de caractere specificată prin opțiunea "`name_ends_with`=string". Trebuie considerate toate tipurile de elemente, atât fișirele cât și folderele.
- Elementele afișate trebuie să aibă permisiunea de scriere pentru proprietar (eng. *owner*), dacă se specifică opțiunea "`has_perm_write`". Trebuie considerate toate tipurile de elemente, atât fișirele cât și folderele.

Deși în practică se pot folosi mai multe criterii de filtrare în aceeași comandă, testele noastre folosesc un singur criteriu la un moment dat. Programul vostru ar trebui să trateze cazul general, dar aceasta nu e o cerință obligatorie.

Ordinea numelor de elemente afișate nu este importantă în aceste teste, doar numărul și valoarea acestora.

## 2.4 Identificarea și parsarea fișierelor SF

Atunci când se folosește opțiunea "`parse`", programul vostru trebuie să verifice dacă fișierul a cărui cale a fost specificată prin opțiunea "`path`" respectă sau nu formatul SF. În următoarea casetă se ilustrează modul de rulare a programului.

Check SF Format Command

```
./a1 parse path=<file_path>
```

Argumentele se pot da în orice ordine.

Validarea formatului SF se face după următoarele reguli:

- Valoarea câmpului MAGIC este cea menționată mai sus, adică "0t".
- Valoarea câmpului VERSION e un număr din intervalul menționat mai sus, adică între 73 și 193, inclusiv.
- Numărul de secțiuni trebuie să fie între 2 și 14, inclusiv.
- Tipurile secțiunilor din fișier trebui să fie dintre următoarele valori: 18 37 21 14 65 70 96 .

Atunci când toate criteriile de mai sus sunt îndeplinite, fișierul se poate considera valid, chiar dacă există alte inconsistențe (exemple: dimensiunea fișierului e prea mică sau secțiunile se suprapun).

În caz că fișierul dat nu este valid, un mesaj de eroare trebuie afișat pe ecran, urmat de primul motiv pentru care fișierul este invalid, conform casetei de mai jos.

Sample Output for an Invalid SF File

```
ERROR
wrong magic|version|sect_nr|sect_types
```

În caz că fișierul dat este valid (respectă formatul SF), anumite câmpuri trebuie afișate pe ecran, în forma ilustrată.

Sample Output for a Valid SF File

```
SUCCESS
version=<version_number>
nr_sections=<no_of_sections>
section1: <NAME_1> <TYPE_1> <SIZE_1>
section2: <NAME_2> <TYPE_2> <SIZE_2>
...
```

## 2.5 Lucrul cu secțiuni

Atunci când se folosește opțiunea "`extract`", programul vostru trebuie să caute și să afișeze o anumită porțiune a fișierului SF. În mod particular, o singură linie trebuie să fie extrasă. Argumentele din linia de comandă necesare sunt "`path`", "`section`" și "`line`", specificând calea spre fișier, numărul secțiunii, respectiv numărul liniei, conform casetei de mai jos.

Extract Section Line Command

```
./a1 extract path=<file_path> section=<sect_nr> line=<line_nr>
```

În cazul în care apare o eroare, mesajul de eroare trebuie afișat, urmat de motivul erorii, ca și în caseta de mai jos.

Sample Output for Extract Section Line Command (Error Case)

```
ERROR
invalid file|section|line
```

În cazul în care fișierul dat respectă formatul SF iar secțiunea și linia căutată există, rezultatul trebuie afișat precum în caseta de mai jos.

Sample Output for Extract Section Line Command (Success Case)

```
SUCCESS
<line_content>
```

Liniile se numără de la finalul secțiunii, ultima linie având numărul 1.

### 2.6 Filtrarea după secțiuni

Atunci când se dă opțiunea "`findall`", programul vostru trebuie să funcționeze ca și în cazul opțiunilor "`list recursive`", dar se vor afișa doar fișierele SF valide, care nu au nici o secțiune cu dimensiunea mai mare decât 965.

Modul de rulare a programului este ilustrat mai jos:

Find Certain SF Files Command

```
./a1 findall path=<dir_path>
```

În caz că se întâmpină erori, se va afișa un mesaj de eroare, urmat de o explicație, în forma de mai jos:

Output for Find Certain SF Files Command (Error Case)

```
ERROR
invalid directory path
```

Output-ul normal va începe cu linia "SUCCESS", urmată de fișierele găsite, câte unul pe linie:

Sample Output for Find Certain SF Files Command (Success Case)

```
SUCCESS
test_root/test_dir/file_name_1
test_root/test_dir/file_name_2
...
```

## 3 Manual de utilizare

### 3.1 Auto-evaluare

Pentru a genera și rula testele pentru soluția voastră, trebuie rulat script-ul "tester.py", furnizat împreună cu cerințele. Script-ul are nevoie de Python 2.x, nu de Python 3.x.

Sample Command for Tests Generation

```
python tester.py
```

Chiar dacă script-ul funcționează și pe Windows (și pe alte sisteme de operare), recomandăm rularea pe Linux, deoarece așa se va evalua tema voastră.

Atunci când se rulează script-ul, se va genera un folder numit "test_root", ce conține diverse sub-foldere și fișiere, pe care se va testa soluția voastră. Chiar dacă conținutul lui "test_root" este aleator și diferit pentru fiecare student, generarea acestuia se face în mod determinist, indiferent de timpul rulării script-ului. Pot totuși apărea diferențe dacă se rulează pe alte sisteme de operare.

După crearea folderului "test_root" și a conținutului acestuia, script-ul va rula de asemenea programul vostru, afișând rezultatul diferitelor teste. În mod normal vor fi 65 de teste, deși pot fi și mai puține. Nota maximă (10) se obține atunci când trec toate testele, fără nici o penalizare (vezi mai jos). Nota exactă se calculează scalând numărul de teste trecute în intervalul 0-10.

Restricții:

- Sunteți limitați la a folosi doar apeluri de sistem către sistemul de operare (funcții low-level). De exemplu, TREBUIE folosi funcțiile `open()`, `read()`, `write()` etc., nu funcțiile de nivel înalt `fopen()`, `fgets()`, `fscanf()`, `fprintf()` etc. Singurele excepții sunt citirea de la `STDIN` și afișarea în `STDOUT` / `STDERR`, cu funcții precum `scanf()`, `printf()`, `perror()`, respectiv funcții de manipulare a string-urilor cum ar fi `sscanf()`, `snprintf()`.

Recomandări:

- Pentru tokenizarea string-urilor (separarea după elemente specifice, cum ar fi spații) recomandăm utilizarea funcțiilor `strtok()` sau `strtok_r()`.

## 3.2 Evaluare și cerințe minime

- Dacă apar warning-uri la compilare, se va aplica o penalizare de 10%.
- Dacă programul vostru are memory leak-uri (memorie alocată și ne-eliberată) găsite de `valgrind`, se va aplica o penalizare de 10%.
- Dacă programul nu respectă stilul de cod, se poate aplica o penalizare de până la 10% (decisă de cadrul didactic de la laborator).

Nu încercați să trișați, deoarece se vor rula tool-uri de detecție a temelor plagiate.

Notă: Testele furnizate nu sunt neapărat aceleași cu cele pe care se va testa soluția voastră. Nu încercați să afișați rezultatele așteptate pe baza numelor fișierelor. Deasemenea, codul vostru trebuie să îndeplinească toate cerințele, chiar dacă unele dintre ele nu sunt acoperite de teste. Este posibil ca unele cazuri mai rare să nu apară în testele voastre, dar să fie testate la evaluare.