

Tema 2 Bibliotecă stdio



- Data publicare: **24.03.2022**
- **Deadline:** 7.04.2022, ora 23:55
- **Deadline hard:** 10.04.2022, ora 23:55

Obiectivele temei

- Familiarizarea cu modul de funcționare al bibliotecii standard input/output (stdio)
- Aprofundarea conceptelor de:
 - I/O buffering
 - Crearea de procese și rularea de fișiere executabile
 - Redirecțarea intrărilor și ieșirilor standard
 - Generarea de biblioteci dinamice

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 depăpa 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 reperitoriu dacă ați folosit **git**. **Asigurați-vă că responsabilitatea de teme este acordată de citire asupra repository-ului vostru.**

Pentru a folosi **git**, puteți să vă creați un repository privat pe **github** și să dați drepturi de citire pentru **molecula2788**.

Motivul pentru care încurajăm acest lucru este că responsabilitatea de teme se pot uita mai rapid pe **github** la temele voastre pentru a vă ajuta în cazul în care întâmpinați probleme/bug-uri.

Crash-course practic de git puteți găsi aici: **git-immersion**

Dacă ați creat deja un repo privat pe **Gitlab puteți să-l folosiți în continuare pe acela.**

Atât semnatura 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:

```
student@so:~$ git clone https://github.com/systems-cs-pub-ro/so
student@so:~$ cd assignments/2-stdio
```



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 2.

Enunț

Să se realizeze o implementare minimală a bibliotecii stdio, care să permită lucrul cu fișiere. Biblioteca va trebui să implementeze structura **SO_FILE** (similar cu **FILE** din biblioteca standard C), împreună cu funcțiile de citire/scriere. De asemenea, va trebui să ofere funcționalitatea de buffering.

Rezolvarea temei va trebui să genereze o bibliotecă dinamică numită **libso_stdio.so/so_stdio.dll** care implementează header-ul **so_stdio.h**. În acest header găsiți semnăturile funcțiilor exportate de biblioteca generată de voi.

SO_FILE

Este tipul de date care descrie un fișier deschis și este folosit de toate celelalte funcții. Un pointer la o astfel de structură este obținut la deschiderea unui fișier, folosind funcția **so_fopen**.

```
SO_FILE *so_fopen(const char *pathname, const char *mode);
```

Argumentele funcției sunt similare cu cele pentru **fopen**:

- **pathname** reprezintă calea către un fișier
- **mode** este un string care determină modul în care va fi deschis fișierul. Poate fi unul din următoarele:
 - **r** - deschide fișierul pentru citire. Dacă fișierul nu există, funcția eşuează.
 - **r+** - deschide fișierul pentru citire și scriere. Dacă fișierul nu există, funcția eşuează.
 - **w** - deschide fișierul pentru scriere. Dacă fișierul nu există, este creat. Dacă fișierul există, este trunciat la dimensiune 0.
 - **w+** - deschide fișierul pentru citire și scriere. Dacă fișierul nu există, este creat. Dacă fișierul există, este trunciat la dimensiune 0.
 - **a** - deschide fișierul în modul append - scriere la sfârșitul fișierului. Dacă fișierul nu există, este creat.
 - **a+** - deschide fișierul în modul append+read. Dacă fișierul nu există, este creat.

Funcția **so_fopen** alocă o structură **SO_FILE** pe care o întoarce. În caz de eroare, funcția întoarce **NULL**.

Închiderea unui fișier se face cu **so_fclose**.

```
int so_fclose(SO_FILE *stream);
```

Închide fișierul primit ca parametru și eliberează memoria folosită de structura **SO_FILE**. Întoarce 0 în caz de succes sau **SO_EOF** în caz de eroare.

Informații generale SO

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

Informații SO 2021-2022

- Examen
 - Examen CA/CC 2012-2013
 - Examen CA/CC 2013-2014
 - Examen CA/CC 2014-2015
 - Examen CA/CC 2015-2016
 - Examen CA/CB/CC 2016-2017
 - Examen CA/CB/CC 2017-2018
 - Examen CA/CB/CC 2018-2019
 - Examen CA/CB/CC 2019-2020
 - Examen CA/CB/CC 2020-2021
- Anunțuri
- Calendar
- Echivalări teme
- Distincții
- Karma Awards
- SO Need to Know
- Reguli generale și notare
- Orar și împărțire pe semigrupe

Laboratoare

- Resurse
 - Windows - **Tips&Tricks**
 - Tutorial Visual Studio
 - Linia de comandă în Windows
 - C/SO Tips
 - Macro-ul DIF
 - GDB
 - Function Hooking and Windows Dll Injection
 - IPC
 - Online
 - 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 - Gestiona memoria
 - Laborator 06 - Memoria virtuală
 - Laborator 07 - Profiling & Debugging
 - Laborator 08 - Threaduri Linux
 - Laborator 09 - Threaduri Windows
 - Laborator 10 - Operații IO avansate - Windows
 - Laborator 11 - Operații IO avansate - Linux
 - Laborator 12 - Implementarea sistemelor de fișiere

Curs

- Capitol 01: Introducere
- Capitol 02: Interfața sistemului de fișiere
- Capitol 03: Procese
- Capitol 04: Planificarea execuției, IPC
- Capitol 05: Gestiona memoria
- Capitol 06: Memoria virtuală
- Capitol 07: Analiza executablelor și proceselor
- Capitol 08: Securitatea memoriei
- Capitol 09: Fie de execuție
- Capitol 10: Sincronizare
- Capitol 11: Dispozitive de intrare/ieșire
- Capitol 12: Implementarea sistemelor de fișiere
- Capitol 13: Networking în sistemul de operare
- Capitol 14: Analiza performanței

Teme

- Tema Asistenți - Guardian process

Citirea și scrierea

Aceste operații se realizează cu ajutorul funcțiilor `so_fgetc`, `so_fputc`, `so_fread` și `so_fwrite`.

```
int so_fgetc(SO_FILE *stream);
```

Citește un caracter din fișier. Întoarce caracterul ca `unsigned char` extins la `int`, sau `SO_EOF` în caz de eroare.

```
int so_fputc(int c, SO_FILE *stream);
```

Scrie un caracter în fișier. Întoarce caracterul scris sau `SO_EOF` în caz de eroare.

```
size_t so_fread(void *ptr, size_t size, size_t nmemb, SO_FILE *stream);
```

Citește `nmemb` elemente, fiecare de dimensiune `size`. Datele citite sunt stocate la adresa de memorie specificată prin `ptr`. Întoarce numărul de elemente citite. În caz de eroare sau dacă s-a ajuns la sfârșitul fișierului, funcția întoarce 0.

```
size_t so_fwrite(const void *ptr, size_t size, size_t nmemb, SO_FILE *stream);
```

Scrie `nmemb` elemente, fiecare de dimensiune `size`. Datele ce urmează a fi scrise sunt luate de la adresa de memorie specificată prin `ptr`. Întoarce numărul de elemente scrise, sau 0 în caz de eroare.

Pozitionarea cursorului în fișier

```
int so_fseek(SO_FILE *stream, long offset, int whence);
```

Mută cursorul fișierului. Noua poziție este obținută prin adunarea valorii `offset` la poziția specificată de `whence`, astfel:

- `SEEK_SET` - noua poziție este `offset` bytes față de începutul fișierului
- `SEEK_CUR` - noua poziție este `offset` bytes față de poziția curentă
- `SEEK_END` - noua poziție este `offset` bytes față de sfârșitul fișierului

Întoarce 0 în caz de succes și -1 în caz de eroare.

```
long so_ftell(SO_FILE *stream);
```

Întoarce poziția curentă din fișier. În caz de eroare funcția întoarce -1.

Buffering

Proprietatea esențială a bibliotecii stdio este că aceasta face buffering. O structură `SO_FILE` are un buffer asociat, iar operațiile de citire/scriere se folosesc de acest buffer:

- Operațiile de citire (`so_fgetc`, `so_fread`) întorc datele direct din buffer. Atunci când bufferul este gol sau nu există date suficiente, acesta este populat cu date din fișier, folosind API-ul pus la dispozitiv de sistemul de operare (`read/Readfile`)
- Operațiile de scriere (`so_fputc`, `so_fwrite`) scriu datele în buffer. În situația când bufferul este plin (sau când se apelează `so_fflush`), datele se scriu în fișier, folosind API-ul pus la dispozitiv de sistemul de operare (`write/Writefile`).

```
int so_fflush(SO_FILE *stream);
```

Are sens doar pentru fișierele pentru care ultima operație a fost una de scriere. În urma apelului acestei funcții, datele din buffer sunt scrise în fișier. Întoarce 0 în caz de succes sau `SO_EOF` în caz de eroare.

Alte funcții

```
/* Linux */  
int so_fileno(SO_FILE *stream);  
/* Windows */  
HANDLE so_fileno(SO_FILE *stream);
```

Întoarce file descriptorul/HANDLE-ul asociat structurii `SO_FILE`.

```
int so_feof(SO_FILE *stream);
```

Întoarce o valoare diferență de 0 dacă s-a ajuns la sfârșitul fișierului sau 0 în caz contrar.

```
int so_ferror(SO_FILE *stream);
```

Întoarce o valoare diferență de 0 dacă s-a întâlnit vre o eroare în urma unei operații cu fișierul sau 0 în caz contrar.

Rularea de procese

```
SO_FILE *so_popen(const char *command, const char *type);
```

Lanseză un proces nou, care va executa comanda specificată de parametrul `command`. Ca implementare, se va executa `sh -c command`, respectiv `cmd /C command`, folosind un pipe pentru a redirecta intrarea standard/ieșirea standard a noului proces. Funcția întoarce o structură `SO_FILE` pe care apoi se pot face operațiile uzuale de citire/scriere, ca și cum ar fi un fișier obișnuit.

Dacă apelul `fork/CreateProcess` eșuează se va întoarce NULL.

Valorile parametrului `type` pot fi:

- "r" - fișierul intors este read-only. Operațiile `so_fgetc`/`so_fread` execute pe fișier vor citi de la ieșirea standard a procesului creat.
- "w" - fișierul intors este write-only. Operațiile `so_fputc`/`so_fwrite` execute pe fișier vor scrie la intrarea standard a procesului creat.

```
int so_pclose(SO_FILE *stream);
```

Se apelează doar pentru fișierele deschise cu `so_popen`. Așteaptă terminarea procesului lansat de `so_popen` și elibereză memoria ocupată de structura `SO_FILE`. Întoarce codul de ieșire al procesului (cel obținut în urma apelului `Waitpid/GetExitCodeProcess`). Dacă apelul `Waitpid/WaitForSingleObject` eșuează, se va întoarce -1.

Precizări/recomandări pentru implementare

- Dimensiunea implicită a bufferului unui fișier este de 4096 bytes.
- Pentru simplitate, sugerăm să începeți implementarea cu `so_fgetc`/`so_fputc`. Apoi `so_fread`/`so_fwrite` pot fi implementate pe baza lor.
- Pentru fișierele deschise în unul din modurile "r", "r+", "w", "w+", cursorul va fi poziționat inițial la începutul fișierului.
- Operațiile de scriere pe un fișier deschis în modul "a" se fac ca și cum fiecare operație ar fi precedată de un

Contestări

- Git. Indicații folosire GitLab
- Indicații generale teme
- Hackathon SO
- Tema 1 Multi-platform Development
- Tema 2 Bibliotecă stdio
- Tema 3 Loader de Executabile
- Tema 4 Planificator de threaduri
- Tema 5 Server web asincron

Table of Contents

- Tema 2 Bibliotecă stdio
 - Obiectivele temei
 - Dezvoltarea temei
 - Enunț
 - `SO_FILE`
 - Citirea și scrierea
 - Poziționarea cursorului în fișier
 - Buffering
 - Alte funcții
 - Rularea de procese
 - Precizări/recomandări pentru implementare
 - Precizări Linux
 - Precizări Windows
 - Testare
 - Materiale ajutătoare
 - Suport, întrebări și clarificări

- seek la sfârșitul fișierului.
- Pentru fișierele deschise în modul "a+", scrierile se fac la fel ca mai sus. În schimb, citirea se face inițial de la începutul fișierului.
- Conform standardului C, pentru fișierele deschise în modul update (i.e. toate modurile care conțin caracterul '+' la final: "r+", "w+", "a+"), trebuie respectate următoarele (de asemenea vor fi respectate în teste):
 - Între o operatie de citire urmată de o operatie de scris trebuie intercalată o operatie de fseek.
 - Între o operatie de scris urmată de o operatie de citire trebuie intercalată o operatie de fflush sau fseek.
- La o operatie de fseek trebuie avute în vedere următoarele:
 - Dacă ultima operatie făcută pe fișier a fost una de scris, tot bufferul trebuie invalidat.
 - Dacă ultima operatie făcută pe fișier a fost una de citire, continutul bufferului trebuie scris în fișier.
- Arhiva cu soluția trebuie să conțină un fișier `Makefile` care să alibă cel puțin următoarele reguli:
 - `build`: compilează biblioteca dinamică `libso_stdio.so`/`so_stdio.dll`
 - `clean`: șterge toate fișierele binare rezultante în urma compilării
- De asemenea, arhiva trebuie să conțină fișierul `README`.

Precizări Linux

- Tema se va realiza folosind funcții POSIX. Astfel `so_fopen` se va implementa folosind `open`, `so_fgetc`/`so_fread` folosind `read`, `so_fputc`/`so_write` folosind `write`, etc.
- În implementarea `open` trebuie să închideți capetele de pipe nefolosite atât în procesul părinte cât și în procesul copil.
- Nu aveți voie să folosiți funcția `system()`.

Precizări Windows

- Tema se va realiza folosind funcții Win32. Astfel `so_fopen` se va implementa folosind `CreateFile`, `so_fgetc`/`so_fread` folosind `ReadFile`, `so_fputc`/`so_write` folosind `WriteFile`, etc.
- În implementarea `open`, înainte de a apela `CreateProcess`, trebuie să marcați ca nemostenibil capătul de pipe nefolosit de către procesul copil. În acest scop, puteți folosi funcția `SetHandleInformation`.
- Pentru a detecta `EOF` la citirea dintr-un pipe trebuie ca `ReadFile` să întoarcă `FALSE`, iar `GetLastError` să întoarcă `ERROR_BROKEN_PIPE`.
- Din motive care îți de funcționalitatea checkerului, va trebui să deschideți fișierul cu `GENERIC_READ|GENERIC_WRITE` și `FILE_SHARE_READ|FILE_SHARE_WRITE`.

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 33 teste. Se pot obține maxim 9.5 puncte prin trecerea testelor. Se acordă 0.5 puncte din oficiu.
- 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.
- Testele de la 2 încolo nu nevoile și ca funcția `so_fleno` să fie implementată.
- Din punctul temei se vor scădea automat puncte pentru întâzieri și pentru warning-uri. La revizia temei, se poate scădea suplimentar pentru nerespectarea criteriilor scrise la secțiunea de `depunctări` ale temelor.
- Pentru neverificarea valorilor de return se vor scădea 0.4 puncte.
- În cazuri excepționale se poate scădea mai mult decât este menționat mai sus.



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



Pentru a inspecta diferențele între output-ul bibliotecii voastre și fișierele de referință ale checker-ului setați `DO_CLEANUP=no` în scriptul `run_test.sh` pentru Linux respectiv Windows.

Materiale ajutătoare

Cursuri utile:

- [Curs 1](#)
- [Curs 2](#)
- [Curs 3](#)

Laboratoare utile:

- [Laborator 1](#)
- [Laborator 2](#)
- [Laborator 3](#)

Resurse:

- Header-ul `so_stdio.h` expus de biblioteca `so_stdio`.

Pagina de Upload:

-

Suport, întrebări și clarificări

Pentru întrebări sau nelămuriri legate de temă folosiți forumul temei.



Orice întrebare pe forum trebuie să conțină o descriere cât mai clară a eventualei probleme. Întrebări de forma: "Nu merge X. De ce?" fără o descriere mai amănunțită vor primi un răspuns mai greu. Înainte să postați o întrebare pe forum citiți și celelalte întrebări (dacă există) pentru a vedea dacă întrebarea voastră a fost deja adresată sub o altă formă (în cazul în care răspunsul din partea echipei vine mai greu este mai rapid să căutați voi deja printre întrebările existente).

ATENȚIE să nu postați imagini cu părți din soluția voastră pe forumul pus la dispoziție sau orice alt canal public de comunicație. Dacă veți face acest lucru, vă asumăți răspunderea dacă veți primi copiat pe temă.