

Tema 3: Mecanisme de comunicare între procese

Varianta: 32453

Student: Nistor Mara

1 Descrierea temei

1.1 Revizuirea format-ului de fișier SF

În această temă se fac trimiteri la formatul de fișier SF (“**section file**”) descris în cerințele primei teme. Pentru a evita nevoia de a consulta două documente diferite, includem în acesta părțile necesare din descrierea primei teme.

Un fișier 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 de **body**.

— SF FILE STRUCTURE —

```
HEADER
BODY
```

Header-ul unui SF conține informații care identifică formatul acestuia și, de asemenea, 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 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 (i.e. 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 subcâmpuri. Astfel, SECTION_HEADERS este compus dintr-o secvență de elemente de tip SECTION_HEADER, fiecare dintre acestea având, la rândul lui, subcâmpurile: SECT_NAME, SECT_TYPE, SECT_OFFSET, și SECT_SIZE.

— HEADER —

```
MAGIC: 4
HEADER_SIZE: 2
VERSION: 4
NO_OF_SECTIONS: 1
SECTION_HEADERS: NO_OF_SECTIONS * sizeof(SECTION_HEADER)
    SECTION_HEADER: 16 + 4 + 4 + 4
        SECT_NAME: 16
        SECT_TYPE: 4
        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, excluzând câmpurile MAGIC și HEADER_SIZE.
- 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, chiar și în cazul în care numărul de versiune diferă de la un fișier la altul — vezi mai jos.

- Câmpul NO_OF_SECTIONS specifică numărul de elemente de tip SECTION_HEADER, descrise mai jos.
- Subcâmpurile unui SECTION_HEADER fie au nume explicite (e.g. SECT_NAME or SECT_TYPE), fie sunt descrise mai jos.

Body-ul unui SF, practic o secvență de octeți, este organizat ca o mulțime de secțiuni. O secțiune este formată dintr-o secvență de SECT_SIZE octeți consecutivi, începând de la octetul SECT_OFFSET din cadrul fișierului. Secțiunile consecutive nu vor fi neapărat una după alta. În 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. Octeții dintre secțiuni pot avea orice valoare, dar aceștia nu au nicio relevanță în interpretarea conținutului unui SF. Caseta de mai jos ilustrează două secțiuni și header-ele acestora într-un posibil SF.

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
...	...

Următoarele restricții se aplică anumitor câmpuri din SF:

- Valoarea câmpului MAGIC este "pJnE".
- Valoarea câmpului VERSION e un număr între 21 și 65, inclusiv.
- Valoarea câmpului NO_OF_SECTIONS e un număr între 8 și 16, inclusiv.
- Valorile valide pentru SECT_TYPE sunt: 63 66 62 77 95 46 .

1.2 Protocol de comunicare prin Pipe-uri

Programul vostru trebuie să comunice cu tester-ul prin pipe-uri cu nume. Numele pipe-urile și informația transmisă vor fi descrise mai jos. Deocamdată, vrem doar să explicăm protocolul de comunicare. Pentru asta, să presupunem că numele pipe-urilor sunt PIPE_1 și PIPE_2.

De asemenea, să presupunem scenariul în care unul din programe (ex. tester-ul) trebuie să trimită prin PIPE_1 un request (cerere) pentru o acțiune celuiilalt program (ex. soluția voastră). Request-ul e un mesaj compus dintr-o secvență de câmpuri, în următorul format:

Request Message Format	
<req_name>	... <number_param> ... <string_param> ...

Fiecare câmp este compus dintr-un număr specific de octeți. Sunt două tipuri de câmpuri:

1. **string-fields**, compuse dintr-o secvență de caractere (octeți care au valoarea unui cod de caracter afișabil) de dimensiune variabilă. Fiecare string-field de dimensiune variabilă este compus din două părți (subcâmpuri): **size** și **contents**. Subcâmpul size are doar un octet, care specifică numărul de octeți ale subcâmpului contents. Cele două subcâmpuri se află unul după celălalt, și compun un singur string-field. Acest tip de structură e necesară pentru procesul care citește din pipe, ca să știe câți octeți trebuie citiți pentru un anume string-field.

În capitolele ce urmează, când vom vorbi despre câmpuri string-fields în anumite mesaje, le vom marca cu ghilimele, de exemplu "CONNECT", "SUCCESS" și "ERROR". Însă, trebuie să aveți grijă ca octeții (valori hexazecimale) trimiși prin pipe să fie "07 43 4f 4e 4e 45 43 54", "07 53 55 43 43 45 53 53", respectiv "05 45 52 52 4f 52", pentru cele trei exemple menționate (spațiile sunt folosite doar pentru vizibilitate și nu sunt trimise prin pipe, octeții ilustrați vin imediat unul după celălalt), incluzând atât dimensiunea string-ului (primul octet) și conținutul (restul octeților).

2. **number-fields**, compuse din octeți care reprezintă un număr. Considerăm number-fields ca valori "unsigned int", reprezentate pe "sizeof(unsigned int)" octeți. În capitolele ce urmează, când vom vorbi despre câmpuri number-fields în anumite mesaje, le vom nota ca și numere, fără ghilimele, de exemplu 1234, 5678, etc. Nu le confundați cu reprezentări textuale ale numerelor, le notăm așa doar pentru claritate, prin pipe se trimite reprezentarea binară. Așa că, pentru notarea unui number-field cu valoarea 1234, următorii patru octeți (valori hexazecimale) trebuie trimiși, de fapt, prin pipe: "00 00 04 D2" (doar arătăm echivalența în hexazecimal al numărului 1234, fără să ținem cont de aspectul little-endian, care e transparent).

Notăți că, față de string-urile C, câmpurile string-fields nu se termină cu '\0'. Dacă aveți nevoie să lucrați cu ele ca și string-uri C (ex. să le afișați pe ecran cu *printf* sau să le copiați cu *strcpy*), trebuie să le adăugați explicit octetul '\0'.

De asemenea, notați că în structura mesajului ilustrată mai sus caracterele spațiu sunt folosite doar pentru o vizibilitate mai bună. Similar, se aplică și la ghilimelele (") folosite pentru a evidenția string-fields și secvențele "...", care doar sugerează că ar putea exista alți parametri de diferite tipuri între cei ilustrați.

Numele request-ului adică "<req_name>", e un string-field, ale cărui valori acceptate sunt descrise mai jos. Un request poate fi urmat de zero sau mai mulți parametri având unul din cele două tipuri descrise mai sus. Prin urmare, "<number_param>" reprezintă un posibil number-field și "<string_param>" un string-field. Numărul și semnificația parametrilor este descrisă mai jos, pentru fiecare tip de request acceptat.

Odată ce mesajul request este citit din pipe de către programul receptor, și requestul este tratat, un răspuns trebuie trimis înapoi, conținând rezultatul tratării request-ului. În scenariul nostru PIPE_2 este folosit pentru a trimite înapoi mesajul response, a cărui structură e ilustrată mai jos:

Response Message Format				
<req_name>	<response_status>	...	<number_param>	... <string_param> ...

Putem observa că structura mesajului response este similară cu cea a mesajului request, câmpurile lui respectând aceeași convenție. Mesajul response conține numele request-ului corepondent pentru validarea acestuia, urmat de un status al tratării request-ului (dat de string-field-ul "<response_status>") și zero sau mai multe câmpuri (număr sau string) prin care se transferă rezultatul tratării request-ului. Valorile acceptate pentru fiecare câmp ale mesajelor response vor fi descrise mai jos, pentru fiecare tip de request.

2 Cerințele temei

Trebuie să scrieți un program C numit "a3.c" care implementează următoarele cerințe.

2.1 Compilare și execuție

Programul vostru C trebuie să **compileze fără erori** pentru a fi acceptat. Comanda de compilare va fi cea de mai jos:

Compilation Command	
gcc	-Wall a3.c -o a3

Warning-uri la compilare vor aduce o penalizare de 10% la nota finală.

La execuție, executabilul vostru (il numim “a3”) **trebuie să ofere funcționalitatea minimă 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.

Running Command

`./a3`

2.2 Conexiunea prin pipe

Programul vostru trebuie să stabilească comunicarea cu tester-ul folosind două pipe-uri cu nume. Pentru a stabili comunicarea, se fac următorii pași:

1. creează un pipe denumit “RESP_PIPE.32453”;
2. deschide un pipe denumit “REQ_PIPE.32453”, care este creat automat de către tester;
3. scrie următorul mesaj request în pipe-ul “RESP_PIPE.32453”

Connection Request Message

`"CONNECT"`

Dacă toți pașii de mai sus s-au executat cu succes, programul vostru trebuie să afișeze pe ecran următorul mesaj:

Connection Successful Message

`SUCCESS`

Altfel, trebuie să afișeze un mesaj de eroare, cum e mai jos.

Connection Failure Message

`ERROR
cannot create the response pipe | cannot open the request pipe`

După stabilirea conexiunii, programul vostru trebuie să execute următorii pași, în buclă:

1. citește din pipe-ul “REQ_PIPE.32453” un mesaj request trimis de programul tester;
2. tratează acel request în felul în care e descris mai jos, pentru fiecare tip de request;
3. scrie în pipe-ul “RESP_PIPE.32453” rezultatul tratării request-ului primit.

2.3 Request ping

Mesajul request arată în felul următor

Ping Request Message

`"PING"`

Pentru un ping request, programul vostru trebuie doar să trimit înapoi următorul mesaj response

Ping Response Message

`"PING" "PONG" 32453`

2.4 Request pentru crearea unei regiuni Shared Memory

Mesajul request arată în felul următor

```
_____ SHM Creation Request Message _____  
"CREATE_SHM" 4792195
```

Pentru un astfel de request, programul vostru trebuie să creeze o regiune shared memory de 4792195 octeți, folosind cheia 15732. Permisunile pentru acea regiunea creată trebuie să fie "664". Dacă s-a creat cu succes, regiunea shared memory trebuie să fie atașată de către programul vostru în propriul spațiu de adrese virtuale.

Mesajul response indică succesul sau eroarea în următorul fel

```
_____ Successful SHM Creation Response Message _____  
"CREATE_SHM" "SUCCESS"
```

```
_____ Error SHM Creation Response Message _____  
"CREATE_SHM" "ERROR"
```

2.5 Request pentru scrierea în Shared Memory

Mesajul request arată în felul următor

```
_____ Write to SHM Request Message _____  
"WRITE_TO_SHM" <offset> <value>
```

Câmpurile "offset" și "value" sunt number-fields și indică un offset (deplasament) și valoarea care trebuie scrisă în regiunea shared memory la offset-ul respectiv (de tip `unsigned int`).

Programul vostru trebuie să verifice că offset-ul dat este în regiunea shared memory (adică între 0 și 4792195) și că toți octeții care ar trebui scriși, de asemenea, corespund unor offset-uri din interiorul regiunii shared memory.

După ce a scris în shared memory, programul vostru trebuie să trimită un mesaj response indicând succesul sau eroarea operației de scriere, într-unul din formatele de mai jos.

```
_____ Successful Write to SHM Response Message _____  
"WRITE_TO_SHM" "SUCCESS"
```

```
_____ Unsuccessful Write to SHM Response Message _____  
"WRITE_TO_SHM" "ERROR"
```

2.6 Request pentru map-area unui fișier în memorie

Mesajul request arată în felul următor

```
_____ Map File Request Message _____  
"MAP_FILE" <file_name>
```

Programul vostru trebuie să map-eze în memorie fișierul cu numele dat de parametrul string-field "<file_name>".

După map-area în memorie a fișierului cerut, programul trebuie să trimită un mesaj răspuns indicând succesul sau eroarea operației, într-unul din formatele de mai jos.

```
_____ Successful Map File Response Message _____  
"MAP_FILE" "SUCCESS"
```

```
_____ Unsuccessful Map File Response Message _____  
"MAP_FILE" "ERROR"
```

2.7 Request pentru citirea de la un offset din fișier

Mesajul request arată în felul următor

```
_____ Read From File Offset Request Message _____  
"READ_FROM_FILE_OFFSET" <offset> <no_of_bytes>
```

Programul vostru trebuie să citească din fișierul care este curent map-at în memorie “<no_of_bytes>” octeți de la offset-ul “<offset>”.

Dacă programul vostru nu citește din fișiere map-ate în memorie, adică direct din memorie, ci folosește funcția `read()` ca să citească direct din fișier, implementarea va fi considerată doar parțial corectă și penalizări vor fi aplicate (vezi mai jos în Section 3.2).

Octeții citiți trebuie să fie copiați la începutul regiunii shared memory înainte de a trimite mesajul response tester-ului.

Mesajul răspuns va avea următoarea structură, depinde de succesul sau eroarea tratării request-ului, într-unul din formatele de mai jos.

```
_____ Successful Read From File Offset Response Message _____  
"READ_FROM_FILE_OFFSET" "SUCCESS"
```

```
_____ Unsuccessful Read From File Offset Response Message _____  
"READ_FROM_FILE_OFFSET" "ERROR"
```

Citirea este cu succes dacă:

- există o regiune shared memory,
- un fișier este map-at în memorie (sau doar deschis, dacă alegeți să lucrați direct cu fișier), și
- offset-ul cerut adunat cu numărul de octeți de citit e un număr mai mic decât mărimea fișierului.

2.8 Request pentru citirea dintr-o secțiune SF

Mesajul request arată în felul următor

```
_____ Read From File Section Request Message _____  
"READ_FROM_FILE_SECTION" <section_no> <offset> <no_of_bytes>
```

Programul vostru trebuie să citească din secțiunea “<section_no>” a fișierului curent map-at în memorie (considerat fișier SF) “<no_of_bytes>” octeți de la offset-ul “<offset>” în acea secțiune. Parametrul “<section_no>” va avea valori valide între 1 și numărul de secțiuni ale fișierului SF map-at în memorie.

Dacă programul vostru nu citește din fișiere map-ate în memorie, adică direct din memorie, ci folosește funcția `read()` ca să citească direct din fișier, implementarea va fi considerată doar parțial corectă și penalizări vor fi aplicate (vezi mai jos în Section 3.2).

Octeții citiți trebuie să fie copiați la începutul regiunii shared memory înainte de a trimite mesajul response tester-ului.

Mesajul răspuns va avea următoarea structură, depinde de succesul sau eroarea tratării request-ului, într-unul din formatele de mai jos.

```
_____ Successful Read From File Section Response Message _____  
"READ_FROM_FILE_OFFSET" "SUCCESS"
```

```
_____ Unsuccessful Read From File Section Response Message _____  
"READ_FROM_FILE_OFFSET" "ERROR"
```

2.9 Request pentru citirea de la un offset din spațiul de memorie logic

Mesajul request arată în felul următor

```
----- Read From Logical Space Offset Request Message -----  
"READ_FROM_LOGICAL_SPACE_OFFSET" <logical_offset> <no_of_bytes>
```

Programul vostru trebuie să citească din fișierul curent map-at în memorie “<no_of_bytes>” octeți de la un offset obținut printr-un calcul, pornind de la “<logical_offset>” dat. “<logical_offset>” este un offset în ceea ce numim “spațiul de memorie logic” al fișierului. Spațiul de memorie logic poate fi văzut ca o zonă contiguă de memorie în memoria procesului, ale cărui conținut poate fi obținut în următorul fel, presupunând că fișierul map-at în memorie este un fișier SF:

- Secțiunile fișierului SF ar trebuie încărcate în memoria procesului începând de la o adresă dată, considerată începutul spațiului de memorie logic al fișierului SF (notați că valoarea adresei nu ne interesează, doar structura zonei de memorie care începe la adresa respectivă);
- header-ul fișierului SF se folosește doar la localizarea și citirea secțiunilor SF, dar nu este încărcat în spațiul de memorie logic;
- fiecare secțiune SF trebuie încărcată în spațiul de memorie logic la următoarea adresă liberă egală cu un multiplu de 1024 octeți după sfârșitul secțiunii încărcate anterior. Numim valoarea 1024 aliniamentul în memorie al secțiunilor, și spunem despre secțiuni că sunt aliniate în memorie la valoarea 1024;
- ordinea secțiunilor SF încărcate în memoria logică este cea în care apar în header-ul fișierului SF; notați că această ordine nu este neapărat aceeași cu cea a conținutului secțiunilor din fișierul SF, adică o secțiune a cărei header este înaintea header-ului altei secțiuni ar putea avea conținutul în fișier la un offset mai mare decât cel al conținutului celeilalte secțiuni (deci, după aceasta).

Pentru clarificare, să considerăm cazul unui fișier SF cu trei secțiuni de 1025, 20, respectiv 2048 octeți și aliniamentul lor în memorie să fie 1024. Presupunem, pentru simplitate, că adresa de început a spațiului de memorie logic este 0 (zero), secțiunile vor fi încărcate în spațiul logic începând de la următoarele offset-uri:

- 0, prima secțiune;
- 2048, a doua secțiune;
- 3072, a treia secțiune.

Notați că în exemplul de mai sus nu am menționat offset-ul din fișier al niciunei secțiuni. Evident, acele offset-uri sunt diferite de cele din spațiul de memorie logic și trebuie extrase din header-ul SF.

De asemenea, notați că nu vi se cere să creați spațiul de memorie logică al unui fișier SF, ci doar să faceți translația de la un octet în memoria logică la octetul corespunzător din fișierul SF.

Dacă programul vostru nu citește din fișiere map-ate în memorie, adică direct din memorie, ci folosește funcția `read()` ca să citească direct din fișier, implementarea va fi considerată doar parțial corectă și penalizări vor fi aplicate (vezi mai jos în Section 3.2).

Octeții citiți trebuie copiat la începutul regiunii shared memory înainte de a trimite mesajul response tester-ului.

Mesajul răspuns va avea următoarea structură, depinde de succesul sau eroarea tratării request-ului, într-unul din formatele de mai jos.

```
----- Successful Read From Logical Space Offset Response Message -----  
"READ_FROM_FILE_OFFSET" "SUCCESS"
```

```
----- Unsuccessful Read From Logical Space Offset Response Message -----  
"READ_FROM_FILE_OFFSET" "ERROR"
```

2.10 Request exit

Mesajul request arată în felul următor

Exit Request Message

"EXIT"

Când primește acest tip de mesaj, programul vostru trebuie să închidă conexiunea / pipe-ul request, să închidă și să șteargă pipe-ul response și să-și termine execuția.

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 pe alte sisteme de operare.

După crearea folder-ului “test_root” și a conținutului acestuia, script-ul va rula, de asemenea, programul vostru, afișând rezultatele diferitelor teste. Nota maximă (10) se obține atunci când trec toate testele, fără nicio 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, pentru lucrul cu fișiere, TREBUIE folosite funcțiile `open()`, `read()`, `write()` etc., nu funcțiile de nivel înalt `fopen()`, `fgets()`, `fscanf()`, `fprintf()` etc. Singurele excepții sunt citirea de la `STDIN` sau afișarea în `STDOUT` / `STDERR`, cu funcții precum `scanf()`, `printf()`, `perror()` respectiv funcții de manipulare a string-urilor, cum ar fi `sscanf()`, `snprintf()`;
- Privind accesul la fișiere SF, după cum am mai menționat, sunteți restricționați din a folosi funcția `read()` pentru citirea din fișier, ci trebuie să map-ați fișierul în memorie și să citiți direct din memorie. Această limitare nu este strictă, totuși, și dacă o încălcați, se vor aplica penalizări la nota finală.

Recomandări:

- Pentru token-izarea string-urilor (separarea după elemente specifice, cum ar fi spații) recomandăm utilizarea funcțiilor `strtok()` sau `strtok_r()`.
- Dacă observați un comportament neașteptat sau ciudat al programului vostru, precum să rămână blocat (hanging), citirea unor octeți neașteptați din pipe etc., asta poate fi (pe lângă alte motive, cum ar fi bug-uri în program) din cauza unor procese rămase blocate în urma unor teste anterioare. Ca să vă asigurați că programul vostru nu este influențat din cauza asta, încercați comanda “killall -9 python a3”.
- Pentru obținerea de detalii despre execuția testelor, ca să vă ajute la depanare, puteți schimba opțiunea “VERBOSE=True” (implicit este False) în fișierul “tester.py”.

3.2 Evaluare și cerințe minime

- Dacă programul are warning-uri la compilare, se va aplica o penalizare de 10%.
- Dacă programul nu map-ează fișierul cerut în memorie și îi citește conținutul din zona de memorie respectivă, ci folosește `read()`, o penalizare de 30% se aplică la testele core-spunzătoare.
- Dacă programul nu respectă stilul de cod cerut, se poate aplica o penalizare de 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 de test. De asemenea, codul vostru trebuie să îndeplinească toate cerințele, deoarece setul de teste nu acoperă neapărat toate cazurile posibile. Este posibil ca unele cazuri mai rare să fie acoperite doar de testele de la evaluare, nu și de cele furnizate.