Curs 02: Interfața sistemului de fișiere

Sumar curs anterior

sistem de operare: roluri, definiţii
de ce să știm sisteme de operare, low-level, de ce să fim full stack: performanță, securitate, depanare/troubleshooting
nevoia de sistem de operare: portabilitate, izolare/securitate, planificarea accesului la resurse
probleme cu sistemul de operare: risc de securitate (TCB (Trusted Computing Base) crescut), overhead
user/application space vs kernel/supervisor space, context privilegiat/context neprivilegiat
apel de sistem (pentru accesul la codul sistemului de operare): protejare/securitate vs. overhead
microkernel vs kernel monolitic: securitate/izolare vs performanţă

Date

preluate date de la intrare și stocare
prelucrare date stocate
generare date și stocare

= sistem de fișiere

datele sunt stocate ca fișiere
sf este organizarea datelor ca fișiere, de obicei pe disc (suport persistent)
persistenţă, organizare/separare, eficienţă spaţială/temporală, partajare
+ diagramă: aplicaţii (utilizare) ---> sistem de fișiere (organizare) ---> date (suport persistent)

Fișiere

fișierul este o secvenţă de octeţi (byte stream) stocaţi pe disc
fișiere binare; text cele pentru care octeţii sunt citibili (human readable)
fișierele sunt formate din date și metadate (atribute ale fișierelor), ambele stocate pe suport persistent
metadatele unui fișier se găsesc într-o structură numită FCB (File Control Block), (ex. inode pe Unix)
+ diagramă: date + metadate (FCB), atribute în metadate
metadate: nume (șir de caractere), identificator (de ex. inode number), dimensiune, user (owner), group, permisiuni, timpi de acces
comanda stat în Unix (și apelul stat(2)/fstat(2)) afișează metadatele unui fișier
+ demo cu stat pe fișiere

Fişier deschis

pentru ca un fişier să fie folosit trebuie să fie deschis
apeluri precum open(2) (Linux)/fopen(3) (standard C) deschid un fişier
un fişier este deschis dintr-un proces
deschiderea unui fişier duce la crearea unei structuri de fişier deschis; structura este în memorie în kernel/supervisor space în urma apelului de sistem de creare a unui fişier (ex open(2) pe Linux, CreateFile pe Windows)
+ diagramă: proces ---> apel de sistem de deschidere ---> structură de fişier deschis ---> structură de fişier pe disc (FCB + date)
structura de fişier deschis conţine: referinţă către FCB-ul fişierului, permisiuni de deschidere, cursor/pointer de fişier (file pointer/cursor)
+ diagramă continuată: adăugare permisiuni de deschidere şi cursor/pointer de fişier în structura de fişier deschis

Deschiderea unui fişier

permisiunile de deschidere sunt un subset al permisiunilor FCB-ului: dacă un fişier are permisiuni de FCB de citire (pentru utilizatorul procesului care face operaţia) atunci permisiunile de deschidere pot fi doar de citire, nu de scriere
cursorul de fişier este plasat pe valoarea 0 sau pe dimensiunea fişierul (din FCB) depinzând de opţiunile de deschidere
+ demo cu apeluri open:
open(..., O_RDWR); // cursorul pus pe 0
open(..., O_RDWR | O_APPEND); // cursorul pus pe dimensiunea fişierului (din FCB)
open(..., O_RDWR | O_TRUNC); // cursorul pus pe 0, dimensiunea pusă pe 0, datele fişierului "şterse" (marcate ca fiind blocuri libere)
+ demo: folosim strace pentru a vedea ce efect au apelurile ANSI fopen(..., "r"); fopen(..., "w"); fopen(..., "rw"); fopen(..., "r+"); fopen(..., "w+"); fopen(..., "a"); fopen(..., "a+");
apelurile de deschidere a unui fişier întorc un handle de gestiune a fişierului, folosit de operaţii de citire şi scriere
la nivelul cel mai de jos, handle-ul este un număr, numit descriptorul de fişier, asta întoarce apelul open(2)

Crearea unui fişier

atunci când deschidem un fişier şi acesta nu există, poate fi creat
pentru a fi creat trebuie să:
* avem permisiuni de scriere în directorul în care va fi creat
* avem opţiunea O_CREAT sau echivalentă la apelul de deschidere
când este creat un fişier, nu are date şi se iniţializează FCB-ul acestuia:
* numele este dat ca parametru
* user/group este al procesului care a efectuat apelul de sistem

* timpii de acces sunt iniţializaţi la timpul curent
* dimensiunea este 0
* permisiunile (numite şi permisiuni de creare) sunt date ca al treilea parametru: atenţie, e o greşeală frecventă omiterea permisiunilor; în caz de omitere fişierul va fi creat cu permisiuni care pot să nu permită toată operaţiile dorite
+ demo cu open(2) pentru creare fişier cu permisiunile de creare bine transmise, în octal

Descriptorul de fişier, tabela descriptorilor de fişiere

descriptorul este un index într-o tabelă numită tabela descriptorilor de fişiere (file descriptor table), reţinută în kernel/supervisor space
există câte o tabelă de descriptori de fişiere per proces
o intrare în tabela descriptorilor de fişiere este un pointer către structura de fişier deschis
+ diagramă completă cu descriptor de fişier (user/application space) ---> tabelă de descriptori de fişiere ---> structură de fişier deschis ---> structură de fişier pe disc (ultimele trei în kernel space)
o intrare în tabelă este validată/iniţializată în momentul deschiderii unui fişier (open(2)/fopen(3)); este invalidată în momentul închiderii unui fişier (close(2)/fclose(3))
+ demo: scris la descriptorul 3 folosind write(2), descriptorul 3 fiind nevalid, afişat eroare (errno + perror)
intrările în tabelă pot referi şi altceva în afară de fişiere şi pot fi create şi altfel: sockeţi (creare intrare cu socket(2), închidere folosind close(2)), pipe-uri (creare cu pipe(2), închidere cu close(2)), terminale; în tabelă se găsesc pointeri către structuri specifice de socket sau pipe sau terminal (nu structuri de fişier deschis)
dimensiunea tabelei este limitată pentru a preveni abuzuri de prea multe fişiere deschise
+ demo cu getfdtablesize(2)
descriptorii 0, 1 şi 2 sunt descriptorii standard şi referă respectiv, standard input, standard output şi standard error (uzual referă structuri de terminal)
+ demo cu un proces sleep şi afişarea descriptorilor standard cu lsof, "numele" fişierului este numele dispozitivului de terminal (/dev/pty/0 sau similar)
+ demo cu afişarea descriptorilor standard pentru shell-ul curent
primul fişier deschis va avea descriptorul 3
+ demo cu descriptorul primului fişier deschis
şi funcţiile C standard obţin tot un descriptor de fişier: fopen(3) apelează open(2), iniţializează o intrare în tabelă şi întoarce indexul corespunzător ca descriptor; acesta este "îmbrăcat" în structura FILE
+ demo cu structura _IO_FILE pe Linux care conţine pe lângă buffere "int fd;" sau ceva similar

Operaţii cu date şi cursorul de fişier

o dată obţinut un descriptor de fişier (sau un handle precum FILE) putem face operaţii cu fişierele
operaţiile uzuale sunt read/write
read: citim date *din* fişier *într-un* buffer (din user/application space)

write: scriem date *dintr-un* buffer (din user/application space) *în* fişier
operaţiile de citire/scriere avansează cursorul de fişier
+ diagramă cu fluxul datelor la citire, fluxul datelor la scriere
citirea nu avansează dincolo de dimensiunea fişierului; scrierea poate trece peste şi modifica/creşte dimensiunea fişierului
+ demo cu afişarea cursorului fişierului după operaţii de citire şi scriere
operaţiile de poziţionare (lseek/fseek/SetFilePointer) pot plasa oriunde cursorul; îl pot da înapoi
+ demo cu afişarea cursorului după operaţii de poziţionare

Operaţii mai interesante/ciudate

putem configura dimensiunea fişierului la o valoare dată, fără a avea date de acea dimensiune: truncate(2)/ftruncate(2) şi comanda truncate
use case: pentru a da impresia unor aplicaţii că au fişier suficient de mari (de exemplu discuri de maşini virtuale)
+ demo cu comanda truncate şi funcţiile truncate(2)/ftruncate(2)
sparse file: fişiere cu "găuri"
poziţionare cursor de fişier dincolo de limita fişierului şi scris informaţii
use case: Bittorrent, se “alocă” fictiv tot fișierul și apoi se scriu date unde sunt primite chunk-urile prin protocolul Bittorrent, în rest rămân găuri
+ demo de creare a unui sparse file
+ tabel cu ce modifică apelurile open, read, write, seek, truncate: cursorul de fişier şi dimensiunea fişierului

Operaţii cu date: system I/O vs buffered I/O

putem efectua operaţii cu date citire/scriere folosind interfaţa low-level (wrappere peste apeluri de sistem: read(2)/write(2) sau WriteFile(2)/ReadFile(2)) sau interfaţa C standard (fread(3)/fwrite(3))
interfaţa C standard este portabilă între platforme şi foloseşte buffere, se mai numeşte buffered I/O
operaţiile trec prin buffere, bufferele sunt parte din libc, în user/application space
+ diagramă cu aplicaţie ---> libc (buffered I/O) cu buffere ---> system I/O / apeluri de sistem ---> kernel (fişiere)
avantaj buffered I/O: evitare apel de sistem, dezavantaj: nu sunt "sincrone" (nu "ajung" direct la dispozitiv) şi consumă memorie pe buffere
se face "flush" la commanda fflush sau când se ajunge la Enter (pentru standard output) sau la buffer plin (pentru fişiere) sau mereu (unbuffered, pentru standard error)
+ demo cu apeluri de sistem efectuate atunci când facem buffered I/O faţă de când face system I/O

Internele redirectărilor

se întâmplă să dorim ca informaţiile afişate la ieşirea standard să ajungă într-un fişier: comanda > file
ca să se întâmple acest lucru trebuie ca intrare cu indexul/descriptorul 1 (corespunzător ieşirii standard) din tabela descriptorilor de fişiere a procesului să refere o structură de fişier deschis pentru fişierul file
+ diagramă cu
acest lucru îl obţinem prin close(1) şi apoi open("file", ...)
+ demo cu close şi open
uzual, însă, pentru mai multă flexibilitate, de exemplu refolosirea unui fişier deja deschis, folosim duplicarea descriptorilor

= duplicarea descriptorilor

facem ca o intrare din tabela de descriptori de fişiere să refere aceeaşi structură de fişier deschis ca altă intrare
apelul dup(fd) face ca prima intrare liberă găsită în tabela de descriptori de fişiere să refere structura de fişier deschis referită de fd
apelul dup2(olfd, newfd) face ca intrarea cu indexul newfd să refere structură de fişier deschis referită de oldfd; newfd este închis dacă este cazul să fie deschis
+ diagramă cu tabelă de descriptori de fişier cu mai mulţi descriptori referind aceeaşi structură de fişier deschis
+ (opţional): discuţie despre nevoia de dup2(2) şi condiţii de cursă pentru dup(2) + close(2)
dup/dup2 sunt folosite la redirectare
+ demo: close(1), dup(fd)
+ demo: dup2(fd, 1)
+ actualizare diagramă cu paşii pentru duplicarea standard output
similar se întâmplă la operatorul | (pipe) din shell de redirectare a comenzilor; îl vom discuta la cursul 3: Procese

Închiderea şi ştergerea fişierelor

fişierele se închid folosind close(2)/fclose(3)
închiderea presupune invalidarea intrării corespunzătoare descriptorului din tabela de descriptori de fişiere
mai există un câmp în structura de fişier deschis: contor de utilizare: operaţiile de duplicare cresc acel contor
atunci când toate toate referinţele dispar, structura de fişier este eliberată din memorie (spunem că fişierul este închis)
+ diagramă cu referinţe multiple din tabela de descriptori de fişier către structura de fişier şi operaţii close(2)
similar pot exista mai multe referinţe de la fişier deschis la acelaşi FCB: dacă facem mai multe apeluri open(2)

diferenţa dintre dup şi mai multe apeluri open(): nu se partajează cursorul de fişiere
+ demo cu dup şi open şi modificarea cursorului de fişiere folosind fseek, urmărire cu lsof

Despre cursul 12: implementarea sistemului de fişiere

operaţiile prezentate nu diferă între sisteme de fişiere diferite
diferenţele de implementare a sistemului de fişiere le vom prezenta în cursul 12
cursul va prezenta internele FCB pe disc
structura unui sistem de fişiere pe disc

Sumar

stocăm datele în fişiere, fişierele sunt date structurate, metadatele fişierelor sunt agregate în FCB (file control block)
fişierele sunt deschide pentru a fi folosite, se întoarce la nivel low-level un descriptor de fişier
un descriptor este o intrare într-o tabelă de descriptori de fişiere care conţine un pointer către o structură de fişier deschis
mai multe structuri de fişier deschis pot referi acelaşi FCB (mai multe apeluri open(2) pe acelaşi fişier)
mai multe intrări în tabela de descriptori de fişiere pot referi aceeaşi structură de fişier deschis (folosind dup(2) sau dup2(2))
operaţiile read(2), write(2) şi lseek(2) modifică cursorul de fişier (prezent în structura de fişier deschis)
operaţia ftruncate(2) modifică dimensiunea fişierului (prezentă în FCB)