Nume:

Roșca Maia-Teodora

Donea Fernando-Emanuel

Loghin Antonia-Miruna

FIFO Server - Deftones

Introducere

Acest proiect implementează un sistem client-server în Linux utilizând limbajul **Bash** și fișiere FIFO (First In, First Out) drept canale de comunicare între clienți și server. Scopul proiectului este să faciliteze accesul la informațiile din paginile de manual ale comenzilor Linux.

Specificații tehnice

1. Fișierul FIFO bine-cunoscut (well-known FIFO)

- Serverul utilizează un fișier FIFO cu un nume bine-cunoscut pentru a primi cererile clienților.
- Locația și numele acestui fișier FIFO sunt configurabile printr-un fișier de configurare.

2. Structura cererilor clientului

- Cererile clienților au următorul format:

BEGIN-REQ (client-pid: command-name) END-REQ

client-pid: ID-ul de proces al clientului.

command-name: Numele comenzii Linux pentru care se dorește informația din pagina de manual.

3. Comportamentul serverului

- Primirea cererilor

Serverul citește cererile clienților din FIFO-ul bine-cunoscut.

- Prelucrarea cererii

Serverul extrage din cerere PID-ul clientului și numele comenzii.

Apelează comanda man command-name pentru a obține informațiile din pagina de manual.

- Transmiterea răspunsului

Serverul creează un fișier FIFO personalizat pentru fiecare client, conform formatului:

/tmp/server-reply-XXXX

Unde XXXX reprezintă PID-ul clientului.

Răspunsul (conținutul paginii de manual) este transmis prin acest fișier FIFO personalizat.

4. Comportamentul clientului

- Clientul creează cererea conform formatului specificat și o trimite serverului prin FIFO-ul bine-cunoscut.
- Clientul așteaptă răspunsul în fișierul FIFO personalizat creat de server.
- După ce conținutul paginii de manual este afișat pe ecran, clientul șterge fișierul FIFO personal.

Dificultăți întâmpinate și soluțiile adaptate:

- Utilizarea mecanismelor PIPE pentru procesarea datelor de ieșire:

Soluție:

Un scenariu comun în utilizarea PIPE-urilor este manipularea ieșirii comenzii man. Dacă ieșirea comenzii este foarte mare sau conține date nestructurate, pot apărea probleme de performanță sau dificultăți în procesare.

În scriptul server.sh, ieșirea comenzii man este redirecționată direct în FIFO-ul clientului folosind:

```
man "$commandName" > "$clientFifo"
```

Aceasta rezolvă problema stocării intermediare, evitând supraîncărcarea memoriei. PIPE-ul este folosit aici implicit prin mecanismul de redirecționare, transferând datele direct către client.

- Blocare la FIFO-ul clientului:

FIFO-urile pot crea blocaje dacă un proces (client sau server) nu este pregătit să citească sau să scrie. De exemplu, dacă clientul nu consumă răspunsul din FIFO, serverul poate rămâne blocat în așteptare.

În scriptul client.sh, clientul este obligat să consume FIFO-ul dedicat prin:

```
cat $clientFifo
```

Această instrucțiune asigură că răspunsul serverului este citit imediat ce este disponibil. În plus, după ce răspunsul este citit, FIFO-ul este șters pentru a preveni conflictele:

```
rm -f $clientFifo
```

- Lipsa FIFO-ului server-ului:

Dacă FIFO-ul serverului ($fifoServer) nu există sau a fost șters accidental, clienții nu pot trimite cereri, iar funcționarea generală este întreruptă.

În server.sh, FIFO-ul serverului este verificat și creat dacă nu există:

```
if [[ ! -p $fifoServer ]]; then
    mkfifo $fifoServer
fi
```

Aceasta asigură că FIFO-ul principal este mereu prezent pentru a primi cereri. Similar, în client.sh, clientul verifică existența acestui FIFO înainte de a trimite cererea:

```
if [[ ! -p $fifoServer ]]; then
    echo "Eroare: FIFO-ul serverului nu există!"
    exit 1
fi
```

- Coruperea datelor în FIFO:

Dacă mai mulți clienți trimit cereri simultan, există riscul ca cererile să se amestece în FIFO-ul serverului, rezultând în date corupte.

În server.sh, serverul verifică integritatea fiecărei cereri folosind o expresie regulată:

```
if [[ $cerere =~ BEGIN-REQ\ \[([0-9]+):\ ([a-zA-Z0-9_-]+)\]\ END-REQ ]]; then
    # Procesare cerere validă
else
    # Ignorare cerere invalidă
fi
```

Această abordare asigură că doar cererile formatate corect sunt procesate, ignorând orice alt tip de intrare.

- Fișierul de configurare lipsește sau este corupt

Dacă fișierul configServer.conf lipsește sau conține date incorecte, scripturile nu pot încărca variabilele necesare ($fifoServer, de exemplu), iar execuția poate eșua.

Ambele scripturi încep prin încărcarea configurării:

```
source configServer.conf
```

Dacă fișierul lipsește, Bash va genera o eroare și va opri execuția scriptului. Aceasta obligă utilizatorul să se asigure că fișierul de configurare este prezent și valid.

Explicarea codului:

1.server.sh

```
#!/bin/bash
```

source configServer.conf

Incarcă și execută conținutul fișierului configServer.conf în aceași sesiune de shell ca script-ul principal, astfel variabilele, funcțiile și alte resurse definite în fișier devin disponibile imediat în script-ul prinicipal, fără a fi nevoie de alta configurare. De asemenea tot conținutul fișierului poate fi folosit în mai multe script-uri pentru variabile sau pentru funcții. Așadar, citește variabilele de configurare din fișier și permite folosirea acestora în script-ul curent.

```
if [[ ! -p $fifoServer ]]; then
```

Verifică dacă server-ul nu există, caz în care îl creeaza, *-p* verifică dacă un fișier este un FIFO (named pipe), necesar pentru comunicarea între doua procese care rulează independent pe sistemul de operare. În cazul de față, permite schimbul de mesaje între server și client. Față de un pipe obișnuit, un FIFO facilitează sincronizarea mesajelor: Clientul scrie, mesajul intră în FIFO, server-ul citește, preia mesajul și îl procesează. FIFO blochează scrierea sau citirea dacă unul dintre procese nu este pregătit, prevenind pierderea datelor.

```
                $fifoServer
fi
echo "Serverul este pornit pe $fifoServer..."
while true; do
```

Menține server-ul active pentru a procesa cereri continuu printr-o buclă infinită

```
    if read -r cerere <  $fifoServer; then
```

Citește cererea trimisă de client pe server, se folosește *–r* pentru a preveni interpretarea caracterelor speciale

```
        if [[ $cerere =~ BEGIN-REQ\ \[([0-9]+):\ ([a-zA-Z0-9_-]+)\]\ END-REQ ]]; then
```

Validarea și parsarea cererii, și extragerea PID-ului clientului și a numelui comenzii solicitate

```
            clientPid="${BASH_REMATCH[1]}"
            commandName="${BASH_REMATCH[2]}"
            clientFifo="/tmp/server-reply-$clientPid"
```

Extragerea variabelelor (PID-ul clientului, numele comenzii, FIFO-ul de reply pentru client

```
            echo "Cerere primita de la PID=$clientPid pentru comanda '$commandName'."
```

Afișarea pe ecran a confirmării înregistrării cererii

```
            if [[ ! -p $clientFifo ]]; then
```

Crearea FIFO-ului Clientului în cazul În care acesta nu există

```
                mkfifo $clientFifo
            fi
```

```
            man "$commandName" > "$clientFifo"
```
Rulează comanda man pentru comanda cerută și scrie rezultatul în FIFO-ul clientului

```
            rm -f $clientFifo
```
Șterge FIFO-ul clientului după ce răspunsul a fost trimis. Evită acumularea de fișiere temporare

```
            fi
        fi
    done
    rm $fifoServer
```
Șterge FIFO-ul server-ului la închiderea scriptului pentru a preveni fișierele rămase și termină execuția cu codul de succes 0.

```
    exit 0
```

1.client.sh

```
#!/bin/bash
source configServer.conf
```
Incarcă și execută conținutul fișierului configServer.conf în aceași sesiune de shell ca script-ul principal, astfel variabilele, funcțiile și alte resurse definite în fișier devin disponibile imediat în script-ul prinicipal, fără a fi nevoie de alta configurare

```
if [[ ! -p $fifoServer ]]; then
```
Verifică dacă FIFO-ul ($fifoServer) există și este valid, în caz contrar afișând un mesaj de eroare și terminând execuția script-ului.

```
    echo "Eroare: FIFO-ul serverului nu există!"
    exit 1
fi
clientPid=$$
```
Variabila specială $$ reține PID-ul (Process ID) procesului curent. este folosit pentru a crea un FIFO unic pentru acest client.

```
clientFifo="/tmp/server-reply-$clientPid"
mkfifo $clientFifo
```
definește un FIFO temporar, unic pentru client, folosind PID-ul său și îl creează cu mkfifo

```
echo "Introdu o comanda: "
read commandName
```
cere comanda de la utilizator

```
cerere="BEGIN-REQ [$clientPid: $commandName] END-REQ"
```
formateaza cererea ce urmeaza să fie trimisă server-ului

```
echo "$cerere" > $fifoServer
```

```
echo "Cererea a fost trimisă serverului: $cerere"
```

Trimite mesajul format în FIFO-ul serverului și afișează un mesaj de confirmare că cererea a fost trimisă.

```
cat $clientFifo
```

Scriptul citește răspunsul de la server din FIFO-ul personalizat al clientului ($clientFifo). Comanda cat blochează execuția până când serverul scrie un răspuns în FIFO.

```
rm -f $clientFifo
```

Șterge FIFO-ul personalizat pentru a evita fișierele temporare inutile.

```
exit 0
```

Rezultate experimentere și erori:

1.Crearea FIFO-ului serverului

FIFO-ul principal al serverului ($fifoServer) este creat pentru a permite clienților să trimită cereri către server. Dacă FIFO-ul există deja, scriptul îl reutilizează.

```
if [[ ! -p $fifoServer ]]; then
    mkfifo $fifoServer
fi
echo "Serverul este pornit pe $fifoServer..."
```

Eroare: FIFO blocat dintr-o rulare anterioară: Dacă un proces anterior a fost întrerupt fără a șterge FIFO-ul, acesta poate rămâne blocat și indisponibil pentru utilizare. În cazul dat, clienții rămân blocați sau afișează erori, iar serverul nu primește cereri.

Soluție: După verificare, FIFO-ul este creat doar dacă nu există, prevenind conflicte.

2. Verificarea existenței FIFO-ului serverului în client.sh

Clientul verifică dacă FIFO-ul serverului există înainte de a trimite o cerere, pentru a evita erori inutile.

```
if [[ ! -p $fifoServer ]]; then
    echo "Eroare: FIFO-ul serverului nu există!"
    exit 1
fi
```

Dacă serverul nu este pornit sau FIFO-ul a fost șters accidental, clientul nu poate trimite cereri. Clientul afișează eroarea și iese.

Eroare: Mesajul „Eroare: FIFO-ul serverului nu există!” apare frecvent.

Soluție: Clientul verifică existența FIFO-ului înainte de a trimite cererea, prevenind o eroare fatală.

3. Primirea și validarea cererii în server

Serverul citește cererea din FIFO-ul său și validează formatul acesteia utilizând o expresie regulată.

```
if read -r cerere < $fifoServer; then
   if [[ $cerere =~ BEGIN-REQ\ \[([0-9]+):\ ([a-zA-Z0-9_-]+)\]\ END-REQ ]]; then
fi
```

Eroare: Dacă formatul cererii este incorect, serverul poate ignora cererea sau poate funcționa eronat. Cererile invalide nu sunt procesate, iar clientul nu primește un răspuns.

Soluție: Expresia regulată validează cererile înainte de a le procesa, prevenind coruperea datelor.

4. Crearea FIFO-ului dedicat clientului

Serverul creează un FIFO personalizat pentru fiecare client, utilizând PID-ul acestuia pentru a genera un nume unic.

```
clientFifo="/tmp/server-reply-$clientPid"
if [[ ! -p $clientFifo ]]; then
   mkfifo $clientFifo
fi
```

Eroare: Dacă doi clienți folosesc același PID (posibil pe o perioadă lungă de rulare a serverului), FIFO-urile lor pot intra în conflict. Astfel, răspunsurile serverului pot fi amestecate între clienți, deci clienții primesc răspunsuri greșite sau incomplete.

Soluție: Numele FIFO-urilor sunt unice datorită includerii PID-ului, minimizând riscul de conflict.

5. Scrierea răspunsului în FIFO-ul clientului

Serverul scrie răspunsul generat de comanda man în FIFO-ul dedicat clientului.

```
man "$commandName" > "$clientFifo"
```

Eroare: Dacă clientul a ieșit înainte de a citi FIFO-ul, serverul poate rămâne blocat, deci serverul nu poate procesa alte cereri. Serverul pare blocat, fără a afișa erori.

Soluție: Serverul șterge FIFO-ul după utilizare.

6. Ștergerea FIFO-urilor după utilizare

Atât serverul, cât și clientul șterg FIFO-urile utilizate pentru a preveni acumularea de fișiere inutile.

```
rm -f $clientFifo
```

Eroare: Dacă scriptul este întrerupt înainte de a ajunge la ștergerea FIFO-ului, fișierul poate rămâne pe disc, cauzând conflicte la rulări ulterioare. În acest caz, sistemul poate deveni confuz din cauza existenței unor FIFO-uri vechi. Pot apărea mesaje de eroare legate de existența FIFO-urilor.

Soluție: Ștergerea explicită a FIFO-urilor minimizează acumularea acestora.

Documentație:

- https://docs.oracle.com/cd/E19455-01/805-7478/6j71mbrdh/index.html#:~:text=FIFOs%20are%20opened%20in%20the,apply%20when%20opening%20a%20FIFO.
- https://stackoverflow.com/questions/4113986/example-of-using-named-pipes-in-linux-shell-bash
- https://www.youtube.com/watch?v=9LMk2ydqMC0
- https://phoenixnap.com/kb/linux-source-command#:~:text=In%20Linux%20systems%2C%20source%20is,to%20be%20read%20and%20run.