Praktika: TFTP zerbitzari hobetua

Deskribapena:

Praktika honetan ikasleak “TFTP zerbitzaria” praktikan garatutako zerbitzaria hobetuko du. Zehazki, 3 hobekuntza egingo zaizkio:

1. Edozein tamainako fitxategiak transferitzeko aukera eman, datu blokeen identifikatzailea modu egokian kudeatuz.
2. Hainbat bezero aldi berean zerbitzatzeko aukera eman, bezero bakoitzarentzat prozesu ume bat sortuz.
3. Paketeak galera kudeatu, denboragailuak erabiliz eta, behar denean. birtansmisioak eginez.

Materiala:

Praktika hau egiteko ikasleak hainbat baliabide lagungarri izango ditu:

1. “TFTP zerbitzaria” praktikaren ebazpenari dagokion zerbitzariaren iturburu kodea (*tftp_zer_rrq_hobetua.py*). Hau da ikasleak aldatu beharko duen fitxategia.
2. data izeneko karpeta bat ausazko edukia duten hainbat fitxategirekin. Probak egiteko erabilgarriak dira.
3. TFTP bezero arrunt bat: “TFTP bezeroa” praktikaren ebazpena.
4. TFTP bezero berezi bat, zeinek, ausaz, bidali beharko lituzkeen mezu batzuk ez dituen bidaliko. Horrela, sarean gerta litezkeen pakete galerak simulatuko dira.

Aplikazio protokoloa

TFTP aplikazioaren protokoloa nahiko sinplea da. Bere lan egiteko modu nagusia eskolan azaldu da eta aurreko saio praktikoetan landu dugu. Xehetasun guztiak RFC 1350 dokumentuan aurki ditzakezu.

Zerbitzaria

RRQ eskaera zerbitzatuko duen zerbitzaria inplementatuta izango duzu, baina 3 hobekuntza gehitu beharko dizkiozu:

1. TFTP protokoloan bidaltzen diren bloke guztiak zenbaki batez identifikatzen dira. Zenbaki hau 16 bitez adierazten da, bai DATA bai ACK mezuetan. Fitxategi batek 2^{16} bloke baino gehiago behar baditu, zerbitzariak huts egingo du, ezingo duelako bloke zenbakia 16 bitetan adierazi. Kasu horretan, pakete kontagailua berrabiarazi beharko da, 0 balioa emanez. Arazoa ikusteko 32 MiB-eko fitxategi batekin proba daiteke (512 Byte x 2^{16} = 32 MiB), aurrerago azaltzen den moduan.
2. Eskuragarri duzun zerbitzariak bezero bat bakarrik zerbitza dezake une berean (50069 portua erabiliko du horretarako). Bezero bat zerbitzatzen ari denean RRQ eskaera bat jasoz gero, errore mezu batekin erantzungo dio. Zerbitzaria aldatu beharko duzu aldi berean nahi beste bezero zerbitza ditzan. Horretarako, 50069 portuan RRQ eskaera bat jaso orduko zerbitzariak prozesu berri bat sortu beharko du bezeroa zerbitza dezan. Prozesuen artean talkarik ez gertatzeko, prozesu ume bakoitzak bere socketa erabiliko du bezeroarekin komunikatzeko. Noski, socket honek ezingo du zerbitzari nagusiak darabilen portu bera

erabili, baina hori ez da arazo bat izango bezeroarentzat. Gogoratu TFTP protokoloak portu aldaketa hau aipatzen duela eta, beraz, bezeroa horretarako prest egongo dela.

3. TFTP protokoloak garraio mailan UDP darabilenez posible litzateke mezuren bat edo beste galtzea. Horregatik zerbitzariak DATA mezuak birtransmititu beharko ditu ez badu dagokion ACK mezua jasotzen esperotako denbora tartean (praktika honetarako ACK iritsi arte 0,5 segundo itxarotea proposatzen dizugu).

Bezeroa

Praktika honetarako bi bezero egongo dira eskuragarri. Lehenengoa, bezero arrunt bat izango da, “TFTP bezeroa” praktikaren ebazpena izango dena.

Bigarrena, aldiz, berezia izango da. Kontuan izan, bezeroa eta zerbitzaria makina berean egikaritzean bien arteko komunikazioan mezuak galtzea ia ezinezkoa dela. Horregatik, aurreko puntuan aipatutako 3. hobekuntza ondo dabilen probatzea ezinezkoa da bezero arrunt batekin. Arazo horri irtenbidea emateko, eskuragarri duzun bezero berezi honek noizean behin, ausaz, erantzun bat ez bidaltzea erabakiko du. Horrela, bezeroaren ACK mezua sarean galdu dela simulatzea lortuko dugu.

Lan egiteko dinamika

Funtseskoa da hurrengo pausoak banan bana ematea eta ariketa bakoitzaren ondoren ondo burutu dela ziurtatzea.

1. Sortu ausazko edukia duen 128 MiB-eko fitxategi bat data izeneko karpetan. Horretarako erabili honako komando hau:

   ```
   dd bs=1k count=128k if=/dev/urandom of=data/file128m
   ```

2. Jarri zerbitzaria martxan eta erabili bezero arrunta zerbitzariari fitxategi hori eskatzeko. Ikusi nola zerbitzarian errore bat gertatzen den bloke zenbakia ezin duelako kodetu 16 bitetan.

3. Aldatu zerbitzaria bloke kontagailua 2^{16} zenbakira iristean zerotik berrabiarazi dadin. Ez pasa hurrengo pausora ariketa ondo egin duzula ziurtatu arte. Horretarako errepikatu aurreko pausoa eta ziurtatu transferentzia ondo burutzen dela. Kontuan izan, bezeroak ere arazo bera duela eta, beraz, zerbitzarian egiten duzun aldaketa bera egin beharko duzula bezeroan ere.

4. Ireki 2 terminal leiho 2 bezero aldi batean martxan jarri ahal izateko. Lehenengoan bezero arrunta erabili 128 MiB-eko fitxategia zerbitzariari eskatzeko (aurreko pausoetan bezala). Transmisioak hainbat segundo iraungo ditu. Bitartean erabili bezero arrunta bigarren leihoan fitxategi txiki bat transmititzeko. Ikusi nola zerbitzariak bigarren eskaerari errore mezu batez erantzuten dion.

5. Aldatu zerbitzaria 50069 portuan jasotako RRQ eskaera bakoitzeko prozesu ume bat sor dezan. Prozesu umea bezero horrekin bakarrik komunikatzen dela ziurtatzeko, erabili connect funtzioa. Ez pasa hurrengo pausora ariketa ondo egin duzula ziurtatu arte. Horretarako errepikatu aurreko pausoa eta ziurtatu transferentziak ondo burutzen direla.

6. Erabili bezero berezia zerbitzariari edozein fitxategi eskatzeko eta ikusi nola batzuetan transferentzia blokeatzen den. Zenbat eta fitxategi handiagoa transferitzen saiatu orduan eta bloke gehiago transmititu beharko dira eta, beraz, altuagoa izango da transferentzia blokeatzeko probabilitatea.

7. Aldatu zerbitzaria prozesu umeek bezeroaren ACK mezua jasotzeko denbora maximo bat itxaron dezaten (0,5 segundo, adibidez). ACK ez bada iritsi epearen barruan, aurreko DATA mezua bidali beharko zaio berriro bezeroari. 3 saiakeraren ondoren ACK jaso gabe jarraituz

gero, transferentzia bertan behera utzi beharko da. Ariketa ondo egin duzula ziurtatu aurreko pausoa errepikatuz eta oraingoan blokeorik gertatzen ez dela ikusiz.

“tftp_zer_rrq_hobetua.py” fitxategia:

```
#!/usr/bin/env python3

import sys
import os
import socket
import signal
import select

NULL  = b'\x00'
RRQ   = b'\x00\x01'
WRQ   = b'\x00\x02'
DATA  = b'\x00\x03'
ACK   = b'\x00\x04'
ERROR = b'\x00\x05'

PORT = 50069
BLOCK_SIZE = 512
FILES_PATH ='./data/'

def send_error(s, addr, code, message):
      resp  = ERROR
      resp += code.to_bytes(2, 'big')
      resp += message.encode()
      resp += NULL
      s.sendto(resp, addr)

def send_file(s, addr, filename):
      try:
            f = open(os.path.join(FILES_PATH, filename), 'rb')
      except:
            send_error(s, addr, 1, 'File not found.')
            exit(1)

      """IKASLEAK BETETZEKO: [Enuntziatuko 5. pausoa]
      "Konektatu" socketa bezeroarekin.
      Erabili send funtzioa sendto funtzioaren ordez prozesu umean.
      Erabili recv funtzioa recvfrom funtzioaren ordez prozesu umean.
      """
      data = f.read(BLOCK_SIZE)
      resp  = DATA
      resp += b'\x00\x01'
      resp += data
      s.sendto(resp, addr)

      block_num = 1
      last = False if len(data) == BLOCK_SIZE else True
      while True:
            """IKASLEAK BETETZEKO: [Enuntziatuko 7. pausoa]
            Ziurtatu bezeroaren mezua iritsi dela 500 ms-ren barruan. Bestela,
birbidali azkeneko mezua.
            3 aldiz saiatu ondoren ez badago bezeroaren berririk, amaitu prozesu
umea.
            """
            resp, cli_addr = s.recvfrom(64)
            opcode = resp[:2]
            if opcode == ERROR:
                  error_code = int.from_bytes(resp[2:4], 'big')
                  print('Server error {}: {}'.format(error_code, resp[4:-
```

```
1].decode()))
                exit(1)
          elif opcode != ACK:
                print('Unexpected response.')
                exit(1)
          else:
                ack_num = int.from_bytes(resp[2:4], 'big')
                if ack_num != block_num:
                      continue
                if last:
                      break
                """IKASLEAK BETETZEKO: [Enuntziatuko 3. pausoa]
                block_num aldagaia 2**16 baliora iristean, 0 balioa esleitu.
                """
                block_num += 1
                data = f.read(BLOCK_SIZE)
                resp  = DATA
                resp += block_num.to_bytes(2, 'big')
                resp += data
                s.sendto(resp, addr)
                if len(data) < BLOCK_SIZE:
                      last = True


      print('{}: {} bytes sent.'.format(filename, (block_num - 1) * BLOCK_SIZE +
len(data)))
      f.close()

if __name__ == '__main__':
      """IKASLEAK BETETZEKO: [Enuntziatuko 5. pausoa]
      Deitu signal funtzioari zombie prozesuak ekiditeko.
      """
      s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
      s.bind(('', PORT))

      while True:
            req, cli_addr = s.recvfrom(64)

            opcode = req[:2]
            if opcode != RRQ:
                  send_error(s, cli_addr, 5, 'Unexpected opcode.')
            else:
                  filename, mode, _ = req[2:].split(b'\x00')
                  if mode.decode().lower() not in ('octet', 'binary'):
                        send_error(s, cli_addr, 0, 'Mode unkown or not
implemented')
                        continue
            filename = os.path.basename(filename.decode()) # For security,
filter possible paths.
            """IKASLEAK BETETZEKO: [Enuntziatuko 5. pausoa]
            Sortu prozesu ume bat socket berri bat sortu eta send_file
funtzioari deitzeko.
            Ez ahaztu prozesu umea hiltzen bere lana amaitutakoan.
            """
            send_file(s, cli_addr, filename)
```

Ariketa osagarriak

(1) Bezeroak azkeneko ACK bidaltzen duenean, socketa itxi eta programa amaitzen du. Kasu horretan azkeneko ACK hori galtzen bada eta zerbitzariak dagokion DATA mezua birtransmititzen badu, errore bat suertatuko da. Aldatu zerbitzaria errore hori kudeatzeko, mezu bat pantailaratuz eta prozesu umea modu ordenatuan itxiz. Ariketa osagarri hau probatzeko ziur aski egokiena bezeroa ere moldatzea izango da, azkeneko pakete hori bidal

ez dezan. Materialean aurkituko duzun bezeroa programatua dago hain zuzen azkeneko ACK hori inoiz ez galtzeko, aurreko ariketetan arazorik ez sortzeko.