**Final Lab Exam**

**Instructions**

* You are allowed to form groups of 2 students.
* The code should be developed within the team and any form of plagiarism will automatically result in zero for both the groups involved without any questions asked.
* Each student is responsible to understand the code being submitted under his or her name. Division of work is not explicitly required, however, each group member will be given average viva marks of the group. This implies that if a group member gets zero in viva and another gets full marks then both will score 50%.
* The submission deadline is midnight before next lab. Assessment will be done based on presentation and live demo.
* Any questions and comments on the lab must reach the lab engineer no later than a day before the deadline through email or in person meeting. No questions will be answered later.

**Introduction**

The purpose of this activity is for you to:

(1) Build the **server for the file management component of distributed file management system** you built in previous labs. The goal is to build the structure for file management to provide access to user(s) to create, delete, update and query files in the system remotely by using threads. You are also free to make the design choices as discussed in book or a combination thereof as long as the requirements for the system is met.

(2) Convert the above client-server system into a **multi-user system with proper synchronization** applied to it. In this task we will protect the files from **reader writer problem**. The description of reader-writer problem is described in section 7.1.2 of your textbook, Operating System Concepts, 10th Edition by Avi Silberschatz et al. You may use monitors, locks or semaphore libraries provided by the language APIs to implement the tasks.

You are allowed to build the system in the language of your choice and in the operating system of your choice as long as the objective of the lab are met.

**Objectives**

By the end of this exercise you will be able to build a server to provide file structure and understand the operations on how to make a server by using threads and socket programming. Also you will learn practical uses of synchronization and implement it to solve reader-writer problem.

**Tasks**

**Task 1**

Before you start your main task, go through the links below to learn how to make a server. You can use code from the following links to understand what a server is and how it works. The codes (in different languages) as a sample to implement socket programming for sending and receiving simple messages is given below.

* Socket programming in Python <https://www.journaldev.com/15906/python-socket-programming-server-client>
* Socket programming in Java <https://www.journaldev.com/741/java-socket-programming-server-client>
* Socket programming in C++ <https://www.educba.com/socket-programming-in-c-plus-plus/>

This task has two further subtasks which are interrelated.

1. **Design a protocol** on top of your previous work in i.e., File Management and Threaded File Management System. A protocol is a pre-defined system of rules to exchange information. Here the client will be calling functions at the server. That means that **server is the file server you implemented and the client will be remotely executing the tasks on the server machine**. A protocol may mean that you assign numbers to each of the function and use some sort of symbols to separate the parameters or some other scheme. You may read some of the common protocols such as POP3 or SMTP to understand how a protocol work.
2. **Implement this protocol through server and provide a client program to the user**. The **server program** will receive the message through socket, execute the task and return the response. The **client side program** will provide an interface to the user where different operations can be executed and results shown.

**Task 2**

This task has two subtask as well however you may choose and implement **any one of the two.** The task here is to implement mutual exclusion of file access. Multiple threads can attempt to access your files but synchronization can be achieved by

1. **Implement a queue for readers**. Multiple threads can attempt to read a file and some can request a write.
2. Your task is to make sure that while the file is being read, no writer is allowed while maintaining the order of writes.
3. Any request to open the file in write mode is considered to be in active writing till the user closes the file.
4. Similarly any request to open the file in read mode is considered to be in active read till the user closes the file.
5. Multiple users can read the file concurrently but writes will be mutually exclusive.
6. **Implement a limit on the user name to access a file.**
7. Multiple users can access your system.
8. Though it is not required that you implement any type of security, but any user may not be able to access more than 5 files at one time.
9. If more than 5 requests are placed then the requesting thread must wait.
10. Each file can only be accessed by 3 users, be it for read or write.

**Requirements**

* + - 1. You must provide two programs, a server and a client.
      2. The client must allow the user to specify the IP address of the server at the time of connection.
      3. The client will allow user to first specify the user name at the connection setup time and all communication should display it.
      4. The client must provide an interface to apply the operations developed in the previous labs.
      5. The client must give errors when the server is not available.
      6. The client must display the response of the actions performed.
      7. The server must respond to multiple requests at the same time (this will require threads)
      8. The server must bind to port 95 whereas the client port can be any number higher than 1024.
      9. The server and clients can run on different machines.

**Deliverables**

1. Complete code
2. Sample data file (sample.dat) consisting of files and directories to show your output and memory map.
3. Viva:
   1. Brief about your implementation of the lab
   2. Development of protocol and establishment of client-server system
   3. Synchronization primitives used and implementation details and reason of your choice.
   4. Testing and debugging

|  |
| --- |
| **CODE:** |
| **filesystem.py** |
| import os  import threading  import sys  import json  import pickle  filesInUse = []  #get current working directory of process  root = os.getcwd()  global datdict  datdict = {}  #to avoid concurrent modifies  lock = threading.Lock()  global clientInfo  clientInfo = {      'message': ''  }  def thread\_function(user\_response):      user\_response = user\_response.strip()      user\_response = user\_response.split(' ')      if user\_response[0] == 'create':          createFile(user\_response[1])      elif user\_response[0] == 'delete':          deleteFile(user\_response[1])      elif user\_response[0] == 'makeDir':          makeDirectory(user\_response[1])      elif user\_response[0] == 'changeDir':          changeDirectory(user\_response[1])      if user\_response[0] == 'open':          user\_response[1] = user\_response[1].replace("<", "")          user\_response[1] = user\_response[1].replace(">", "")          user\_response[1] = user\_response[1].replace("\n", "")          args = user\_response[1].split(',')          openFile(args[0], args[1])      elif user\_response[0] == 'read\_from\_file':          user\_response[1] = user\_response[1].replace("<", "")          user\_response[1] = user\_response[1].replace(">", "")          user\_response[1] = user\_response[1].replace(",", "")          user\_response[1] = user\_response[1].replace("\n", "")          user\_response[2] = user\_response[2].replace(",", "")          user\_response[3] = user\_response[3].replace("\n", "")          openFile(user\_response[1], 'x', '', user\_response[2], user\_response[3])      elif user\_response[0] == 'close':          user\_response[1] = user\_response[1].replace("<", "")          user\_response[1] = user\_response[1].replace(">", "")          user\_response[1] = user\_response[1].replace("\n", "")          args = user\_response[1].split(',')          closeFile(args[0])      elif user\_response[0] == 'write\_to\_file':          user\_response[2] = ' '.join(user\_response[2:])          user\_response[1] = user\_response[1].replace("<", "")          user\_response[1] = user\_response[1].replace(">", "")          user\_response[1] = user\_response[1].replace("\n", "")          user\_response[1] = user\_response[1].replace(",", "")          user\_response[2] = user\_response[2].replace(",", "")          user\_response[2] = user\_response[2].replace("\n", "")          openFile(user\_response[1], 'w', user\_response[2])      elif user\_response[0] == 'write\_at':          user\_response[1] = user\_response[1].replace("<", "")          user\_response[1] = user\_response[1].replace(">", "")          user\_response[1] = user\_response[1].replace("\n", "")          user\_response[1] = user\_response[1].replace(",", "")          more\_text = a[2:-1]          user\_response[3] = user\_response[3].replace("\n", "")          user\_response[3] = user\_response[3].replace(",", "")          user\_response[4] = user\_response[4].replace("\n", "")          more\_text = ' '.join(more\_text)          openFile(user\_response[1], 'w', more\_text, user\_response[-1])      elif user\_response[0] == 'truncate\_file':          truncateFile(user\_response[1], user\_response[2])      elif user\_response[0] == 'mov':          moveFile(user\_response[1], user\_response[2])      elif user\_response[0] == 'show\_mmap':          showDat()      elif user\_response[0] == 'exit':          sys.exit(0)      else:          clientInfo  def showDat():      readDat()      clientInfo["message"] = ""      for key in datdict:          clientInfo["message"] += key + " " + datdict[key] + "\n"  def saveDat():      file = open(root+"/" + "dat.dat", "w")      for key in datdict:          file.write(key + "#" + datdict[key] + "\n")      file.close()  def writeDat(filename):      readDat()      file = open(filename, "r")      content = file.read()      datdict[filename] = content      saveDat()  def readDat():      if os.path.isfile(root+"/"+"dat.dat"):          file = open(root+"/"+"dat.dat", "r")          for line in file:              if '#' in line:                  (key, val) = line.split('#', 1)                  datdict[key] = val          file.close()      else:          file = open(root+"/"+"dat.dat", "w")          file.close()  def createFile(filename):      file = open(filename, "w")      file.close()      writeDat(filename)      clientInfo["message"] = 'File ' + filename + ' created successfully!'  def deleteFile(filename):      print(filesInUse)      if filename not in filesInUse:          datdict.pop(filename)          print(filename)          os.remove(filename)          clientInfo["message"] = 'File ' + filename + ' deleted successfully!'      else:          clientInfo["message"] = 'File ' + filename + ' is in use!'  def makeDirectory(directory):      if os.path.isdir(directory):          print("Directory already exists!!!")          clientInfo["message"] = "Directory already exists!!!"      else:          directoryName = root + "/" + directory          print(directoryName)          clientInfo["message"] = "Directory created successfully --> " + directoryName          os.mkdir(directoryName)  def changeDirectory(directory):      if directory == "..":          directoryName = root      else:          directoryName = root + "/" + directory          directoryName = directoryName.replace("\n", "")      print(directoryName)      clientInfo["message"] = "Directory changed successfully!"      if os.path.isdir(directoryName):          os.chdir(directoryName)          clientInfo["message"] = "Directory changed successfully to " + directoryName      else:          print("Directory does not exist!")          clientInfo["message"] = "Directory does not exist!"  def moveFile(filename, destination):      lock.acquire()      if os.path.exists(filename):          if filename in filesInUse:              print("File is in use, cannot be moved!")              clientInfo["message"] = "File is in use, cannot be moved!"          else:              directoryName = root + "/" + destination              if os.path.isdir(directoryName):                  os.rename(filename, directoryName + "/" + filename)                  clientInfo["message"] = "File moved successfully to " + \                      directoryName              else:                  print("Directory does not exist!")                  clientInfo["message"] = "Directory does not exist!"      else:          print("File does not exist!")          clientInfo["message"] = "File does not exist!"      lock.release()  def openFile(filename, mode, content='', startingIndex=0, size=0):      fileName = filename      if os.path.exists(filename):          match mode:              case 'w':                  if filename not in filesInUse:                      file = open(filename, "r")                      filesInUse.append(file)                      contents = file.read()                      filesInUse.remove(file)                      file.close()                      contents = str(contents)                      if int(startingIndex) > len(contents):                          startingIndex = len(contents)                      contents = contents[:int(startingIndex)] + \                          content + contents[int(startingIndex):]                      file = open(filename, "w")                      filesInUse.append(file)                      if (startingIndex == 0):                          file.write(content)                      else:                          file.write(contents)                      if filename in filesInUse:                          filesInUse.remove(filename)                      file.close()                      clientInfo["message"] = "File written to successfully!"                  else:                      print("File is in use, cannot be written to!")                      clientInfo["message"] = "File is in use, cannot be written to!"              case 'r':                  filename = open(filename, "r")                  filesInUse.append(fileName)                  print("Contents of " + fileName +                        ": " + filename.read())                  clientInfo["message"] = "Contents of " + \                      fileName + ":\t" + filename.read()                  filename.close()              case 'x':                  file = open(filename, "r")                  filesInUse.append(file)                  index = startingIndex                  length = size                  contents = file.read()                  print(contents[int(index):int(index) + int(length)])                  clientInfo["message"] = contents[int(                      index):int(index) + int(length)]                  file.close()              case \_:                  print("Invalid mode!")                  clientInfo["message"] = "Invalid mode!"          writeDat(fileName)      else:          print("File does not exist!")          clientInfo["message"] = "File does not exist!"  def truncateFile(filename, size):      if os.path.exists(filename):          fileSize = os.path.getsize(filename)          if int(size) < fileSize:              file = open(filename, "r+")              filesInUse.append(file)              file.truncate(int(size))              filesInUse.remove(file)              file.close()              writeDat(filename)          else:              print("File is already smaller than the size you want to truncate to!")              clientInfo["message"] = "File is already smaller than the size you want to truncate to!"      else:          print("File does not exist!")          clientInfo["message"] = "File does not exist!"  def closeFile(filename):      if filename in filesInUse:          filesInUse.remove(filename)      else:          print("File is not open!")          clientInfo["message"] = "File is not opened!"  def memoryMap():      arr = {}      #os.walk() creates tree structure of the directory      for root, dirs, files in os.walk('.', topdown=False):          for name in files:              file = open(os.path.join(root, name), "r")              arr[os.path.join(root, name)] = hex(id(file))          for name in dirs:              if os.path.isdir(os.path.join(root, name)) == False:                  file = open(os.path.join(root, name), "r")                  arr[os.path.join(root, name)] = hex(id(file))          clientInfo["message"] = ''          for key, value in arr.items():              print(key + ':\t' + value+'\n')              clientInfo["message"] += key + ':\t' + value+'\n'  if \_\_name\_\_ == "\_\_main\_\_":      filenames = ['fileOfText.txt','siraj.txt','file1.txt']      nThreads = int(sys.argv[1])      print("Threads: " + str(nThreads))      for i in range(nThreads):          f = open(filenames[i], "r")          lines = f.readlines()          t = threading.Thread(target=thread\_function, args=[i, lines])          t.start() |
| **Client.py** |
| import socket  print("================================================================================")  print('                        WELCOME TO THE DISTRIBUTED FILE SYSTEM                  ')  print("================================================================================")  print("This remote file system supports the following commands;   \        the command format is also specified with each command.\n")  print("1. Create a file: 'create file\_name.txt'\n \      2. Delete a file: 'delete file\_name.txt'\n \      3. Create a directory: '' mkdir dir\_name'\n \      4. Change the directory : 'changeDir dir\_name'\n \      5. Move a file: 'mov file\_name.txt dir\_name'\n \      Note: The file modes are 'w' for write and 'r' for read.\n \      6. Open a file: 'open <file\_name.txt,mode>'\n \      7. Close a file: 'close <file\_name>'\n \      8. Write to a file: 'write\_to\_file <file\_name.txt>,string\_to\_write'\n \      9. Read from a file: 'read\_from\_file <file\_name.txt>,starting\_position,end\_position'\n \      Note: to read whole file use starting\_position as 0 and end\_position as -1.\n \      10. Truncate a file: 'truncate\_file file\_name.txt position'\n \      11. Show memory Map: 'show'\n \      12. To exit the system enter: 'BYE' (We'll be sorry to see you go :(\      That's all folks, enjoy using the distributed file system! \n")  print("The client is ready to connect!\n")  while True:      #host = input('Enter the IP address of the server:  ')      host = 'localhost'      port = 95      ClientSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)      try:          ClientSocket.connect((host, port))          print('Connection Successful!!')          break      except socket.error as e:          print("The IP you have entered might be INCORRECT OR Server NOT running(It's not you, it's US.)")  while True:      username = input("Please enter your Username :  ")      ClientSocket.send(str.encode(username))      Response = ClientSocket.recv(2048)      if Response.decode('utf-8') != 'This username is taken :(':          break      print(Response.decode('utf-8'))  print(Response.decode('utf-8'))  while True:      Input = input('Operation: ')      if Input == 'EXIT':          ClientSocket.send(str.encode(Input))          break      ClientSocket.send(str.encode(Input))      Response = ClientSocket.recv(2048)      print(Response.decode('utf-8'))  ClientSocket.close() |
| **Server.py** |
| import socket  from \_thread import \*  import filesystem as fs  from filesystem import clientInfo  #creating TCP Socket  #sock = socket.socket(socket.AF\_NET, socket.SOCK\_STREAM)  #get host name on which the server is being executed.  #host = socket.gethostname()  host = 'localhost'  #gets the IP address using the host name  print(socket.gethostbyname(host))  port = 95  #bind the socket  #server\_address = (socket.gethostbyname(host), port)  #sock.bind(server\_address)  ThreadCount = 0  usernames = ['admin']  def client\_handler(connection):      while True:          data = connection.recv(2048)          username = data.decode('utf-8')          print('Please enter your Username :  ' + username)          if username not in usernames:              break          connection.send(str.encode('Username is taken'))      usernames.append(username)      print(username + ' is sucessfully connected!')      connection.send(str.encode(username + ' is sucessfully connected!'))      while True:          data = connection.recv(2048)          message = data.decode('utf-8')          fs.thread\_function(message)          print(f"{username} ==> {clientInfo['message']}")          if message == 'EXIT':              print(username + ' is disconnected :(!')              break          reply = f"{username} ==> {clientInfo['message']}"          connection.sendall(str.encode(reply))      usernames.remove(username)      connection.close()  def accept\_connections(ServerSocket):      Client, address = ServerSocket.accept()      print('Connected to socket ==> ' + address[0] + ':' + str(address[1]))      start\_new\_thread(client\_handler, (Client, ))  def start\_server(host, port):      #AF\_INET specifies that the internet address is IPv4.      #SOCK\_STREAM specifies that the socket is TCP.      ServerSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)      try:          ServerSocket.bind((socket.gethostbyname(host), port))      except socket.error as e:          print(str(e))      print(f'\n ----Server is listening on the port {port}-----')      #limiting the maximum nmuber of client connections to 5.      ServerSocket.listen(5)      while True:          accept\_connections(ServerSocket)  start\_server(host, port) |

|  |
| --- |
| **OUTPUT Snapshots:** |
| **Client 1:** |
|  |
| **Client 2:** |
|  |
| **Server:** |
|  |

**Evaluation** inNext Lab

Viva: 10 mins