Advance Operating System Lab (CSD-416)
National Institute of Technology, Hamirpur

ADVANCE OPERATING SYSTEM LAB (CSD-416) ASSIGNMENT 6

Lamport's ME Algorithm and Richart Agarwala's Algorithm

AKSHAT RAJ VANSH (185520)

December 11, 2021

Computer Science Department
National Institute of Technology, Hamirpur

Contents

1 Lamport's Mututal Exclusion Algorithm

1.1 Code

```
import sys
import threading
import time
from random import randint

no_threads = 5
no_requests = 5

print("Threads =", no_threads)
print("Requests =", no_requests)

distribution = []
no = 0
for itr in range(0, no_threads):
    distribution.append(0)
for itr in range(0, no_requests):
    distribution[no] += 1
    no = (no + 1) % no_threads

requestno = 0
y = -1
x = -1
b = []
for itr in range(0, no_threads):
    b.append(0)


def thread_lamport_fast(threadno):
    global distribution
    global requestno
    global no_requests
    global no_threads
    global y
    global x
    global b
    reqs = distribution[threadno]
    time.sleep(1)

    while reqs:
        print(f"Process {threadno} Requesting to Access Critical Section")
        b[threadno] = 1
        x = threadno
        if y != -1:
            b[threadno] = 0
            while (y != -1):
                pass
            continue
        y = threadno

        if x != threadno:
            b[threadno] = 0
            for j in range(0, no_threads):
                while (b[j] != 0):
                    pass
```

```
            if y != threadno:
                while (y != -1):
                    pass
                continue
        requestno = requestno + 1
        print(f"Process {threadno} Entering Critical Section")
        reqs = reqs - 1
        print(f"Process {threadno} Exiting Critical Section")
        y = -1
        b[threadno] = 0


print("Running Lamport's fast mutual exclusion algorithm:")
for threadno in range(0, no_threads):
    th = threading.Thread(target=thread_lamport_fast, args=(threadno,))
    th.daemon = True
    th.start()

while (requestno < no_requests):
    pass
time.sleep(1)
print("Finishing Lamport's fast mutual exclusion. All requests served. requestno =", requestno, "\n\
time.sleep(20)
```

1.2 Output

Lamport's Mututal Exclusion Algorithm Output

```
Threads = 5
Requests = 5
Running Lamport's fast mutual exclusion algorithm:
Process 1 Requesting to Access Critical Section
Process 1 Entering Critical Section
Process 2 Requesting to Access Critical Section
Process 0 Requesting to Access Critical Section
Process 1 Exiting Critical Section
Process 3 Requesting to Access Critical Section
Process 4 Requesting to Access Critical Section
Process 4 Entering Critical Section
Process 2 Requesting to Access Critical Section
Process 0 Requesting to Access Critical Section
Process 3 Requesting to Access Critical Section
Process 4 Exiting Critical Section
Process 0 Requesting to Access Critical Section
on
Process 0 Entering Critical Section
Process 0 Exiting Critical Section
Process 3 Requesting to Access Critical Section
Process 2 Requesting to Access Critical Section
Process 3 Entering Critical Section
Process 3 Exiting Critical Section
Process 2 Requesting to Access Critical Section
Process 2 Entering Critical Section
Process 2 Exiting Critical Section
Finishing Lamport's fast mutual exclusion. All requests served. requestno = 5
```

2 Richart Agarwala's Algorithm

2.1 Code - Server

```
import socket as socket
import _thread
import threading
from datetime import datetime


class Server:

    def __init__(self, port, host="",):
        self.host = host
        self.port = port
        self.connection = []
        self.allocated = ""
        self.timestamp = 0
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    def configure(self):
        try:
            self.server.bind((self.host, self.port))
            print("Server binded to port", self.port)
            self.server.listen(5)
            print("Server is listening")
        except Exception as e:
            print(e)

    def decode(self, value):
        return value.decode('ascii')

    def encode(self, value):
        return value.encode('ascii')

    def broadcast(self, client, message):
        for x in self.connection:
            if x != client:
                x.send(self.encode(message))

    def threaded(self, client, client_addr, client_name):
        while True:
            try:
                data = client.recv(1024)
                if not data or str(self.decode(data)) == "./leave":
                    self.connection.remove(client)
                    break
                data = str(self.decode(data))
                command = data[0:data.find("~")]
                if command == "REQUEST":
                    if self.allocated == "":
                        self.allocated = client_name
                        self.timestamp = int(data[data.find("~")+1:])
                        client.send(self.encode("REPLY"))
                    else:
                        print("Timestamp of {} is {}".format(
                            client_name, datetime.fromtimestamp(int(data[data.find("~")+1:]))))
                        print("Timestamp of {} is {}".format(
```

```
                            self.allocated, datetime.fromtimestamp(self.timestamp)))
                        print("Resource cannot be allocated to {}".format(
                            client_name))

                if command == "REPLY":
                    if self.allocated == client_name:
                        self.allocated = ""
                        self.timestamp = 0
                        self.broadcast(client, "REPLY")
                        print("Resource is released")

            except Exception as e:
                print(e)
                break
        client.close()

    def start(self):
        self.configure()
        while True:
            client, client_addr = self.server.accept()
            client_name = client.recv(1024)
            self.connection.append(client)
            print('Connected to :', client_addr[0], ':', client_addr[1])

            _thread.start_new_thread(
                self.threaded, (client, client_addr, client_name.decode('ascii')))


if __name__ == '__main__':
    server = Server(1236)
    server.start()
print(e)
```

2.2 Code - Client

```
import socket
import json
import _thread
import time
import threading


class Error:
    commandInputError = Exception("Please enter correct command")
    portInputError = Exception("Please enter correct port number")
    controllerError = Exception("Controller Error. Try After Sometime")
    createRoomError = Exception("Error in creating the room")


class Client:
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.flag = 0

    def createSocket(self, port):
        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        client.connect((self.host, port))
        return client

    def decode(self, value):
        return value.decode('ascii')

    def encode(self, value):
        return value.encode('ascii')

    def listen(self, client):
        while True:
            try:
                data = client.recv(1024)
                if not data:
                    break
                print(self.decode(data))
            except:
                continue
        client.close()
        exit(0)

    def listen_10(self, client):
        client.settimeout(0.1)
        vr = False
        for i in range(100):
            try:
                data = client.recv(1024)
                if(self.decode(data) == "REPLY"):
                    print("Request Granted")
                    vr = True
                    exit(0)
            except:
                continue
```

```
57          if(vr == False):
58              print("Request Denied")
59              exit(0)
60
61      def send(self, client):
62          while True:
63              try:
64                  message = input("")
65                  if(message == "REQUEST"):
66                      timestamp = time.time()
67                      message = message + "~" + str(int(timestamp))
68                      _thread.start_new_thread(self.listen_10, (client,))
69                  else:
70                      message = message + "~"
71              except Exception as e:
72                  continue
73              client.send(self.encode(message))
74
75      def start(self):
76          client = self.createSocket(self.port)
77          name = input("Enter Name : ")
78          client.send(name.encode('ascii'))
79
80          _thread.start_new_thread(self.send, (client,))
81          # _thread.start_new_thread(self.listen, (client,))
82          while True:
83              continue
84
85
86  if __name__ == '__main__':
87      client = Client('127.0.0.1', 1236)
88      client.start()
```

2.3 Output

Richart Agarwala's Algorithm Server Output

```
Server binded to port 1236
Server is listening
Connected to : 127.0.0.1 : 55658
Connected to : 127.0.0.1 : 55670
Timestamp of Raj is 2021-12-11 14:00:34
Timestamp of Akshat is 2021-12-11 14:00:31
Resource cannot be allocated to Raj
Resource is released
```

Richart Agarwala's Algorithm Client 1 Output

```
Enter Name : Akshat
REQUEST
Request Granted
REPLY
```

Richart Agarwala's Algorithm Client 2 Output

```
Enter Name : Raj
REQUEST
Request Granted
REPLY
```