Operating Systems

Final Project

Princess Sumaya University for Technology
Computer Science Department
Operating Systems
Course Number: 11335

Submitted to: Dr. Mustafa Al-Fayoumi

Submitted By: Sewar Khalifeh

ID No. : 20160147

9.5.2019

Introduction:

This project presents two of the most important topics in operating systems, The first on is threading and how it happens on a single or multicore processing system. The second topic is Communications in Client-Server Systems using sockets.

1) Threads:

A thread is a path of execution within a process. **A process can contain multiple threads.** Threads are very useful in modern programming whenever a process has multiple tasks to perform independently of the others.

- As shown in Figure 4.1, **multi-threaded** applications have multiple threads within a single process, each having their own program counter, stack and set of registers, but sharing common code, data, and certain structures such as open files.

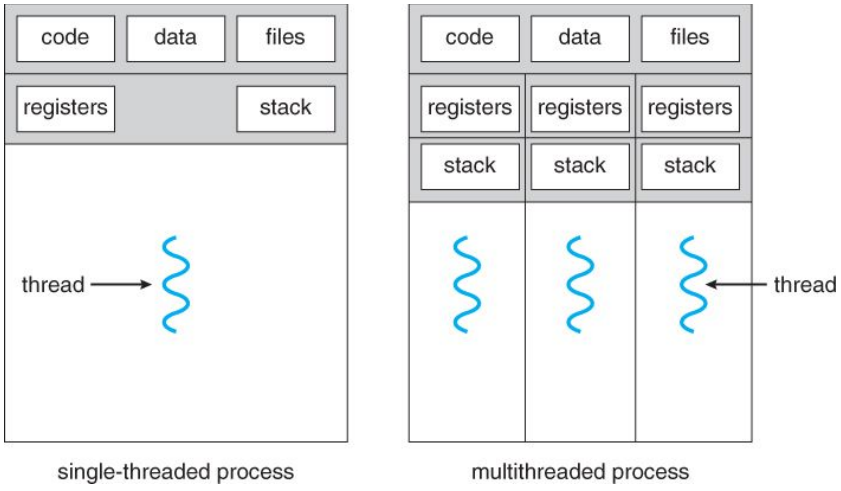

Figure 4.1 - Single-threaded and multithreaded processes

- A computer architecture can have multiple ***cores***, or CPUs on a single chip.
- A multi-threaded application running on a traditional single-core chip would have to interleave the threads, as shown in Figure 4.2. On a multi-core chip, however, the threads could be spread across the available cores, allowing true parallel processing, as shown in Figure 4.3.

Figure 4.2 - Concurrent execution on a single-core system.

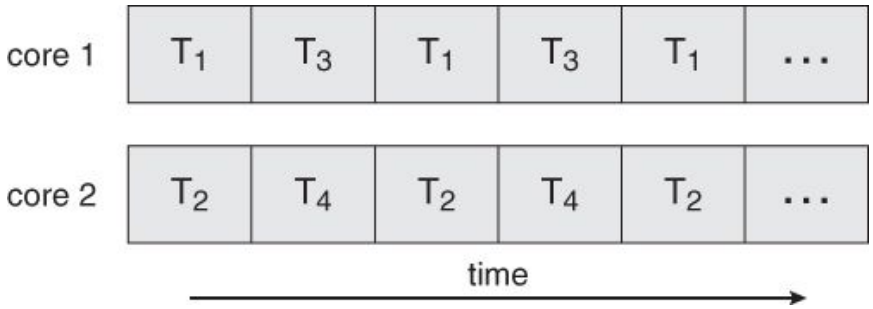

Figure 4.3 - Parallel execution on a multicore system

2) Process Communication using Sockets:

Socket can be defined as the end point of communication, it's the concatenation of the IP address and a port. A communication link can happen between two sockets which has to be unique.

A diagram that illustrates sockets is as follows:

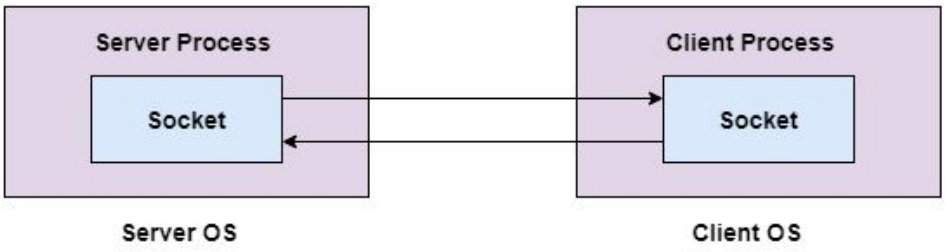

Figure 4.4 - Client-Server Communication using sockets

Procedure for Question 1:

1) A Python program to create a thread that calculates the factorial for two large numbers entered by the user and print the summation and the time interval for the factorial function calculation.

   Single thread code:

```python
import time , threading

def factorial(x):
    a = 1
    while x >= 1:
        a = a * x
        x = x - 1
    return a


num1 = int(input("Enter the first number: \n"))
num2 = int(input("Enter the second number: \n"))

a = time.time()
print("sum of factorials= \n",factorial(num1)+factorial(num2))
b = time.time()

print("time consumed in single thread:\n",b-a)

```

Same program but using **multi-threading:**

```
# -*- coding: utf-8 -*-
"""
Created on Thu May  9 17:04:10 2019

@author: khali
"""
import queue

import time , threading

def factorial(x):
    a = 1
    while x >= 1:
        a = a * x
        x = x - 1
    return a
Q = queue.Queue()

y=0
num1 = int(input("Enter the first number: \n"))
num2 = int(input("Enter the second number: \n"))
a = time.time()
t1 = threading.Thread(target=lambda q, arg1: q.put(factorial(arg1)), args=(Q, num1))
t2 = threading.Thread(target=lambda q, arg1: q.put(factorial(arg1)), args=(Q, num2))
t1.start()
t2.start()
t1.join()
t2.join()
b = time.time()
while not Q.empty():
    result = Q.get()
    y=y+result
print (y)

print("time consumed in multithreading:\n",b-a)

```

Results for Question 1:

I applied the previous code on two numbers : 100000 and 50000, the results in time was as the following screenshots show (summation results were very large numbers):

For single thread:

```
time consumed in single thread:
 13.73292064666748
```

For Multithreading:

```
time consumed in multithreading:
 6.583444595336914
```

Procedure for Question 2:

Develop a Chatting Application in any programming language that works in a client-server environment satisfying the following requirements:

1.Client read a line of characters (data) from its keyboard and send the data to the server.

2.The server receives the data and converts characters to uppercase.

3.The server sends the modified data to the client.

4.The client receives the modified data and displays the line on its screen.

The following codes were written in **Python**

```
4
5 @author: sewar khalifeh
6 Socket programming with UDP for Client Server instant communication
7 Transfering through UDP Model is unreliable but fast"""
8
9 import socket #include Python's socket library
10 serverPort = 19999
11 clientSocket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
12 message = input(str('Input lowercase sentence:')) #get user input
13 clientSocket.sendto(message,serverPort) #Attach server port to message; send to server socket
14 newMessage, serverAddress = clientSocket.recvfrom(2048)
15 print (newMessage)
16 clientSocket.close()
17
```

Client Code

```
4
5 @author: Sewar khalifeh
6 Socket programming with UDP for Client Server instant communication
7 Transfering through UDP Model is unreliable but fast
8 """
9
10 from socket import *
11 serverPort = 19999
12 serverSocket = socket(AF_INET, SOCK_DGRAM) #UDP socket
13 serverSocket.bind(('', serverPort)) #bind socket to local port 19999
14 print ("server ready")
15 while True:
16  message, clientAddress = serverSocket.recvfrom(2048)
17  newMessage = message.upper() #converts to uppercase
18  serverSocket.sendto(newMessage, clientAddress) #send upper case  back to client
```

Server Code

Result for Question 2:

By entering the string “os project” in the client process, the server returned the upper-case of it as the next screenshot:

```
Input lowercase sentence:os project
OS PROJECT
```

Conclusion:

We can conclude that **multithreading** actually provides a higher speed in processing and calculation and this can be noticed when applied on complex processes. Using multithreading can result in Improved throughput and CPU Utilization

We can conclude also that **sockets** are a way for two processes to communicate *regardless* of whether it's over the network or within the same machine. Efficient socket based programming can be easily implemented for general communications.