GOVERNMENT COLLEGE OF ENGINEERING, JALGAON
(An Autonomous Institute of Government of Maharashtra)
Department of Computer Engineering

Name: Vaibhav Bharat Sontakke PRN No: 1942207

Class: T. Y. B. Tech. Batch: L4

Date of Performance: _____ Date of Completion: _____

Subject: DSL Sign. of Teacher with Date: _____

Experiment No.: 05

Aim: Design a distributed application which consist of a server and client using threads

Title: Program for Distributed Application which consist of a server and client

using threads **Software Requirements**: Ubuntu OS, Eclipse IDE 4.11 **Theory:**

PROCESS:

In a conventional centralized operating system process management deals with mechanisms and policies for sharing the processor the system among all processes .Similarly ,in a distributed operating system ,the main goal-of process management to make the best possible use of the processing resources of the entire system by sharing the among all processes. Three important concepts are used in distributed operating systems to achieve this goal:

1. Processor allocation:

Deals with the process of deciding which process should be assigned to which processor.

2. Process migration:

Deals with the movement of a process from its current location of the processor to which has been assigned. **3. Threads:**

Deals with fine-grained parallelism for better utilization of the processing capability of the system. The processor all location concept ready been described in the previous chapter on resource management. Therefore, this chapter presents a description of the process migration and threads concepts.

THREAD:

A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.

A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.

A thread is also called a lightweight process. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

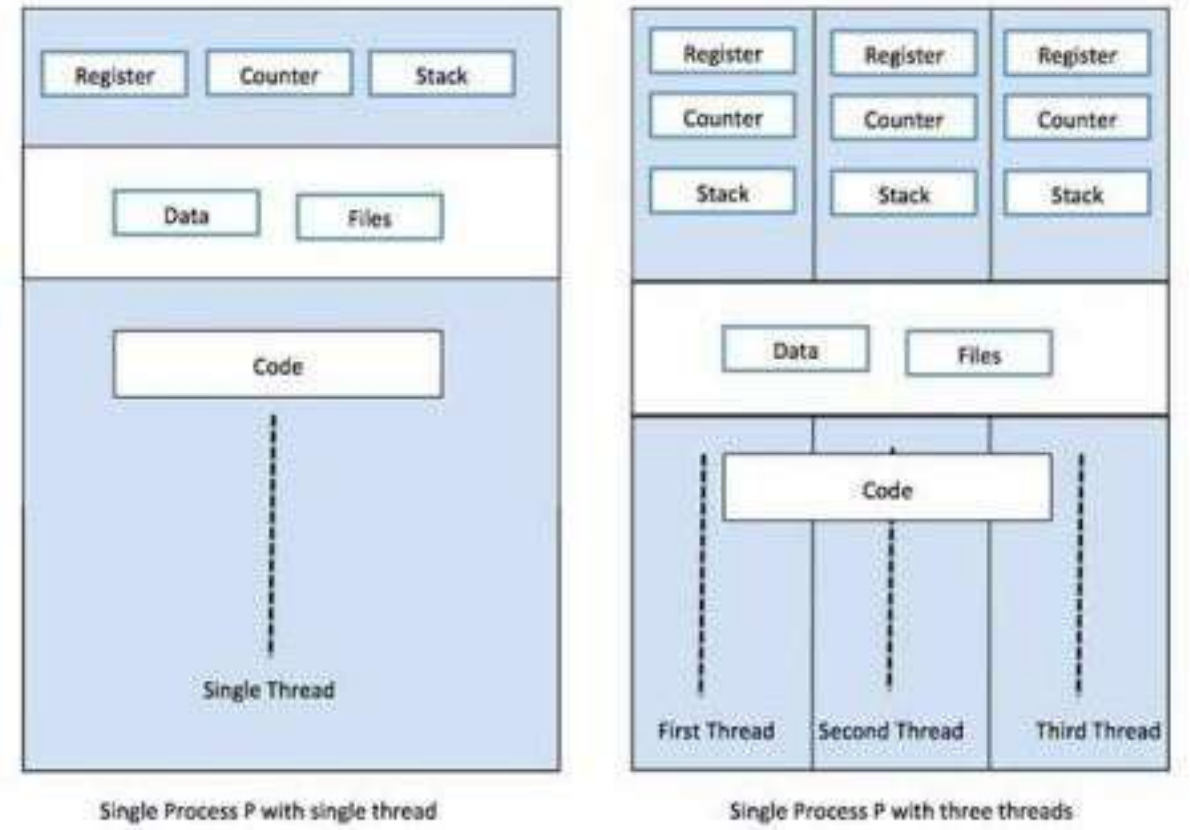


Figure 1: thread concept with Process

ADVANTAGES OF THREAD

1. Threads minimize the context switching time.
2. Use of threads provides concurrency within a process.
3. Efficient communication.
4. It is more economical to create and context switch threads.
5. Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

MULTITHREADING:

Multithreading is the ability of a program or an operating system process to manage its use by more than one user at a time and to even manage multiple requests by the same user without having to have multiple copies of the programming running in the computer. Each user request for a program or system service (and here a user can also be another program) is kept track of as a thread with a separate identity. As programs

work on behalf of the initial request for that thread and are interrupted by other requests, the status of work on behalf of that thread is kept track of until the work is completed.
The user threads must be mapped to kernel threads, by one of the following strategies:

1. Many to One Model
2. One to One Model
3. Many to Many Model

1. Many to One Model

1. As shown in figure 2, In the **many to one** model, many user-level threads are all mapped onto a single kernel thread.
2. Thread management is handled by the thread library in user space, which is efficient in nature.

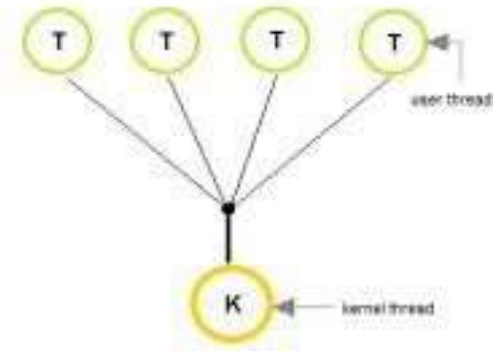
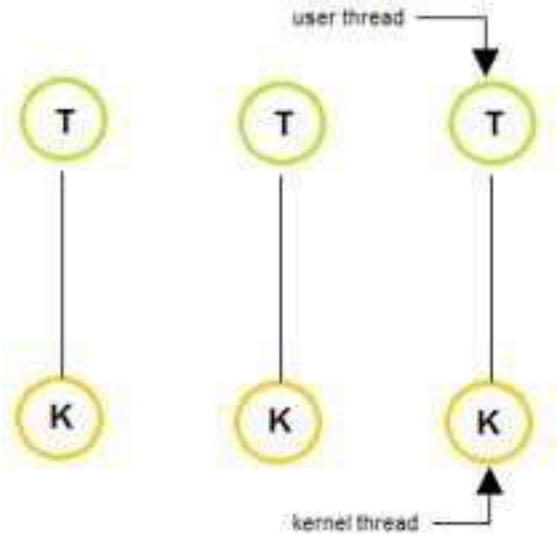


Figure 2: Many to one model

2 .One to One Model:

1. As shown in figure 3 The **one to one** model creates a separate kernel thread to handle each and every user thread.
2. Most implementations of this model place a limit on how many threads can be created.



3 .Many to Many Model

1. The many to many model multiplexes any number of user threads onto an equal or smaller
3. Linux and Windows from 95 to XP implement the one-to-one model for threads.

Difference between Process and

Thread:

Figure 3: One to one model

number of kernel threads, combining

the best features of the one-to-one and

many-to-one models shown in figure 4.

2. Users can create any number of the

threads. 3. Blocking the kernel system

calls does not block the entire process.

4. Processes can be split across multiple processors.

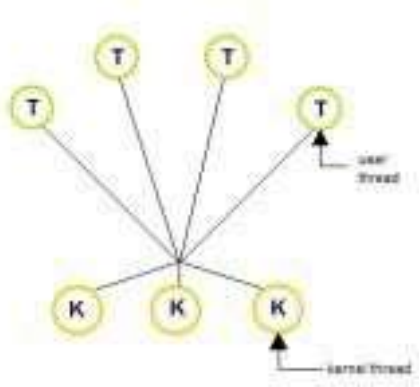


Figure 4- Many to many model

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Methods Used:

1. **public Socket(String host, int port) throws UnknownHostException, IOException.**

This method attempts to connect to the specified server at the specified port. If this constructor does not throw an exception, the connection is successful and the client is connected to the server.

2. **public InputStream getInputStream() throws IOException**

Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket. 3. **public OutputStream getOutputStream() throws IOException**

Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket.

4. **public void close() throws IOException**

Closes the socket, which makes this Socket object no longer capable of connecting again to any server. 5. **public ServerSocket(int port) throws IOException**

Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application.

6. **public Socket accept() throws IOException**

Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the setSoTimeout() method.

ALGORITHM:

Steps

Client Program:

1. Initialise input and output stream.
2. Open socket on given port.
3. Open output and input stream.
4. If everything has been initialized then we want to write some data to the socket we have opened aconnection to on the given port
5. Close input and output stream.
6. Close the socket

Server Program:

1. Calling constructor of server final and initializing in object(server) of serverfinal.
2. Calling startserver function
3. Declare a server socket and a client socket for the server
4. Declare the number of connections
5. Try to open a server socket on the given port(Note that we can't choose a port less than 1024 if we are not) 6. Whenever a connection is received, start a new thread to process the connection and wait for the next connection.
7. Close the client connection.

Conclusion:

In this Practical, We learnt about development of distributed application consisting server and client using threads.

……..

Miss. Archana Chitte

Name & Sign of Course Teacher