16/04/25

Page No:
Date:

Thread Programming -

Thread - Single flow of execution is known as thread

Multithread - A multiple flow of execution is known as multithread.
It is much similar to multitasking or multiprocessing.

Multitasking :

Task - A running state of program/ job / process.

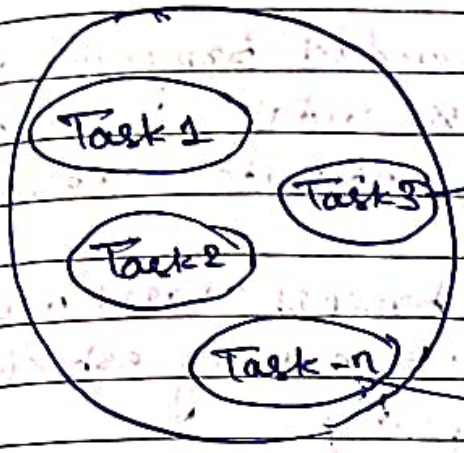
Task are of two types.
- Single tasking
- Multi-tasking

Multi-tasking:
- At a time more than one process or job can be performed.
- Multitasking means executing more than one task at a the same time.

Single tasking:
- At a time user can perform single process or job.

Multitasking can be classified into two categories
(i) Process based multitasking
(ii) Thread based multitasking

(i) Process based multitasking -

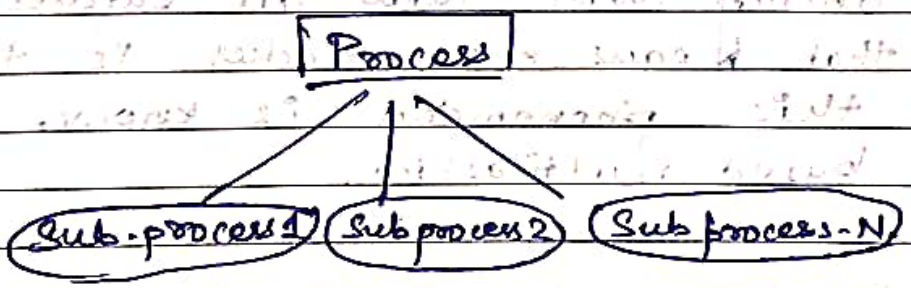


→ It occupies its own memory space for the execution of the program.

Similarly every task occupies its own memory for the program execution.

In process based multitasking, the program execution is based on memory i.e., RAM.

(ii) Thread based multitasking - A process is sub-divided into sub-processes to perform a task. All the sub-processes contains packet (small amount of information).



- Thread based multitasking increases the efficiency of the program.
- Based on scheduling algorithm i.e., Round Robin

Process Based Multitasking

Each task has independent program or process and every process contains own memory for individual memory for separate execution of the program.

The process based multitasking work on the principle of scheduling especially round robin.

Thread Based Multitasking

Each task is independent thread and execute at separate part of the program.

Ex:-

Suppose we have 100 students in coaching centre. Each batch has 10 student and 4th batch is going on by one teacher that is based on time slot. If company hires 3 more teacher and batches are distributed among them then all batches goes parallely that means every teacher is thread and this phenomenon is known as thread based multitasking.

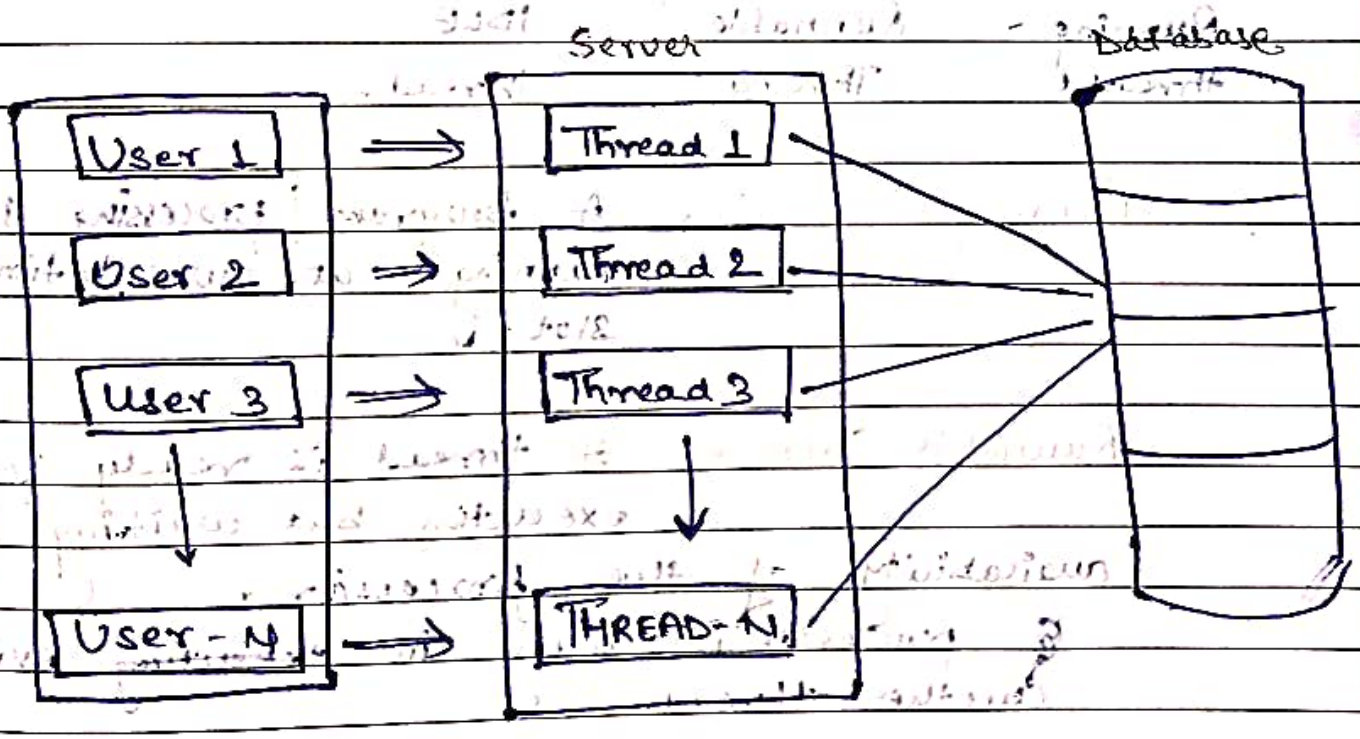
* Thread - also called as single thread
It is an operating system object that execute the instruction of the program. It follows the scheduling algorithm also Round Robin scheduling.

Page No:.....
Date:...../...../.....

Main thread – When we start to create a python program, one thread runs automatically (immediately) known as main thread of program.

Multithreading – It is a conceptual programming where a program is divided into ~~two~~ or more than one or 'n' sub programs and all that programs get executed simultaneously / concurrently. This phenomenon is called as multithreading programming.

Note :– Main thread is created by PVM (Python Virtual Machine). PVM is sometimes known as python interpreter.

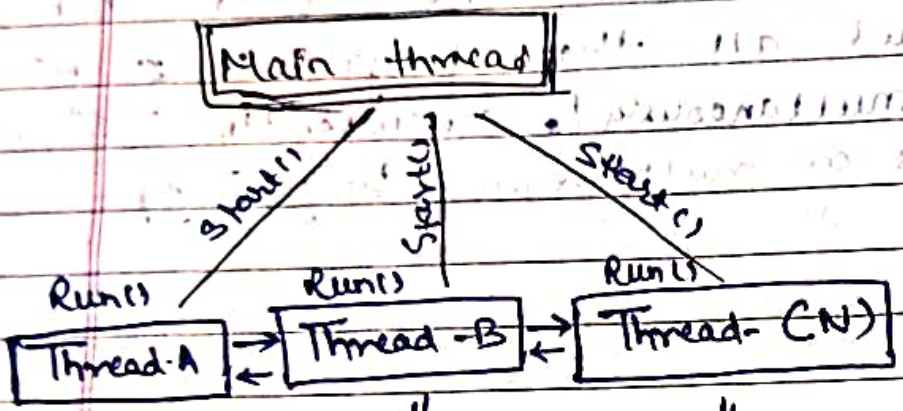


Example of multithreading:

(i) Pubg - In PUBG user can perform several task at the same time concurrently.

(ii) Image displaying

(iii) Multimedia Graphics



Running thread

Runnable Thread

IDLE Thread.

Running Thread. - A program / processing is running at current time slice / slot.

Runnable Thread:- A thread is ready for the execution but waiting for the availability of the processor.

processor is busy in ~~executing~~ execution of another thread.

Idle thread - It is neither dead nor alive. It maybe either suspended or waiting for processor.

Page No:.....
Date:...1.1....

Note :- A multithreading allows you to run multiple thread concurrently within a single process. This is known as Thread based parallelism.

Multi threading in python is very useful for multiple input/output operation. Generally a program sequentially executes the instruction from start to end whereas multithreading divides the main task into more than one sub-tasks and execute them in overlapping manner.

Creating Thread
(How to create a thread in Python)

- We can create thread by two ways
  - By thread class present in Thread Module
  - By extending thread
    (Inheritence)

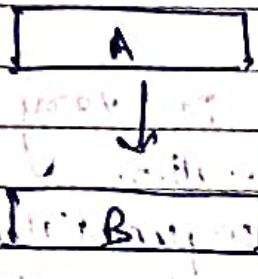
Inheritance :

Inheritance is the very important concept of OOPs that is followed in python.

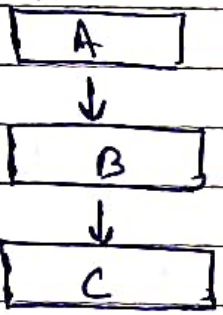
It is a mechanism that allows the subclass or child class or derived class to acquire the attribute / property / method / behaviour of parent / super class.

Types of inheritance.

(i) Single level Inheritance

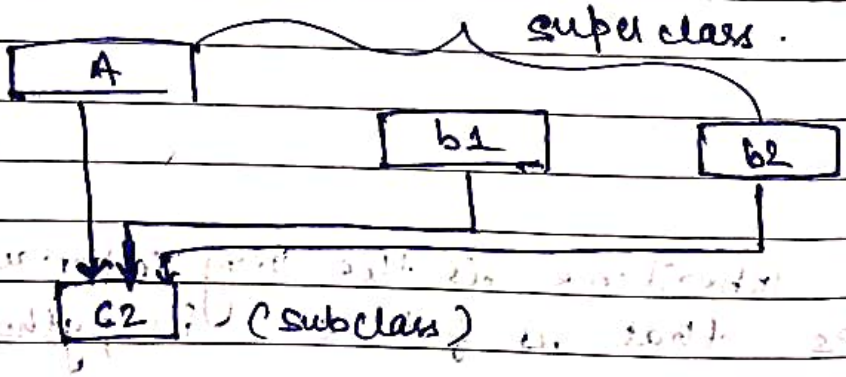


(ii) Multilevel inheritance.



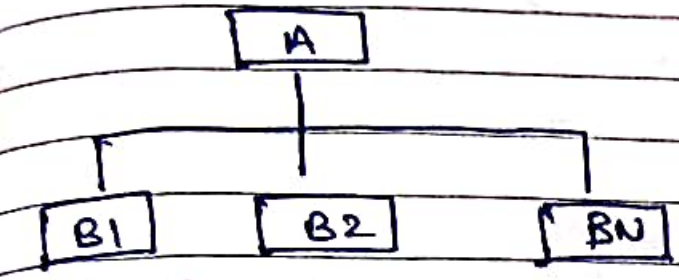
Multiple

(iii) ~~Hierarchical~~ Inheritance

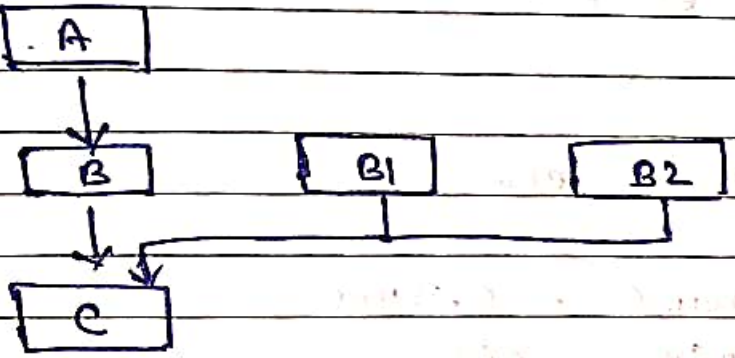


Hierarchical

(iv) ~~Hybrid~~ Inheritance.



(v) Hybrid Inheritance (Multilevel + Multiple)



Note - The inheritance allow the user for code reuse ~~that means~~ and reducing data redundancy as well as promoting the hierarchical structure of OOPs.

Inheritance

It is the process to inherit the properties and method (behaviour) from base class (attributes) or super class to derived class

17/04/25

Example of inheritance : Single Level.

```
class EMP:
    def __init__(self, ename, eid, salary, city):

        self.ename = ename
        self.eid = eid
        self.salary = salary
        self.city = city

    def emp_info(self):
        print(f" The Empname = {self.ename}
                  Identification = {self.eid}
                  Salary = {self.salary}
                  City = {self.city}

class DEPT(EMP):
    def __init__(self, ename, eid, salary,
                 city, dept_name, sub):

        ~~super = super().defi~~

        super().__init__(ename, eid, salary,
                         city)
        self.dept_name = dept_name
        self.sub = sub

    def emp_info1(self):
        super().emp_info()

        print(f" Department = {self.dept_name}
                  Subject = {self.sub} ")
```

```
class DEPT(Emp):
    def __init__(self, ename, eid, salary,
                 city, deptname, sub):

emp1 = Emp()
dept1 = DEPT("AMIT", 111, 10000, Ranchi,
             "MCA", "PYthon")

dept.info

dept1 . emp_info()
dept  . emp_info1()
```

Note:- Dept class inherit the attribute and behavior of employee emp class by the help of single level.

WAP in python using OOPs inheritance input name, roll, gender, 5 submarks. Find total marks and average and display result using single level inheritance.

19/04/25

Page No:.....
Date:.../.../....

Multithreading

→ How to create a thread class

Step 1 :- Using API package

```
from threading import *
```

(or)

```
import threading
```

Step 2 :- Now, we can create new thread with the help of thread object.

```
Objectname = threading.Thread(target, arg = ())
```
(arg = () — optional)

(or)

```
objectname = Thread(target = ())
```

Step 3 :- Using two method to execute and stop the thread

(a) Start thread

```
Object_name.start()
```

(b) Stop thread

objectname.stop(). (or) objectname.join()

WAP to print first 10 natural using thread class

```
from threading import *
class Natural:

    def compute (self):

        num = 0

        while (num <10)
            print (num)
            num = num +1

Obj1 = Natural()
thread1 = threading.Thread (target = obj1.
                            compute())

thread1.start()
thread1.join()
```

WAP to implement thread class & and find even no. sum within given range.

```
import threading
class EvenNumber:

    def display (self):
        i = 10
        j = 20
```

for loop syntax

Page No:
Date: / /

for "counter_variable" in range (start, stop, increment/decrement)

```
for num in range ( i, j+2):
    if (num % 2 == 0)
        print( num)
        (or)
        print (f" The Even number i = {self.num}")

en = EvenNumber()
thread1 = Thread (target = en.display())
        thread1.start()
        thread1.join()
```

* WAP in python to create two thread and find even and odd number from 30 to 50 without using thread class. By function

```
import threading
    (or)
from threading import *

def Even_number() :
        print ("List of even number")
        for num in range ( 30 , 51 + 2) :
```

print(num)

(or)

print(num, end = " ")

def Odd_Number():

print("List of ~~Even~~ Odd Numbers")

for num1 in range(~~2~~ 31, 50 +2):

print(num1)

(or)

print(num1, end = " ");

thread1 = threading.Thread(target = Even_numbers)
thread2 = ~~the~~ threading.Thread(target = Odd_Numbers)

thread1.start()
thread1.join()
thread2.start()
thread2.join()

Note:- Using ~~*~~ print(xyz, end = " ")

takes the cursor to the next line.

" - double space within inverted comma"

Page No:
Date: / /

Assignment

WAP using thread class and find the factorial of given number.

WAP in pytho to create one thread without the help of thread class and find the factorial of given number with the help of user defined functions

22/04/25

Page No:
Date:

How to create thread by extending thread class.

We know we can create a thread by three approach.

1. By thread class
2. By function approach
3. By extending thread class

Method - overriding

It is a very important concept of OOP or polymorphism that means ~~one name~~ it is much more similar to one name with multiple form.

In another word we can say that method overriding refers to a method in subclass with same name as method in superclass

Method overriding can be done using

i.e., one for inbuilt functions

run() → An inbuilt function/method (pre-defined method)

this method is invoked by start()

⇒ We can never change the properties and behavior of the inbuilt method. Also its name cannot be changed.

```
class EMP:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
    def getName(self):
        return self.name
    def getSalary(self):
        return self.salary

class Office(EMP):
    def __init__(self, name, salary, inc):
        super().__init__(name, salary)
        self.inc = inc
    def getSalary(self):
        return self.salary + self.inc

e1 = EMP("RAM", 70000)
print(f"The salary of employee = {self.salary}")
e1.getName()
e1.getSalary()
```

→ Method overloading is done within the class
→ Method over-riding is done in extended class.

c2. Office ("RITA", 50000)
print(f" The salary with increment = {self.salary}")
c2. getSalary()

Python provide inbuild thread class. We have to create a class by extending from the main thread in the class.

Inside the thread class python provide "run()" that means run method will be invoked inside the main thread and we have to over-ride the "main()".

The run() will override the through execution logic when start method is called.

We can say that run() must be invoked by another method i.e., start().

Note:- A run() is a heart and soul of thread programming In case of extend class module that means we can say that a small program and complex program must use the run() for over-riding the thread execution from super. class to sub class.

Page No:.....
Date:.../.../....

~~Each~~

Note: If subclass over-ride the constructor, it must ~~invoke~~ ~~star~~ make sure to invoke ~~thread~~ ~~.thread.init~~

Thread.__init__() method, before doing anything else to read.

23/04/25

Example

When we start a python program one thread running ~~thread~~ immediately is known as parent thread or main thread and user can call the property and behavior of main thread to subthread in case of extending thread class

run() → In thread programming we can implement it in case of over-riding.

```
from threading import
class Test (Thread):

    def run(self)

        ~~for i in rar~~
        for i in range (5):
            print ('AMIT')
```

```
class Test1 (Thread):

    def run (self):

        for i in range (5):
            print ('Advance Python')
```

```
obj1 . Test ()
obj2 . Test1 ()

obj1 . run()
obj2 . run ()
```

in this case the o/p will be first completed from first class then the 2nd class is executed

```
AMIT
 :   } 5 times
AMIT
Advance Python
 :   } 5 times
Advance Python
```

This made it as a single threaded program.

- To make it a multithreaded program, we must first invoke the start() method.

```
obj1 . Test ()
obj2 . Test1 ()

Obj1 . start ()
Obj2 . start ()
```

```
obj1 . run ()
obj2 . run ()
```

in this case a random output from both class is generated.

Hence, multithreading concept is used.

Page No:.....
Date:...1.1....

Creating a thread by extending thread using a Constructor.

How constructor will be over-ridden in thread.

```
from threading import *
class Test A(Thread):
    def __init__(self, name):
        Thread.__init__(self)
        self.name = name
    def run(self):
        for i in range(5):
            print(self.name)

class Test1(Thread):
    def __init__(self, name):
        Thread.__init__(self)
        self.name& = name
    def run(self):
        for i in range(5):
            print(self.name)

obj = Test('MCA')
obj1 = Test1('MSc IT')

obj.start()
obj1.start()
```

Assignment

WAP in python to implement the concept of over-ride constructor from main thread constructor in case of ~~inheritance~~ employee payroll system with the help of extend class module.