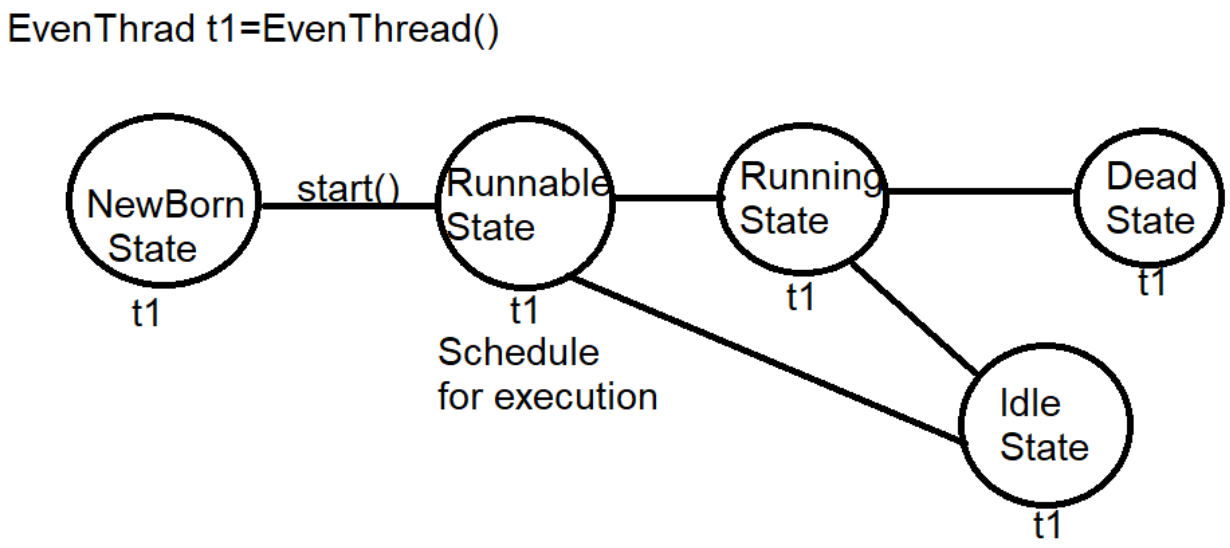
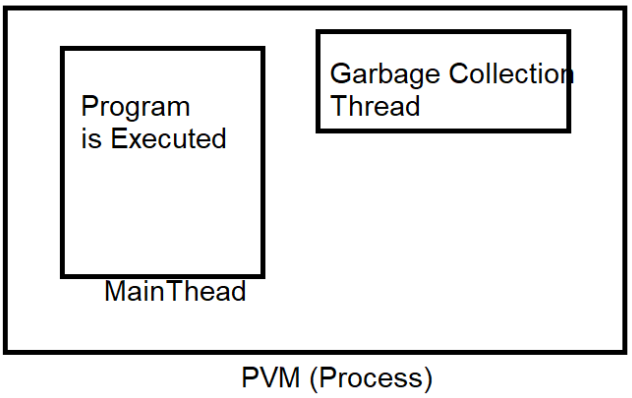
Life Cycle of a thread
Thread execution is done by PVM (Python Virtual Machine). There are different states of thread.

1. New born state
2. Runnable state
3. Running state
4. Idle state
5. Dead state



Every python program is executed within PVM process as a thread. Default thread created by PVM for executing python program is MainThread.



current_thread()

It is a predefined function of threading module; which return reference of current thread or thread which is currently in execution.

```
import threading
t1=threading.current_thread()
print(t1)
print(t1.name)
```

Output:

```
<_MainThread(MainThread, started 8936)>
MainThread
```

Every thread object created with default property called name.
This property can be used to read thread name or change thread name.

Example:

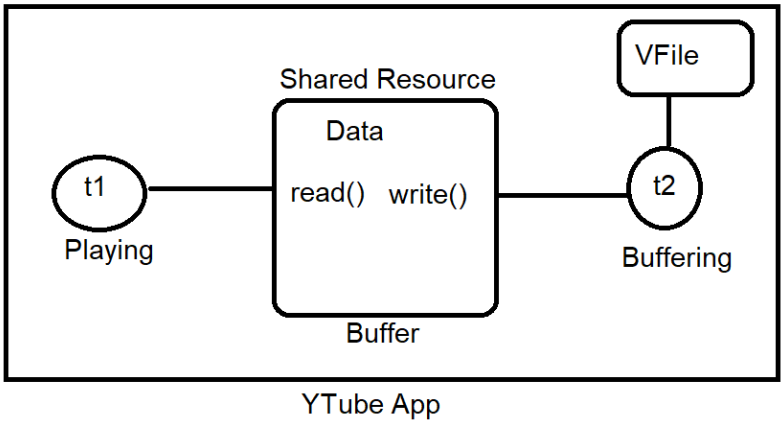
```
import threading

class AlphaThread(threading.Thread):
    def __init__(self):
        super().__init__()
    def run(self):
        name=self.name
        for n in range(65,91):
            print(f'{name}-->{chr(n)}')



def main():
    t1=AlphaThread()
    t1.name="naresh"
    t2=AlphaThread()
    t2.name="suresh"
    t1.start()
    t2.start()

main()
```

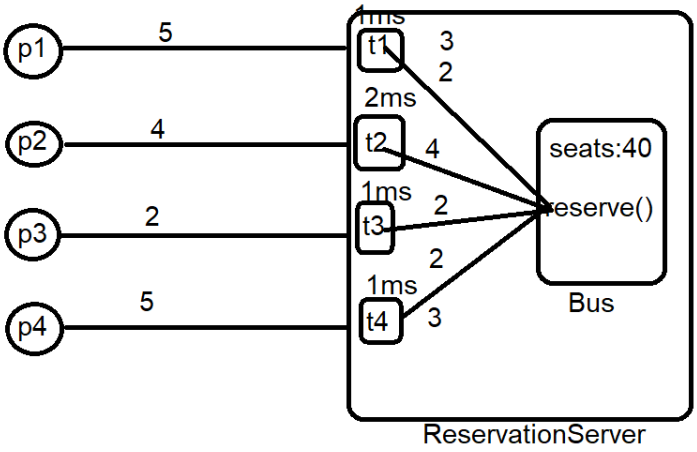
Output:

```
E:\student drive\python7amjuly>python ttest6.py
naresh-->A
naresh-->B
naresh-->C
naresh-->D
suresh-->A
naresh-->E
suresh-->B
naresh-->F
suresh-->C
naresh-->G
suresh-->D
naresh-->H
suresh-->E
naresh-->I
suresh-->F
naresh-->J
suresh-->G
naresh-->K
suresh-->H
naresh-->L
suresh-->I
```

Thread Synchronization (Locking)

Synchronization is a process of acquiring lock of the object by thread. Synchronization allows creating thread safe application or code.





Lock object
threading.**Lock**
The class implementing primitive lock objects. Once a thread has acquired a lock, subsequent attempts to acquire it block, until it is released; any thread may release it.

acquire(blocking=True, timeout=- 1)
Acquire a lock, blocking or non-blocking.

release()
Release a lock. This can be called from any thread, not only the thread which has acquired the lock.

Example:

```
import threading

class Bus:
    def __init__(self):
        self.__seats=50
        self.__l=threading.Lock()
    def reserve(self,name,s):
        self.__l.acquire()
        for i in range(s):
            self.__seats=self.__seats-1
        print(f'{name} your seats are reserved and available seats
{self.__seats}')
        self.__l.release()
```

```
class ReserveThread(threading.Thread):
    def __init__(self,b,n,s):
        super().__init__()
        self.__b=b
        self.__name=n
        self.__seats=s
    def run(self):
        self.__b.reserve(self.__name,self.__seats)


bus1=Bus()
t1=ReserveThread(bus1,"naresh",5)
t2=ReserveThread(bus1,"suresh",10)
t3=ReserveThread(bus1,"kishore",5)
t1.start()
t2.start()
t3.start()
```

Output:

```
E:\student drive\python7amjuly>python ttest7.py
naresh your seats are reserved and available seats 45
suresh your seats are reserved and available seats 35
kishore your seats are reserved and available seats 30

E:\student drive\python7amjuly>
```

Logging Module

It is a default module which comes with python software.
This module defines functions and classes which implement a flexible event logging system for applications and libraries.

Types Messages

1. Information message
2. Debugging message
3. Warning message
4. Error message
5. Critical message

All these messages are printed or displayed using print function. It is complex to identify the type of message. Print function cannot manage by programmer. This can be avoided by display messages logging module. Logging module provides different methods of display messages (File, Database, Console, Network,…)

1. log()
2. error()
3. debug()
4. warning()
5. critical()