Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

School of Computer Engineering and Technology

Lab Assignment-10

Write a python program to implement multithreading scenarios.

- Create two threads to display cube and square of 5 numbers from list.
- Threads can simultaneously execute a Cube and square functions from program to access the shared list of 5 numbers.

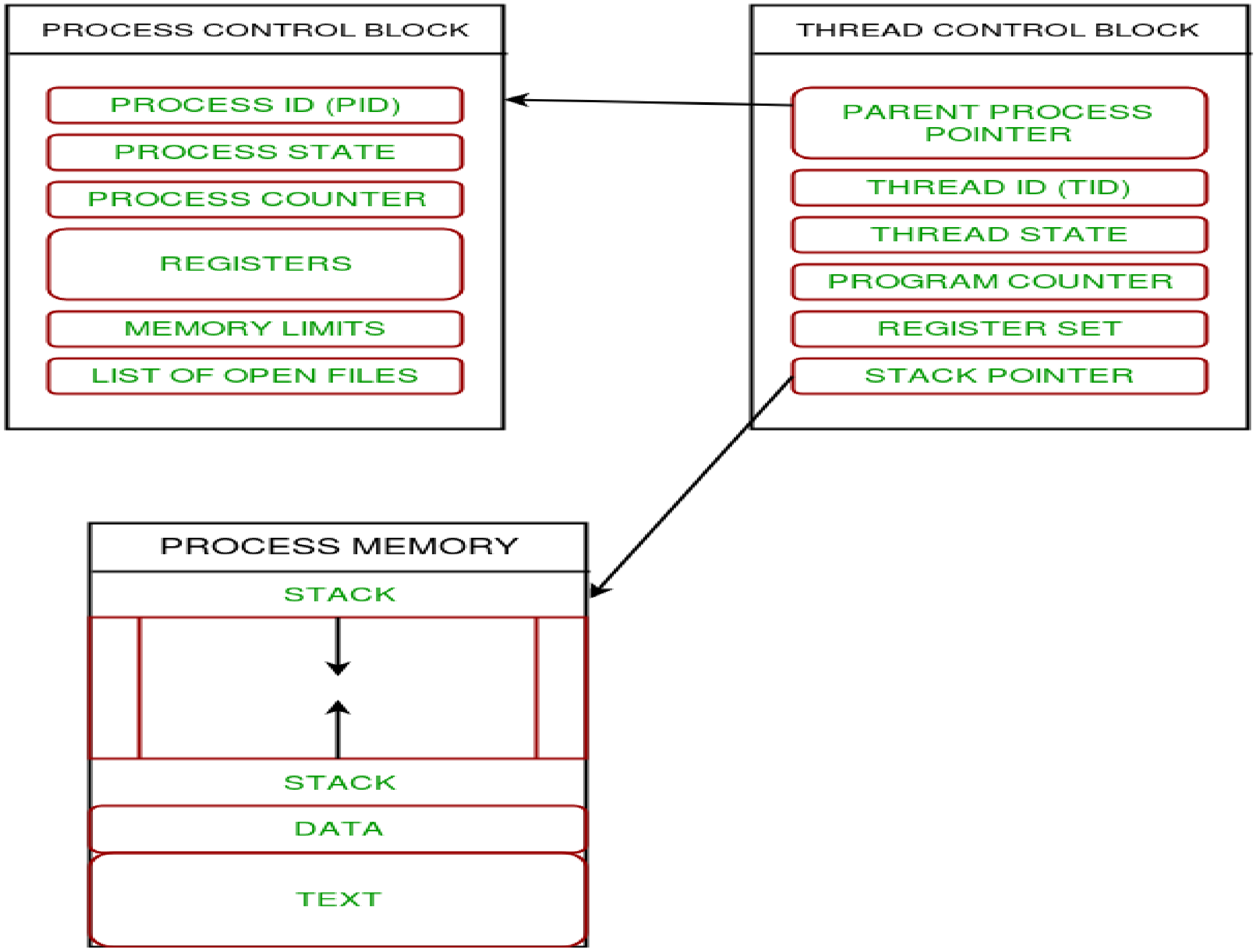
What is Thread?

- In computing, a **process** is an instance of a computer program that is being executed. Any process has 3 basic components:
  - An executable program.
  - The associated data needed by the program (variables, work space, buffers, etc.)
  - The execution context of the program (State of process)
- A **thread** is an entity within a process that can be scheduled for execution.
- It is the smallest unit of processing that can be performed in an OS (Operating System).
- A **thread** is a sequence of such instructions within a program that can be executed independently of other code.

Thread Control Block (TCB)

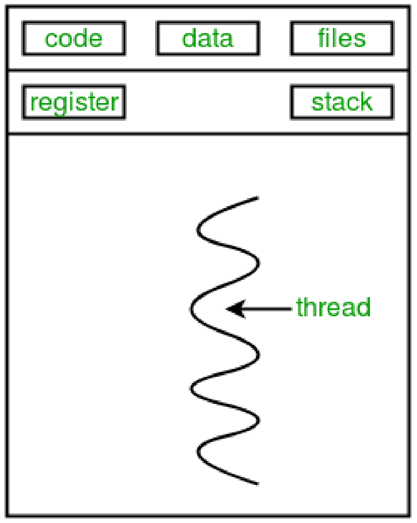
- **Thread Identifier:** Unique id (TID) is assigned to every new thread
- **Stack pointer:** Points to thread's stack in the process. Stack contains the local variables under thread's scope.
- **Program counter:** a register which stores the address of the instruction currently being executed by thread.
- **Thread state:** can be running, ready, waiting, start or done.
- **Thread's register set:** registers assigned to thread for computations.
- **Parent process Pointer:** A pointer to the Process control block (PCB) of the process that the thread lives on.

Relationship between the process and its thread:

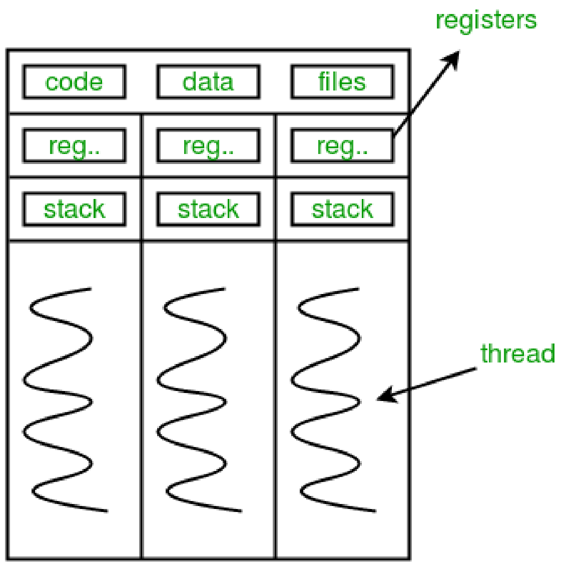


Multi-threading

- Multiple threads can exist within one process where:
  - Each thread contains its own **register set** and **local variables (stored in stack)**.
  - All threads of a process share **global variables (stored in heap)** and the **program code**.



single-threaded process



multithreaded process

Benefits of Multithreading

- It ensures effective utilization of computer system resources.
- Multithreaded applications are more responsive.
- It shares resources and its state with sub-threads (child) which makes it more economical.
- It makes the multiprocessor architecture more effective due to similarity.
- It saves time by executing multiple threads at the same time.
- The system does not require too much memory to store multiple threads.

Multi-threading in Python

- In Python, the **threading** module provides a very simple and intuitive API for spawning multiple threads in a program.
- To import the threading module
  - **import threading**
- To create a new thread, we create an object of threading.Thread() class. use. The syntax of Thread() class is:

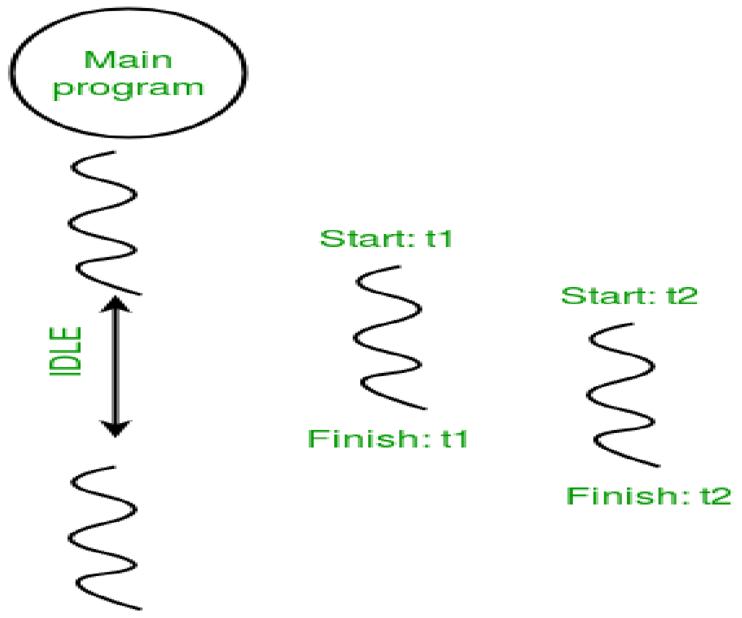
```
t1=threading.Thread(group=None, target=None, name=None,
args=(), kwargs={}, *, daemon=None)
```

  - leave **group** as None.
  - **target** is the callable object to be invoked by the run() method of Thread.
  - **name** is the Thread name that you can provide and refer to later in the program.
  - **args** is the argument tuple for the target invocation.
  - **kwargs** is a dictionary of keyword arguments for the target invocation.
  - **daemon** is not None, will be set to be daemonic.

Start a Thread

- Once you have created a thread using Thread() class, you can start it using Thead.start() method.

  - **t1 = threading.Thread()**
  - **t1.start()**

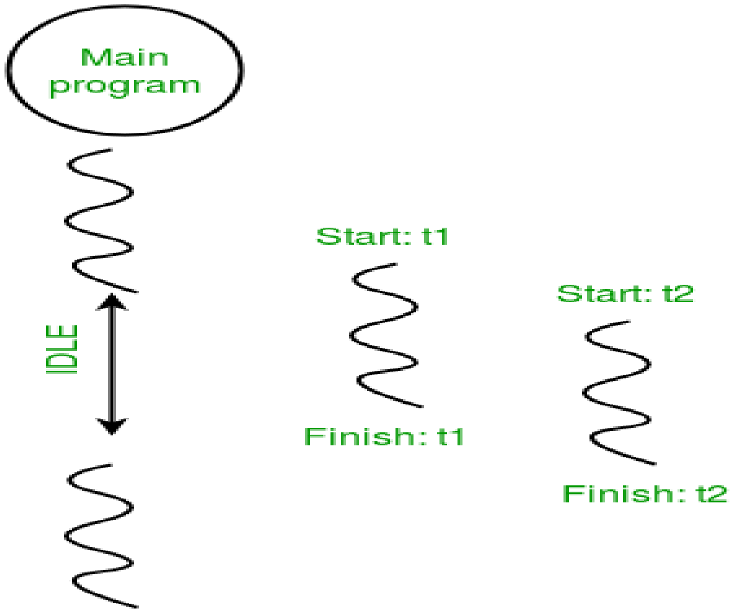


Wait until the thread is finished

- Program Counter of the main process to wait until a specific thread is finished, In order to stop execution of current program until a thread is complete join() method on the Thread object is used

  - **t1.join()**

- The execution in the main program waits here at this statement until t1 completes its target execution.



Example – Python Mutlithreading with Two Threads

```
import threading

def print_one():
        for i in range(10):
                print(1)

def print_two():
        for i in range(10):
                print(2)

# create threads
t1 = threading.Thread(target=print_one)
t2 = threading.Thread(target=print_two)

# start thread 1
t1.start()
# start thread 2
t2.start()

# wait until thread 1 is completely executed
t1.join()
# wait until thread 2 is completely executed
t2.join()
# both threads completely executed

print("Done!")
```

OUTPUT

```
1
1
1
2
2
2
1
1
2
1
2
2
2
2
2
2
1
1
1
1
Done!
```

Thank you