SET-224 /CET-225
Operating Systems

LAB # 11

LAB Title

Inter process communication (IPC) using Shared Memory.

Assessment of CLO: 04, PLO: 03

Student Name:			
Roll No.			
Semester		Session	

S. No.	Perf. Level / Criteria	Excellent (2.5)	Good (2)	Satisfactory (1.5)	Needs Improvement (0 ~ 1)	Marks Obtained
1	Project Execution & Implementation	Fully functional, optimized, and well-structured.	Minor errors, mostly functional.	Some errors, requires guidance.	Major errors, non-functional, or not Performed.	
2	Results & Debugging Or Troubleshooting	Accurate results with effective debugging Or Troubleshooting.	Mostly correct, some debugging Or Troubleshooting needed.	Partial results, minimal debugging Or Troubleshooting.	Incorrect results, no debugging Or Troubleshooting, or not attempted.	
3	Problem-Solving & Adaptability (VIVA)	Creative approach, efficiently solves challenges.	Adapts well, minor struggles.	Some adaptability, needs guidance.	Lacks innovation or no innovation, unable to solve problems.	
4	Report Quality & Documentation	Clear, structured, with detailed visuals.	Mostly clear, minor gaps.	Some clarity issues, missing details.	Poorly structured, lacks clarity, or not submitted.	
					Total Marks Obtained Out of 10	

Experiment evaluated by

Instructor's Name	Engr.Bushra Aziz		
Date		Signature	

Objective: Implementation of Inter-process communication using shared memory by using multiprocessing module in python.

Theory:

Inter-process communication (IPC)

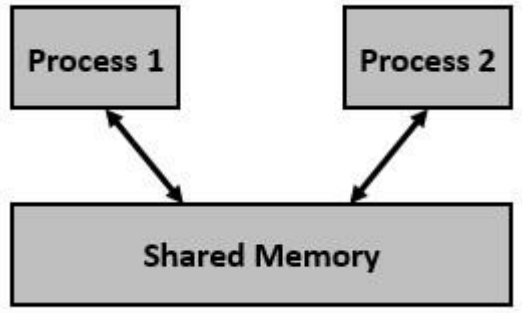
Inter process communication (IPC) is a mechanism which allows processes to communicate with each other and synchronize their actions.

There are several different ways to implement IPC, for example pipe, shared memory, message passing. IPC is set of programming interfaces, used by programs to communicate between series of processes.

Shared memory

Shared memory is a memory shared between two or more processes.

Shared memory is the fastest inter-process communication mechanism. The operating system maps a memory segment in the address space of several processes, so that several processes can read and write in that memory segment without calling operating system functions.



Shared memory is memory that may be simultaneously accessed by multiple programs with an intent to provide communication among them or avoid redundant copies.

Implementation of Shared memory using Python

It is possible to create shared objects using shared memory which can be inherited by child processes. Data can be stored in a shared memory map using Value or Array.

multiprocessing.Value(typecode_or_type, *args, lock=True)

Return a **ctypes object allocated from shared memory. By default, the return value is actually a synchronized wrapper for the object. The object itself can be accessed via the *value* attribute of a Value. typecode_or_type determines the type of the returned object: it is either a ctypes type or a one character typecode of the kind used by the array module. *args is passed on to the constructor for the type.

If lock is True (the default) then a new recursive lock object is created to synchronize access to the value. If lock is a Lock or RLock object then that will be used to synchronize access to the value. If lock is False then access to the returned object will not be automatically protected by a lock, so it will not necessarily be "process-safe".

Return a **ctypes array allocated from shared memory. By default, the return value is actually a synchronized wrapper for the array.

If *size_or_initializer* is an integer, then it determines the length of the array, and the array will be initially zeroed. Otherwise, *size_or_initializer* is a sequence which is used to initialize the array and whose length determines the length of the array.

** ctypes is a foreign function library for Python. It provides C compatible data types, and allows calling functions in DLLs or shared libraries. It can be used to wrap these libraries in pure Python.

Example#1:

Consider following example code to store data in shared memory using **multiprocessing.Value:**

```
import multiprocessing
```

```
def sharedM(v):
  v.value= 2.678

v = multiprocessing.Value('d',0.0)
p1 = multiprocessing.Process(target= sharedM,args=(v,))

p1.start ()
p1.join ()
print (v.value)
```

Example#2:
Consider following example code to store data in shared memory using **multiprocessing.Array:**

```
import multiprocessing
def sharedM (a):
   for i in range(10):
     a[i] = a[i]*a[i]

arr = multiprocessing.Array('i',range(10))
p= multiprocessing.Process (target= sharedM,args=(arr,))
p.start()
p.join()
print(arr[:])
```

The `d' and 'I' arguments used when creating **v** and **arr** are type codes of the kind used by the array module: 'd' indicates a double precision float and 'i' indicates a signed integer.

Exercise:

1. Start five processes using multiprocessing. Process objects, each process will update shared memory Value object using their own target function (callable object to be invoked by the run() method). After execution of all child processes, parent process should display the value of the object.

2. Generate 10 random numbers between 0 and 10, and calculate square of each number such that process#1 calculates square of first five numbers and process#2 calculates square of remaining five numbers, Store the square results in an array (shared memory region) using multiprocessing module