1 DSS_py

DSS_py is a Distributed System Simulator. This library allows one to simulate a distributed system in a single system. One can create a number of machine instances, assign them tasks in the form of functions and make them communicate between each other through messages.

The simulation of a Distributed environment needs an abstract layer, over which one can implement the logic and algorithm of the desired method. To achieve this abstraction, this library was developed. This library can be used to create instances and work with them in a way in which one would work on an independent system. The communication is connectionless, which facilitates the transfer of small messages efficiently.

The code is available in github. Link : `https://github.com/harisphnx/Distributed_Sytem_Simulator-DSS_py.git`

1.1 How the library works

1.1.1 Creating a New Instance

```
machine_1 = machine()
```

1.1.2 Assigning a Task in the Form of a Function

```
machine_1.execute_func("func_name", 1, 2)
```

Where *func_name* is the name of the function, followed by the arguments that the function accepts.

Every function that is to be assigned as a task to the machine instance has to have their first parameter as the identity variable. One can name it whatever he/she wants. This identity variable has to be used while calling other functions such as *send()*, *recv()* and *get_machine_id()*. The function can have its formal variables (as sent through *execute_func()*) just after the identity variable.

For example, lets say one wants to create a machine instance **m1** and assign it a task of printing first 10 numbers.

Keep the function definition in functions.py

```
def foo(identity_variable, x):
    for i in range(x):
        print i
# just a note, this identity_variable has to be used to call send(),
# recv() and other further defined functions
```

Creating machine instance and assign task through function

```
>> from dss import *
>> m1 = machine()
>> m1.execute_func("foo", 10)
```

NOTE: The machine instance would run the function in the background. One can create another machine instance just after the above lines and assign some function.

1.1.3 Sending a Message to another Machine Instance

```
identity_variable.send("machine_1", "hello")
```

Where *"machine_1"* is the machine ID of the destination machine and *"hello"* is the message.

send() returns 1 on success.

send() returns -1 when unsuccessful. This error will occur when you pass machine id that is not present.

1.1.4 Receiving a Message from other Machine Instance

```
identity_variable.recv()
```

recv() is blocking, and will return only when some machine has sent a message that is buffered or some machine sends at that time.

recv() returns *message, sender_machine_id* on success.

recv() returns *-1, -1* when unsuccessful.

1.1.5 Get the current machine ID

```
identity_variable.get_machine_id()
```