**Report**

**Operating System**

**Topic :**

* **Multiprocessing and Multithreading examples in python.**

**MEMBERS :**

Ersum Moghis(60616)

Omer Shahab(60154)

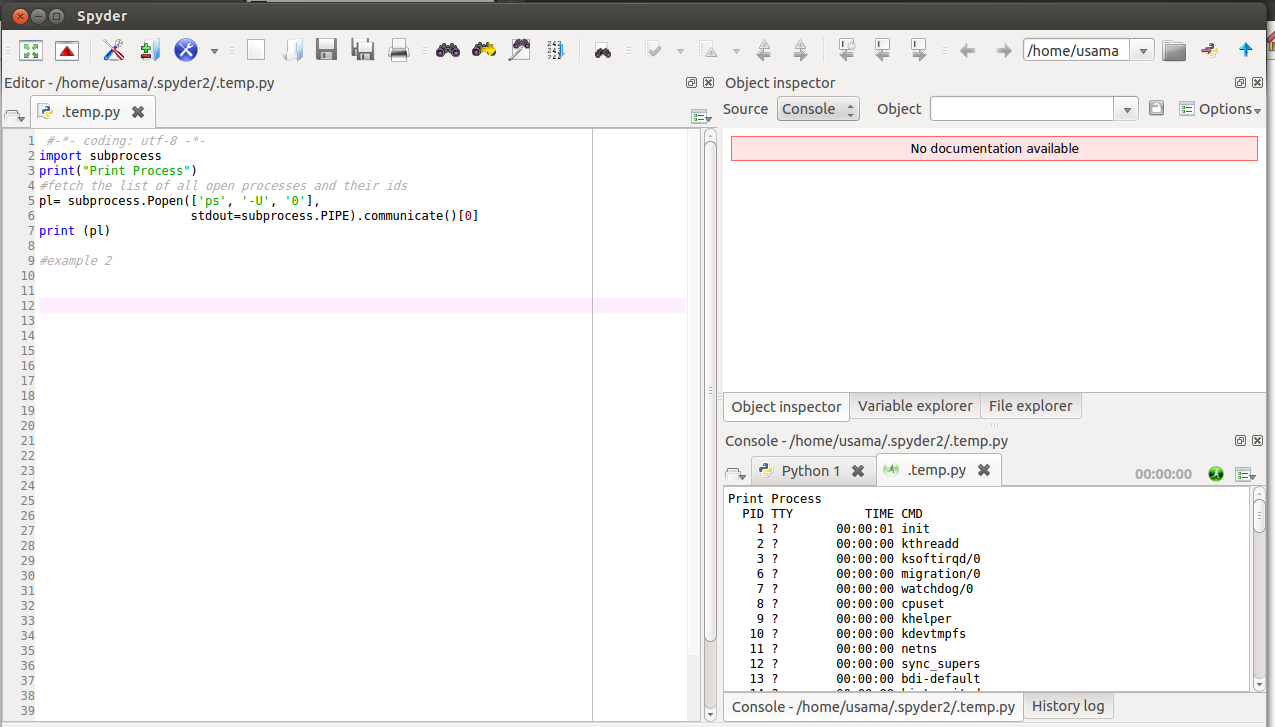
Saad Siddiqui (60181)

Usama-Ul-Haq(60608)

Raheel Saleem (60682)

**Multiprocessing:-**

**Example # 1**



**CODE :**

#-\*- coding: utf-8 -\*-

import subprocess

**#fetch the list of all open processes and their ids**

pl= subprocess.Popen(['ps', '-U', '0'],

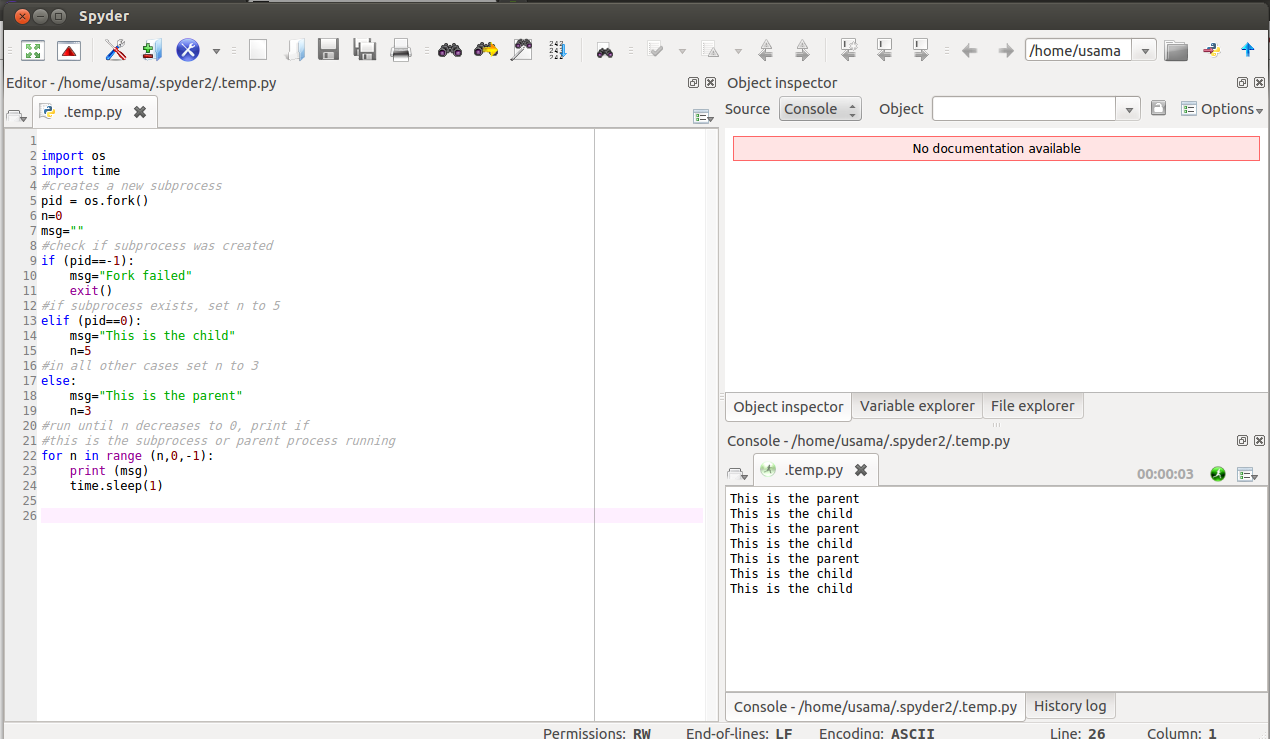
stdout=subprocess.PIPE).communicate()[0]

print (pl)

**Description :**

In this code it is fetching all open processes in the machine, along with there ids. As you can see theoutput in the above screenshot.

**Example # 2 :**



**Code :**

import os

import time

**#creates a new subprocess**

pid = os.fork()

n=0

msg=""

**#check if subprocess was created**

if (pid==-1):

msg="Fork failed"

exit()

**#if subprocess exists, set n to 5**

elif (pid==0):

msg="This is the child"

n=5

**#in all other cases set n to 3**

else:

msg="This is the parent"

n=3

**#run until n decreases to 0, print if**

**#this is the subprocess or parent process running**

for n in range (n,0,-1):

print (msg)

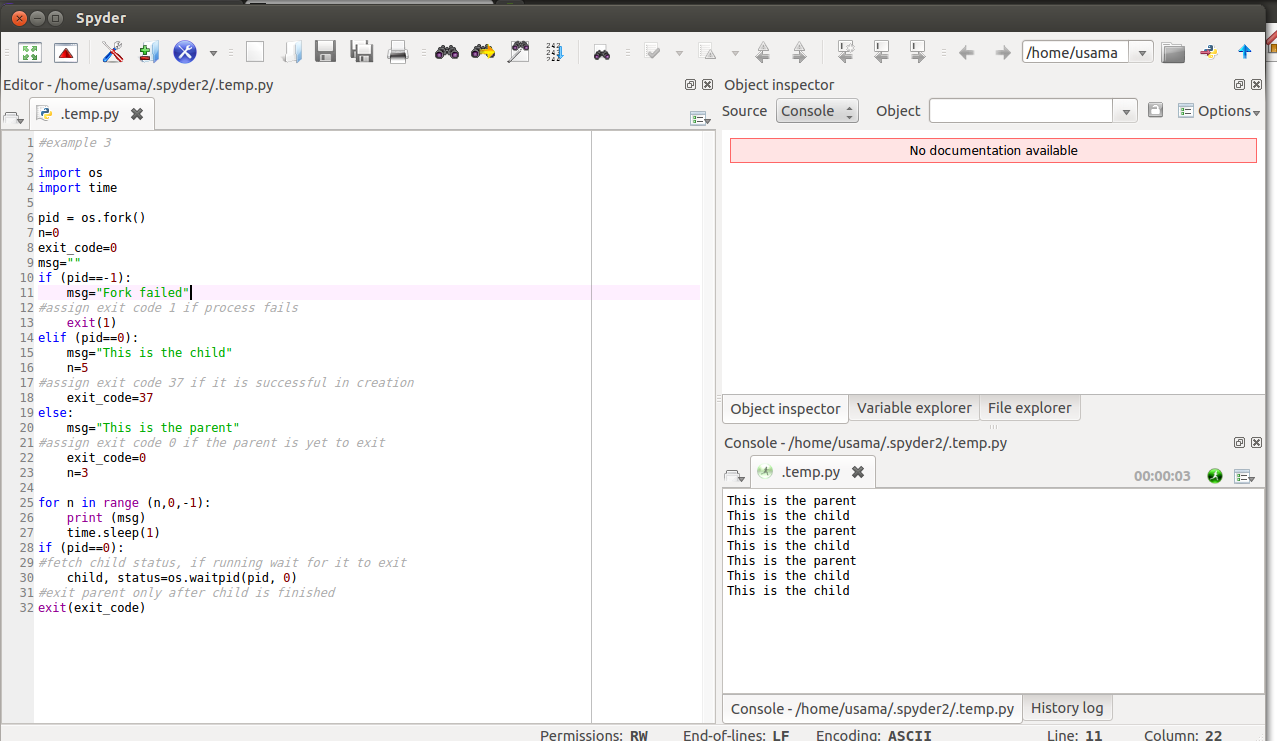
time.sleep(1)

**Description :**

In this example we are creating subprocesses. There are two parents which have subprocesses also known as child process. It also check if the process Is already create so it won’t create it again and will return error.

**Example # 3 :**

**Screenshot :**

****

**Code:**

import os

import time

pid = os.fork()

n=0

exit\_code=0

msg=""

if (pid==-1):

msg="Fork failed"

**#assign exit code 1 if process fails**

exit(1)

elif (pid==0):

msg="This is the child"

n=5

**#assign exit code 37 if it is successful in creation**

exit\_code=37

else:

msg="This is the parent"

**#assign exit code 0 if the parent is yet to exit**

exit\_code=0

n=3

for n in range (n,0,-1):

print (msg)

time.sleep(1)

if (pid==0):

**#fetch child status, if running wait for it to exit**

child, status=os.waitpid(pid, 0)

**#exit parent only after child is finished**

exit(exit\_code)

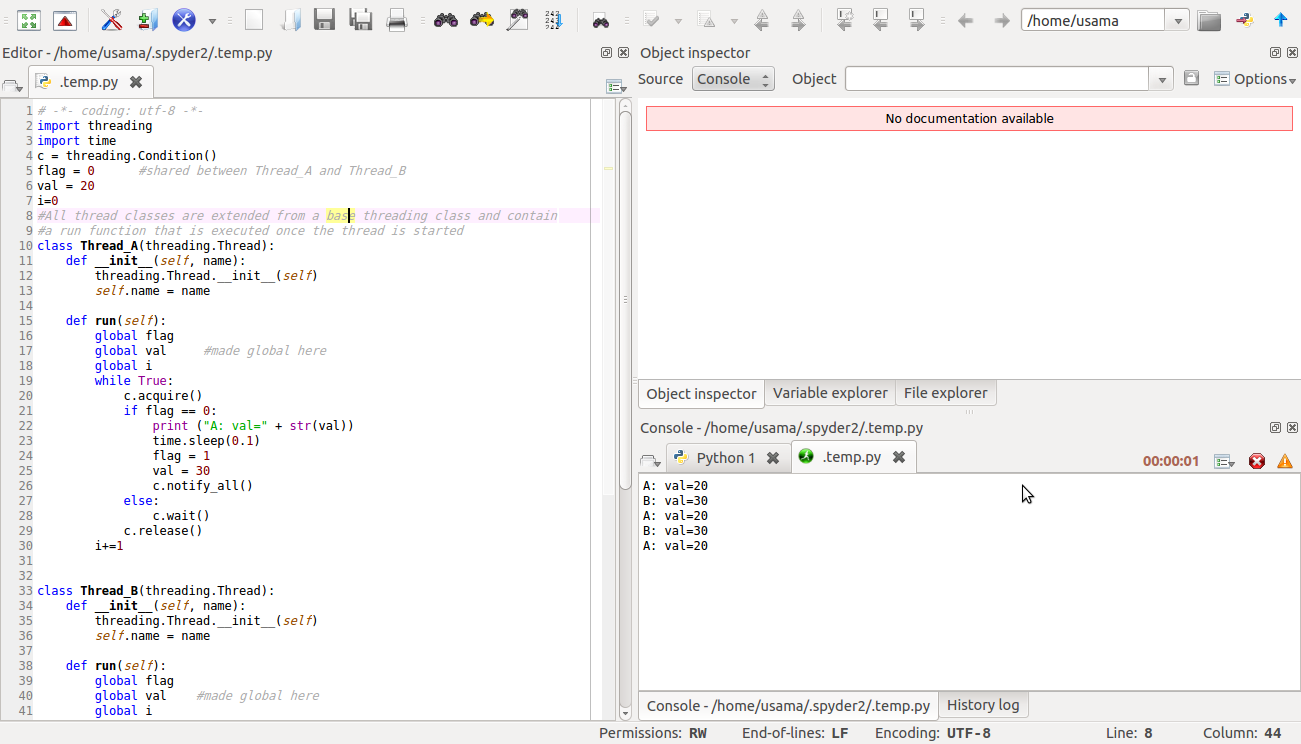
**Description:**

The parent process, which got a nonzero return from the fork call, uses the wait system call to suspend its own execution until status information becomes available for a child process. This happens when the child calls exit; we gave it an exit code of 37. The parent then continues, determines that the child terminated normally by testing the return value of the wait call, and extracts the exit code from the status information.

**Mulithreading:**

**Example # 1 :**

**Screenshot:**

****

**Code:**

# -\*- coding: utf-8 -\*-

import threading

import time

c = threading.Condition()

flag = 0 #shared between Thread\_A and Thread\_B

val = 20

i=0

**#All thread classes are extended from a base threading class and contain**

**#a run function that is executed once the thread is started**

class Thread\_A(threading.Thread):

def \_\_init\_\_(self, name):

threading.Thread.\_\_init\_\_(self)

self.name = name

def run(self):

global flag

global val **#made global here**

global i

while True:

c.acquire()

if flag == 0:

print ("A: val=" + str(val))

time.sleep(0.1)

flag = 1

val = 30

c.notify\_all()

else:

c.wait()

c.release()

i+=1

class Thread\_B(threading.Thread):

def \_\_init\_\_(self, name):

threading.Thread.\_\_init\_\_(self)

self.name = name

def run(self):

global flag

global val **#made global here**

global i

while True:

c.acquire()

if flag == 1:

print ("B: val=" + str(val))

time.sleep(0.5)

flag = 0

val = 20

c.notify\_all()

else:

c.wait()

c.release()

i+=1

**#creating objects of thread class**

a = Thread\_A("myThread\_name\_A")

b = Thread\_B("myThread\_name\_B")

**#start threads**

b.start()

a.start()

**#run threads simaltaneously**

a.join()

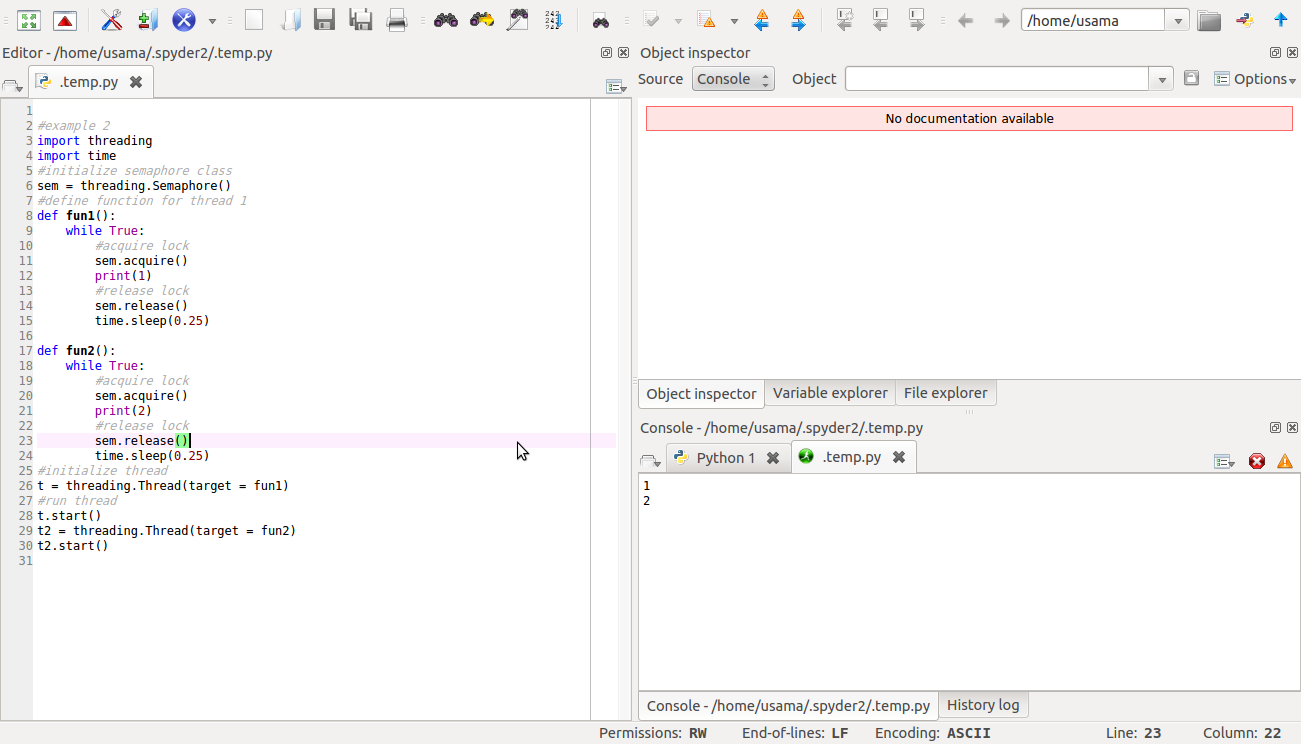
b.join()

**Description**

All thread classes are extended from a base threading class and contain. A run function that is executed once the thread is started. There is flag variable which shares thread between Thread A and Thread B. Other explanation are mention in code with comments.

**Example # 2 :**

**Screenshot:**

****

**Code:**

import threading

import time

**#initialize semaphore class**

sem = threading.Semaphore()

**#define function for thread 1**

def fun1():

while True:

**#acquire lock**

sem.acquire()

print(1)

**#release lock**

sem.release()

time.sleep(0.25)

def fun2():

while True:

**#acquire lock**

sem.acquire()

print(2)

**#release lock**

sem.release()

time.sleep(0.25)

**#initialize thread**

t = threading.Thread(target = fun1)

**#run thread**

t.start()

t2 = threading.Thread(target = fun2)

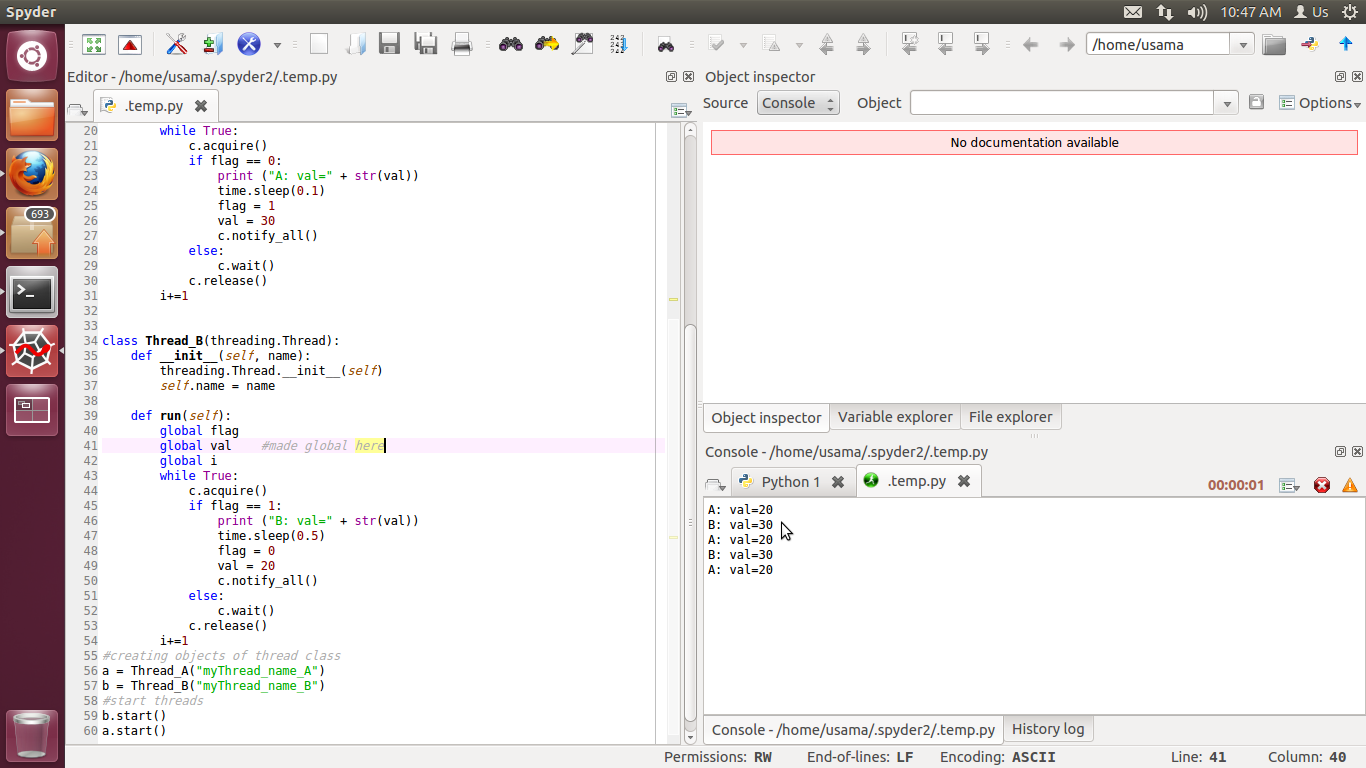
t2.start()

**Description**

In this example we are using multithreading using semaphores.

**Example # 3 :**

**Screenshot:**

****

**Code:**

import threading

import time

c = threading.Condition()

flag = 0 **#shared between Thread\_A and Thread\_B**

val = 20

i=0

#All thread classes are extended from a base threading class and contain

#a run function that is executed once the thread is started

class Thread\_A(threading.Thread):

def \_\_init\_\_(self, name):

threading.Thread.\_\_init\_\_(self)

self.name = name

def run(self):

global flag

global val **#made global here**

global i

while True:

c.acquire()

if flag == 0:

print ("A: val=" + str(val))

time.sleep(0.1)

flag = 1

val = 30

c.notify\_all()

else:

c.wait()

c.release()

i+=1

class Thread\_B(threading.Thread):

def \_\_init\_\_(self, name):

threading.Thread.\_\_init\_\_(self)

self.name = name

def run(self):

global flag

global val **#made global here**

global i

while True:

c.acquire()

if flag == 1:

print ("B: val=" + str(val))

time.sleep(0.5)

flag = 0

val = 20

c.notify\_all()

else:

c.wait()

c.release()

i+=1

**#creating objects of thread class**

a = Thread\_A("myThread\_name\_A")

b = Thread\_B("myThread\_name\_B")

**#start threads**

b.start()

a.start()

**Description**

All thread classes are extended from a base threading class and contain. A run function that is executed once the thread is started. There is flag variable which shares thread between Thread A and Thread B. Other explanation are mention in code with comments.