Experiment 5

Aim : implement peterson's solution for 2 process P0 and P1 in process synchronization

CODE:

```
# Initialize global variables
flag = [False, False]
Turn = 0
b = 0
a = 0
B = 0
def P0():
    global a, b  # Specify that you want to modify global 'a' and 'b' within this function
    while True:
        flag[0] = True
        Turn = 1
        while Turn == 1 and flag[1] == True:
            pass
        # Critical Section
        a = a + 1
        print("Context Switch karna hai?" + "Press 1 for Yes" + "Press 0 for No")
        count = int(input("Enter Your choice : "))
        if count == 1:
            P1()
        else:
            pass
        flag[0] = False
        b = b + 1
        print("b is " + str(b))
        P1()
        return b

def P1():
    global a, B  # Specify that you want to modify global 'a' and 'B' within this function
    while True:
        flag[1] = True
        Turn = 0
        while Turn == 0 and flag[0] == True:
            pass
```

```
        # Critical Section
        a = a + 1
        flag[1] = False
        B = B + 1
        print("B is " + str(B))
        return B

t = 1
while t:
    P0()
    t = t – 1
```

Output :-

```
D:\Programming\ML\Python_projects\project_code\venv\Scripts\python.exe
Context Switch karna hai?Press 1 for YesPress 0 for No
Enter Your choice : 0
b is 1
B is 1
```

Experiment 6

Aim : a) implement binary semaphore to solve critical section problem for 2 process - process P0 and process P1

b) implement counting semaphore to solve critical section problem to achieve multi process synchronization

CODE :- 6(A)

```
import time

class Semaphore:
    def __init__(self):
        self.value = 1

    def wait(self):
        while self.value == 0:
            pass
        self.value = 0

    def signal(self):
        self.value = 1

def critical_section(process_id):
    print(f"Process {process_id} is in critical section.")

def process(semaphore, process_id):
    while True:
        # Non-critical section
        print(f"Process {process_id} is in non-critical section.")

        # Entry section
        semaphore.wait()
        time.sleep(1)

        # Critical section
        critical_section(process_id)

        # Exit section
        semaphore.signal()
```

```
        # Remainder section
        break

if __name__ == "__main__":
    semaphore = Semaphore()

    process(semaphore, 0)
    time.sleep(1)
    process(semaphore, 1)
```

Output :-

```
Process 0 is in non-critical section.
Process 0 is in critical section.
Process 1 is in non-critical section.
Process 1 is in critical section.
```

CODE :- 6(B)

```
import heapq
# Global Variable to track the Processes going into Critical Section
COUNTER=1

class Semaphore:
        def _init_(self,value):
                # Value of the Semaphore passed to the Constructor
                self.value=value
                # The Waiting queue which will be using the heapq module of Python
                self.q=list()

        def getSemaphore(self):
                ''' Function to print the Value of the Semaphore Variable '''
                print(f'Semaphore Value: {self.value}')

def block(process):
        print(f'Process {process} Blocked.')

def wakeup(process):
        print(f'Process {process} Waked Up and Completed it's work.')
```

```
def P(s):
        global COUNTER
        s.value=s.value-1
        if(s.value<0):
                heapq.heappush(s.q,COUNTER)
                block(COUNTER)
        else:
                print(f'Process {COUNTER} gone inside the Critical Section.')
                COUNTER+=1
                return

def V(s):
        global COUNTER
        s.value=s.value+1
        if(s.value<=0):
                p=heapq.heappop(s.q)
                wakeup(p)
                COUNTER-=1
        else:
                print(f"Process {COUNTER} completed it's work.")
                COUNTER-=1
                return

# Can Pass the Value of the Counting Semaphore to the Class Constructor

# Example for Counting Semaphore value as 2
s1=Semaphore(2)
s1.getSemaphore()
P(s1)
s1.getSemaphore()
P(s1)
s1.getSemaphore()
P(s1)
s1.getSemaphore()
V(s1)
s1.getSemaphore()
V(s1)
s1.getSemaphore()
V(s1)
s1.getSemaphore()
```

Output :-

```
Semaphore Value: 2
Process 1 gone inside the Critical Section.
Semaphore Value: 1
Process 2 gone inside the Critical Section.
Semaphore Value: 0
Process 3 Blocked.
Semaphore Value: -1
Process 3 Waked Up and Completed it's work.
Semaphore Value: 0
Process 2 completed it's work.
Semaphore Value: 1
Process 1 completed it's work.
Semaphore Value: 2
```

Experiment 7

Aim:- To implement the banker's algorithms in any of the programming language.

Code: -

```
if __name__=="__main__":
	n = 5 # Number of processes
	m = 3 # Number of resources
	alloc = [[0, 1, 0 ],[ 2, 0, 0 ],
			[3, 0, 2 ],[2, 1, 1] ,[ 0, 0, 2]]
	max = [[7, 5, 3 ],[3, 2, 2 ],
			[ 9, 0, 2 ],[2, 2, 2],[4, 3, 3]]
	avail = [3, 3, 2] # Available Resources
	f = [0]*n
	ans = [0]*n
	ind = 0
	for k in range(n):
		f[k] = 0
	need = [[ 0 for i in range(m)]for i in range(n)]
	for i in range(n):
		for j in range(m):
			need[i][j] = max[i][j] - alloc[i][j]
	y = 0
	for k in range(5):
		for i in range(n):
			if (f[i] == 0):
				flag = 0
				for j in range(m):
					if (need[i][j] > avail[j]):
						flag = 1
						break
				if (flag == 0):
					ans[ind] = i
					ind += 1
					for y in range(m):
						avail[y] += alloc[i][y]
					f[i] = 1
	print("Following is the SAFE Sequence")
	for i in range(n - 1):
		print(" P", ans[i], " ->", sep="", end="")
```

```
        print(" P", ans[n - 1], sep="")
```

OUTPUT: -

```
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2
```