# EXPERIMENT 10 – PETERSON’S ALGORITHM

##### OBJECTIVES

* Learning Critical Section
* Learning how to implement Peterson’s algorithm **TIME REQUIRED** : 3 hrs **PROGRAMMING LANGUAGE** : C/C++/Java

**SOFTWARE REQUIRED** : Ubuntu/Fedora, gcc/gc, Windows, Dev, NetBeans

**HARDWARE REQUIRED** : Core i5 in Computer Labs

##### PRODUCER-CONSUMER PROBLEM:

In computing, the **producer–consumer problem** (also known as the **bounded-buffer problem**) is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer used as a queue. The producer's job is to generate data, put it into the buffer, and start again. At the same time, the consumer is consuming the data (i.e., removing it from the buffer), one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

##### TASK 10.1:

Create two processes (producer and consumer) or functions or threads that access the same shared variable. One process increments the variable while the other decrements it. The maximum value of the variable is 5 and the producer cannot produce anymore items if the variable is 5. The consumer cannot consume if the variable is zero.

##### EXERCISE 10.1 [2]

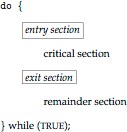
In the main function write a sequence of calls that simulate the producer consumer problem. Output the normal case, the case where the consumer should be blocked and the case where the producer should be blocked. Show the trace only.

##### EXERCISE 10.2 [2]

How does race condition occur in the cases of point 5? Explain and trace the case.

##### CRITICAL SECTION:

Access to a shared item is considered critical and is carried out in critical section only.



##### PETERSON’S SOLUTION:

It provides a solution to the Critical Section.

Two processes share two variables:

int **turn**; boolean **flag[2];**

//The variable **turn** indicates whose turn it is to enter the critical section

//The **flag** array is used to indicate if a process is ready to enter the critical section. //**flag[i]** = true implies that process **Pi** is ready!

##### ALGORITHM FOR PROCESS PI:

do {

flag[i] = True; turn = j;

while (flag[j] && turn = j){

//busy waiting

}

//go to critical section i.e., access the shared item flag[i] = FALSE;

// go to remainder section

} while (TRUE);

##### EXERCISE 10.3 [6]

Implement and show trace of Peterson’s Algorithm for two processes. Show code and output here.

##### RESOURCES

<https://en.wikipedia.org/wiki/Producer%E2%80%93consumer_problem> <https://en.wikipedia.org/wiki/Peterson%27s_algorithm>

<http://yab-jab.blogspot.com/2012/03/petersons-algorithm-mutex-for-2.html>