LAB-3 and Project Synopsis Submission (5 Marks)(15-21 March 2021)

Submit the Project Synopsis hard copy with following details:

1. Title of project
2. Problem statement
3. Project Description

Threads Creation

1. Write a program where 5 threads are created and each thread prints the thread Id and message "thread x executing".

2. Write a program where 2 threads are created and each thread prints information like (Name, Hall No., employee ID ,branch).

3. Write a program where 2 threads communicate using a single global variable "balance" and initialized to 1000..Thread 1 deposits amount = 50 for for 50 times and prints the balance amount and thread 2 withdrawals amount=20 for 20 times and prints the final balance .Execution of thread 1 and thread 2 should not interleave.

4. Write a program where 2 threads operate on a global variable "account" initialized to 1000. There is a deposit function which deposits a given amount in this "account": int deposit(int amount)

   There is a withdrawal function which withdraws a given amount from the "account":
   int withdrawal(int amount)

   However there is a condition: withdrawal function should block the calling thread when the amount in the "account" is less than 1000, i.e. you can't withdraw if the "account" value is less than 1000. Threads calling the deposit function should indicate to the withdrawing threads when the amount is greater than 1000.

5. Write a thread program which demonstrates how to "wait" for thread completions by using the Pthread join routine. Since some implementations of Pthreads may not create threads in a joinable state, therefore explicitly created attribute in a joinable state so that they can be joined later. Created thread should perform the calculation of sum =sum + sin(i) + Tan(i), where i =0 to 10000. Print the out in the following manner

Main: creating thread 0
Main: creating thread 1
Thread 0 starting...
Main: creating thread 2
Thread 1 starting...
Main: creating thread 3

Thread 2 starting...
Thread 3 starting...
Thread 1 done. Result = -3.153838e+06
Thread 0 done. Result = -3.153838e+06
Main: completed join with thread 0 having a status of 0
Main: completed join with thread 1 having a status of 1
Thread 3 done. Result = -3.153838e+06
Thread 2 done. Result = -3.153838e+06
Main: completed join with thread 2 having a status of 2
Main: completed join with thread 3 having a status of 3
Main: program completed. Exiting.

6. Write a thread program which demonstrates Pthreads condition variables. The main thread creates three threads. Two of those threads increment a "count" variable, while the third thread watches the value of "count". When "count" reaches a predefined limit, the waiting thread is signaled by one of the incrementing threads. The waiting thread "awakens" and then modifies count. The program continues until the incrementing threads reach TCOUNT. The main program prints the final value of count.

Synchronization-1

1. Given two character strings s1 and s2. Using C and pthread to write a parallel program to find out the number of substrings, in string s1, that are exactly the same as string s2. The strings are ended with '\0'. For example, suppose number_substring(s1, s2) implements the function, then number_substring ("abcdab", "ab") = 2, number_substring("aaa", "a")= 3, number_substring("abac", "bc") = 0. Suppose the size of s1 and s2 are n1 and n2, respectively, and p threads are used, we assume that n1 mod p = 0, and n2 < n1/p.Strings s1 and s2 are stored in a file named "strings.txt". String s1 is evenly partitioned among p threads to concurrently search for matching with string s2. After a thread finishes its work and obtains the number oflocal matching, this local number is addedinto a global variable showing the total numberof matched substrings in string s1. Finallythis total number is printed out. The format of the strings.txt is like this(the first string iss1 and the second one is s2):
   s1: Hello we are doing pthread testing with string.
   s2:in

   HINT: divide the s1 into two half and create two threads to search substring in this two half.

2. Demonstration of Race Condition in producer and consumer problem using thread implementation.

3. Write a program to implement producer consumer problem (Using POSIX semaphores)
   Description: Theproducer-consumer problem (Also called the bounded-buffer problem.) illustrates the need for synchronization in systems where many processes share a resource. In the problem, two processes share a fixed-size buffer. One process (producer) produces information and puts it in the buffer, while the other process (consumer) consumes

information from the buffer. These processes do not take turns accessing the buffer, they both work concurrently. Herein lies the problem. What happens if the producer tries to put an item into a full buffer? What happens if the consumer tries to take an item from an empty buffer? In order to synchronize these processes, we will block the producer when the buffer is full, and we will block the consumer when the buffer is empty. So the two processes, Producer and Consumer, should work as follows:

Procedure for doing the experiment:

1. Declare variable for producer & consumer as pthread-t-tid produce tid consume.
2. Declare a structure to add items, semaphore variable set as struct.
3. Read number of items to be produced and consumed.
4. Declare and define semaphore function for creation and destroy.
5. Define producer function.
6. Define consumer function.
7. Call producer and consumer.
8. Stop the execution.

4. Write a program to implement producer consumer problem (Using MUTEX semaphores).

5. Demonstrate dead lock in dining philosophers' problem.

Synchronization-2

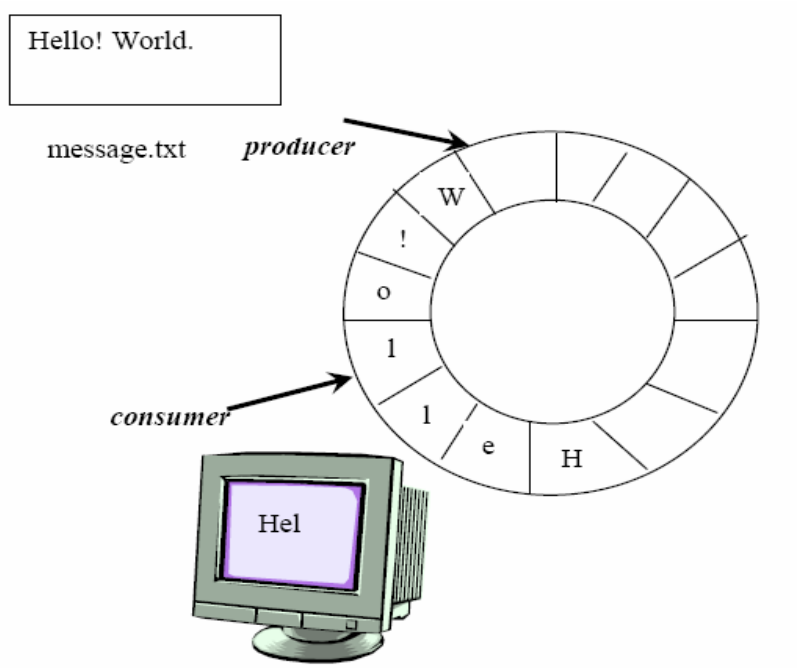
1. To avoid deadlock in dining philosophers' problem use a possible solution as the odd numbered philosophers grab the right and then the left. Implement this solution using pthread mutual exclusion lock.

2. Implementation of Boss/Worker Model: Here the idea is to have a single boss thread that creates work and several worker threads that process the work. Typically the boss thread creates a certain number of workers right away -even before any work has arrived. The worker threads form a thread pool and are all programmed to block immediately. When the boss thread generates some work, it arranges to have one worker thread unblock to handle it. When all workers be busy the boss thread might do one of the following by taking request from the user
    1. Queue up the work to be handled later as soon as a worker is free.
    2. Create more worker threads.
    3. Block until a worker is free to take the new work.

   If no work has arrived recently and there are an excessive number of worker threads in the thread pool, the boss thread might terminate a few of them to recover resources. In any case, since creating and terminate threads is relatively expensive (compared to, say, blocking on a mutex) it is generally better to avoid creating a thread for each unit of work produced.

3. Using condition variables to implement a producer-consumer algorithm. Define two threads: one producer and one consumer. The producer reads characters one by one from a string stored in a file named "string.txt", then writes sequentially these characters into a circular queue. Meanwhile, the consumer reads sequentially from the queue and prints them in the

same order. The diagram illustrates the process. Upon completion of running the program, “Hello! World.” is printed on the screen. In the program, use #define to specify the size of the queue. For example, #define QUEUE_SIZE 5. Make sure to test your program with different queue sizes, including 1.



4. Write a program to implement producer consumer problem (Using conditional semaphores and monitors)

5. The barber has one barber chair and a waiting room with a number of chairs in it. When the barber finishes cutting a customer's hair, he dismisses the customer and then goes to the waiting room to see if there are other customers waiting. If there are, he brings one of them back to the chair and cuts his hair. If there are no other customers waiting, he returns to his chair and sleeps in it.

   Each customer, when he arrives, looks to see what the barber is doing. If the barber is sleeping, then the customer wakes him up and sits in the chair. If the barber is cutting hair, then the customer goes to the waiting room. If there is a free chair in the waiting room, the customer sits in it and waits his turn. If there is no free chair, then the customer leaves. Based on a naïve analysis, the above description should ensure that the shop functions correctly, with the barber cutting the hair of anyone who arrives until there are no more customers, and then sleeping until the next customer arrives.

   Solve the above problem using conditional semaphore.