**Lecture 05 Assignments**

1. What is a *semaphore*? Complete the semaphoresolution of the **bounded buffer producer-consumer** problem by filling the blanks of the given *producer* and *consumer* structures:

A **bounded buffer** with ***n*** locations;

Semaphore **mutex** initialized to the value 1;

Semaphore **full** initialized to the value 0;

Semaphore **empty** initialized to the value ***n***;

**Producer process**

do { produce an Item;

1.1 ------------------------------------

1.2 ------------------------------------

add item to the buffer; *//Critical Section*

1.3 ------------------------------------

signal (full); // *signal to consumer that the buffer is full*

} while (TRUE);

**Consumer process**

do {

1.4 ------------------------------------

1.5 ------------------------------------

remove an item from the buffer; *//Critical Section*

signal (mutex); // *signal to producer that the mutex is free*

1.6 ------------------------------------

} while (TRUE);

2). Complete the semaphoresolution of *Readers priority situation*of **Readers-Writers** synchronization problem by filling the blanks of the structure of the  *Readers process* based on the following data:

i. Dataset

ii. Semaphore **rw\_mutex** initialized to 1

iii. Semaphore **mutex** initialized to 1

iv. Integer **read\_count** initialized to 0

**Readers process**

do {

*//first finding readers using mutex*

2.1 ----------------------------- read\_count++; // *find readers*

if (read\_count == 1) *//if at least one reader*

wait(rw\_mutex); // *then a writer should wait*

2.2 -----------------------------

/\* reading is performed \*/

2.3 -----------------------------

read count--; // *reading by readers*   
 if (read\_count == 0) // *if no more readers*

2.4 -----------------------------

signal(mutex); *// signal ‘mutex’ to synchronized writers*

} while (TRUE);

1. What are two differences between **user-level threads** and **kernel-level threads**? Under what circumstances is one type better than the other?
2. How do the **user-level threads** recognized by an OS for their execution?
3. How would the **user-level threads**, which are generated by a compiler recognized by an OS for their execution (hint: *you need to describe the three various user- level to kernel-level threads mapping schemes*)?

6). What is a Critical Section (CS)? How would the *semaphore* solve the issue(s) of the CS in a process synchronization problem?

7). How do the *user-level threads*recognized by an OS for their execution? Describe the importance of Light Weight Processes (LWPs) at this scenario.

8). What are the two models of InterProcess Communication (IPC)? What are the strengths and weaknesses of the two approaches?

9). Identify the nature of the process structure including its IPC model which is shown below:

while (true)

{

if (counter == BUFFER\_SIZE)

/\* do nothing \*/

Buffer[i] = next\_item;

in = (in + 1) % BUFFER\_SIZE;

counter++; }

10). Briefly describe the issue of **race condition** in a process synchronization problem.

11). What is the meaning of the term busy waiting? What other kinds of waiting are there in an operating system? Can busy waiting be avoided altogether? Explain your answer.

12). Show that, if the wait() and signal() semaphore operations are not executed atomically, then mutual exclusion may be violated.

13). Illustrate how a binary semaphore can be used to implement mutual exclusion among n processes.

14). The implementation of mutex locks suffers from busy waiting. Describe what changes would be necessary so that a process waiting to acquire a mutex lock would be blocked and placed into a waiting queue until the lock became available.