Debates/Discussions – Week 9

1. Explain how the Semaphore prevents busy waiting.
2. In class we discussed Incorrect use of semaphore operations: (i) **signal** …. **wait**, (ii) **wait** … **wait**, (iii) Omitting of **wait** or **signal** (or both). Explain what happens in each case
3. Explain why priority inversion happens, and how we can solve it.
4. Explain the *readers-writers problem*. Use Figure in next-page to explain. Use semaphores **wrt**, **mutex** and integer **readcount**. Consider the following cases where **R1, R2** are *readers* and **W1, W2** are *writers*. **(Ent)** means **enter**, **(Exit)** means **exit**.

   (1) **W1 (Ent) → W2 (Ent) → W1 (Exit) → W2 (Exit)**

   (2) **R1 (Ent) → R2 (Ent) → W1 (Ent)→ R1 (Exit) → R2 (Exit) → W1 (Exit)**

   (3) **W1 (Ent) → R1 (Ent) → R2 (Ent) → W1 (Exit) → R1 (Exit) → R2 (Exit)**

Readers-writers Problem

(1) W1 (Ent) → W2 (Ent) → W1 (Exit) → W2 (Exit)

(2) R1 (Ent) → R2 (Ent) → W1 (Ent)→ R1 (Exit) → R2 (Exit) → W1 (Exit)

(3) W1 (Ent) → R1 (Ent) → R2 (Ent) → W1 (Exit) → R1 (Exit) → R2 (Exit)

- The structure of a reader process

```
do {
        wait (mutex);
        readcount ++;
        if ( readcount == 1 )
                wait(wrt);
        signal(mutex)

         //  reading is performed

        wait(mutex);
        readcount --;
        if ( readcount == 0 )
                signal(wrt);
        signal (mutex);
} while (TRUE);
```

Figure 7.4

writer process

```
do
{
        wait (wrt);

        //  writing is performed

        signal (wrt);

} while (TRUE);
```

Figure 7.3