COSC1112/1114: Operating Systems Principles

Tutorial 05 (week 06)

1. **Race conditions are possible in many computer systems. Consider a banking system with two methods: deposit(amount) and withdraw(amount). These two methods are passed the amount that is to be deposited or withdrawn from a bank account. Assume that a husband and wife share a bank account and that concurrently the husband calls the withdraw() method and the wife calls deposit(). Describe how a race condition is possible and what might be done to prevent the race condition from occurring.**

Answer:
Assume the balance in the account is 250.00 and the husband calls withdraw(50) and the wife calls deposit(100). Obviously the correct value should be 300.00. Since these two transactions will be serialized, the local value of balance for the husband becomes 200.00, but before he can commit the transaction, the deposit(100) operation takes place and updates the shared value of balance to 300.00 We then switch back to the husband and the value of the shared balance is set to 200.00 - obviously an incorrect value.

2. **Assume that a finite number of resources of a single resource type must be managed. Processes may ask for a number of these resources and —once finished—will return them. As an example, many commercial software packages provide a given number of licenses, indicating the number of applications that may run concurrently. When the application is started, the license count is decremented. When the application is terminated, the license count is incremented. If all licenses are in use, requests to start the application are denied. Such requests will be granted only when an existing license holder terminates the application and a license is returned. The following program segment is used to manage a finite number of instances of an available resource. The maximum number of resources and the number of available resources are declared as follows:**

```
#define MAX_RESOURCES 5
int available_resources = MAX_RESOURCES;
```

When a process wishes to obtain a number of resources, it invokes the decrease_count() function:

```
/* decrease available resources by count resources */
/* return 0 if sufficient resources available, */
/* otherwise return -1 */
int decrease_count ( int count ) {
        if ( available_resources < count )
                return -1;
        else {
                available_resources - = count;
                return 0;
        }
}
```

When a process wants to return a number of resources, it calls the increase_count() function:

```
/* increase available resources by count */
int increase_count( int count ) {
        available_resources += count ;
        return 0;
}
```

The preceding program segment produces a race condition. Do the following:

a) **Identify the data involved in the race condition.**
b) **Identify the location (or locations) in the code where the race condition occurs.**
c) **Using a semaphore, fix the race condition.**

Answer:

a) Identify the data involved in the race condition: The variable available_resources.
b) Identify the location (or locations) in the code where the race condition occurs: The code that decrements available_resources and the code that increments available_resources are the statements that could be involved in race conditions.
c) Using a semaphore, fix the race condition: Use a semaphore to represent the available_resources variable and replace increment and decrement operations by semaphore increment and semaphore decrement operations.

3. Describe two kernel data structures in which race conditions are possible. Be sure to include a description of how a race condition can occur.

Answer:

There are many answers to this question. Some kernel data structures include a process id (pid) management system, kernel process table, and scheduling queues. With a pid management system, it is possible two processes may be created at the same time and there is a race condition assigning each process a unique pid. The same type of race condition can occur in the kernel process table: two processes are created at the same time and there is a race assigning them a location in the kernel process table. With scheduling queues, it is possible one process has been waiting for IO which is now available. Another process is being context-switched out. These two processes are being moved to the Runnable queue at the same time. Hence there is a race condition in the Runnable queue.

4. Discuss the tradeoff between fairness and throughput of operations in the readers-writers problem. Propose a method for solving the readers-writers problem without causing starvation.

Answer:

Throughput in the readers-writers problem is increased by favoring multiple readers as opposed to allowing a single writer to exclusively access the shared values. On the other hand, favoring readers could result in starvation for writers. To avoid starvation in the readers-writers problem, when a reader arrives and notices that another reader is accessing the database, it would enter the critical section only if there are no waiting writers. These restrictions would guarantee fairness.

5. Explain why implementing synchronization primitives by disabling interrupts is not appropriate in a single-processor system if the synchronization primitives are to be used in user-level programs.

Answer:

If a user-level program is given the ability to disable interrupts, then it can disable the timer interrupt and prevent context switching from taking place, thereby allowing it to use the processor without letting other processes execute.