Operating Systems (CSEx61)

Homework Assignment #5
Concurrency: Mutual Exclusion and Synchronization

Hatem Mohamed Ahmed (20010447)

5.1 MUTUAL EXCLUSION: SOFTWARE APPROACHES

Question 1 Define the followings:

1. **Mutual exclusion**:

> **Answer**
>
> - Mutual exclusion is a property of concurrent systems where only one process at a time is allowed to access a shared resource or section of code.
> - This prevents concurrent access that could lead to data corruption or inconsistency.

2. **Critical section**:

> **Answer**
>
> - A critical section is a part of a program that accesses shared resources and must be executed under mutual exclusion.
> - Only one process or thread can execute the critical section at a time to avoid conflicts and maintain data integrity.

Question 2 True or false:

1. **Peterson's algorithm can not be generalized to the case of processes**

> **Answer**
>
> **True.**
> Peterson's algorithm is typically used for thread synchronization in the context of shared memory systems and is not directly applicable to the general case of inter-process synchronization.

5.2 PRINCIPLES OF CONCURRENCY

Question 3 Define the followings:

1. **Race condition**:

Answer

- A race condition occurs in a concurrent system when the behavior of the system depends on the relative timing or interleaving of operations by multiple executing threads or processes.
- This can lead to unexpected and erroneous behavior of the system.

Question 4 What are the requirements for mutual exclusion:

Answer

1.

a) Exclusive access:

Only one process at a time can access the shared resource.

b) No deadlock:

Processes cannot be indefinitely blocked waiting for access to the resource.

c) No starvation:

Every process that requests access to the resource eventually gets it.

d) Progress: If no process is in the critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder section can participate in decisions regarding which will enter its critical section next, and this decision cannot be postponed indefinitely.

5.3 MUTUAL EXCLUSION: HARDWARE SUPPORT

Question 5 Machine-instruction approach that enforces mutual exclusion has a number of disadvantages.

1. **mention them:**

Answer

a) Lack of flexibility:

Hardware-based mutual exclusion solutions are often fixed and may not be easily changed or adapted to different synchronization requirements.

b) Complexity:

Hardware-based solutions for mutual exclusion can be complex and may require specialized hardware support, leading to increased system complexity and potentially higher costs.

c) Limited scalability:

Hardware-based approaches may not scale well to support a large number of concurrent processes or threads, leading to potential performance limitations in highly concurrent systems.

d) Lack of portability:

Hardware-based mutual exclusion solutions may not be portable across different hardware architectures, limiting their applicability in heterogeneous systems.

Question 6 Comparison of Semaphore Definitions: Note one difference: With the preceding definition, a semaphore can never take on a negative value. Is there any difference in the effect of the two sets of definitions when used in programs?
That is, could you substitute one set for the other without altering the meaning of the program?

Listing 1: Semaphore Program 1

```
1. struct semaphore {
     int count;
     queueType queue;
   };

   void semWait(semaphore s) {
     s.count--;
     if (s.count < 0) {
       /* place this process in s.queue */
       /* block this process */
     }
   }

   void semSignal(semaphore s) {
     s.count++;
     if (s.count <= 0) {
       /* remove a process P from s.queue */
       /* place process P on ready list */
```

```
    }
  }
```

Listing 2: Semaphore Program 2

2.
```
void semWait(s)
{
  if (s.count > 0)
    s.count--;
  else {
    place this process in s.queue;
    block;
  }
}

void semSignal(s)
{
  if (there is at least one process blocked on semaphore s) {
    remove a process P from s.queue;
    place process P on ready list;
  }
  else
    s.count++;
}
```

Answer

- The First :
- Semaphore count can take negative value resulting in blocking the running process after calling `semWait` and places the process in the queue.
- But when calling `semSignal` the blocked processes that are waiting to be unblocked are unblocked and places in the ready list to execute their critical section next .

- The Second :
- The same as before but semaphore count cannot take a negative value .

The main difference between both is that the negative value semaphore count take can be indecation to the number of the waiting processes in the waiting queue but both have the same functionality and can be replaced by the other.

General Questions

7. Consider the following program:

Listing 3: Concurrent Program

```
const int n = 50;
int tally;

void total() {
    int count;
    for (count = 1; count <= n; count++) {
        tally++;
    }
}

void main() {
    tally = 0;
    parbegin (total (), total ());
    write (tally);
}
```

1. **Determine the proper lower bound and upper bound on the final value of the shared variable tally output by this concurrent program.**
   Assume processes can execute at any relative speed and that a value can only be incremented after it has been loaded into a register by a separate machine instruction.

   Answer

   -In case of parallel process execution.
   - The lower bound on the final value of tally would be 50.
   - The upper bound on the final value of tally in this case would be 100, as each process can increment tally up to 50 times.
   But, if the processes execute truly in parallel, both processes could read the current value of tally at the same time, increment their respective local copies, and then write back the same value to tally, causing one or more of the increments to be lost.

2. **Suppose that an arbitrary number of these processes are permitted to execute in parallel under the assumptions of part (a).**
   What effect will this modification have on the range of final values of tally?

   Answer

- It can increase the upper bound on the final value of tally. But this will cause more of the increments to be lost.

8. Consider the following program:

Listing 4: Mutual Exclusion Program

```
boolean blocked[2];
int turn;

void P(int id) {
    while (true) {
        blocked[id] = true;
        while (turn != id) {
            while (blocked[1-id])
            // do nothing;
            turn = id;
        }

        /* critical section */
        blocked[id] = false;
        /* remainder */
    }
}

void main() {
    blocked[0] = false;
    blocked[1] = false;
    turn = 0;
    parbegin (P(0), P(1));
}
```

1. **This software solution to the mutual exclusion problem for two processes is proposed, Find a counterexample that demonstrates that this solution is incorrect.**

Answer

Counterexample:

- If process P(1) starts first and continue executing to the part where $while(turn! = id)$ is $false$ as $turn$ is initialized by zero so it will stuck in the $while(turn! = id)$ loop .
- Then the process P(0) starts execution making ($blocked[0]$) is $false$. - Parallely the condition ($blocked[1 - id]$) is $false$ which exits the nested loop in

process P(1) executing $turn = 1$ which makes the condition $while(turn! = id)$ set to $false$ in the first loop as $id = turn = 1$.

- Then the process P(1) reaches the line of the code where it enters the critical section.
- Parallely the process P(0) also reaches the line of the code where it enters the critical section.
- Both Processes now can acess critical section making the code not correct to use.