COMP-3500

Introduction to Operating Systems

FROM: Midas Oden

TO: Dr. Xiao Qin

DATE: September 17, 2021

LAB SECTION: 001

Homework #1

Problem 1

[60 points] Consider the following program:

```
P1: {
  shared int x;
  x = 10;
  while (1) {
     x = x - 1;
     x = x + 1;
     if (x != 10)
       printf("x is %d",x)
  }
}
```

```
P2: {
  shared int x;
  x = 10;
  while ( 1 ) {
    x = x - 1;
    x = x + 1;
    if (x!=10)
      printf("x is %d",x)
  }
}
```

1.1 [25 points] Show a sequence (i.e., trace the sequence of interleavings of statements) such that the statement "x is 10" is printed.

→ The scheduler is a uniprocessor system, i.e. it will only execute one instruction at a given time. We are given two process P1 and P2 in which both processes use a shared variable "x" and will initialise the value of "x = 10" while executing their instruction concurrently.

Process Number	Instruction	X-value
P1	x = x - 1;	9
P1	x = x + 1;	10
P2	x = x - 1;	9
P1	if (x != 10)	9
P2	x = x + 1;	10
P1	printf ("x is %d", x)	10
		"x is 10" is printed

1.2 [35 points] Show a sequence such that the statement "x is 8" is printed.

Process Number	Instruction	X-value	Register R0 for P1	Register R0 for P2
P1	LD R0, X	10	10	
P1	DECR R0	10	9	
P1	STO R0, X	9	9	
P2	LD R0, X	9	9	9
P2	DECR R0	9	9	8
P2	STO R0, X	8	8	8
P1	LD R0, X	8	9	8
P1	INCR R0	8	9	8
P2	LD R0, X	8	9	8
P2	INCR R0	8	9	9
P2	STO R0, X	9	9	9
P2	if (x!= 10) printf ("x is%d", x);	9	9	9
		"x is 9" is printed		
P1	LD R0, X	9	9	9
P1	DECR R0	9	8	9
P1	STO R0, X	8	8	
P2	LD R0, X	8	8	8
P2	DECR R0	8	8	7
P2	STO R0, X	7	8	7
P1	LD R0, X	7	7	7
P1	INCR R0	7	8	7
P1	STO R0, X	8	8	7
P1	if (x!= 10) printf ("x is%d", x);	**"x is 8" is printed**		

Problem 2

[10 points] What is the difference between binary and general semaphores?

- **Binary Semaphore:**
  - **Binary semaphores can only hold the values 0 or 1.**
  - **It is nothing, but similar to a lock, with two values: 0 and 1. Here 0 means blocked or busy, while 1 means unblocked or not busy.**
  - **The value 0 means that a process or a thread is accessing the critical section, other process should wait for it to exit the critical section. The value 1 represents the critical section is free.**
  - **Allows only one process at a time to enter the critical section(thus allowing it to access the shared resource).**
  - **Guarantees mutual exclusion since no two processes can be in the critical section at any point in time.**
  - **Doesn't guarantee bounded wait**

```
typedef struct {
        int semaphore_variable;
}binary_semaphore;
```

Figure 1: Structure implementation of a binary semaphore

- **General Semaphore:**
  - **A counting semaphore is a semaphore that has multiple values of the counter. The value can range over an unrestricted domain.**
  - **It is a structure, which comprises a variable, known as a semaphore variable that can take more than two values and a list of task or entity, which is nothing but the process or the thread.**
  - **The value of the semaphore variable is the number of process or thread that is to be allowed inside the critical section.**
  - **The value of the counting semaphore can range between 0 to N, where N is the number of the number of process which is free to enter and exit the critical section.**
  - **Doesn't guarantee mutual exclusion, since more than one process or thread can enter the critical section at a time.**
  - **Guarantees bounded wait**

```
typedef struct {
        int semaphore_variable;
        //A queue to store the list of task
        Queue list;
}counting_semaphore;
```

Figure 2: Structure implementation of a counting semaphore

Problem 3

[10 points] What is a monitor?

→ A monitor is high-level abstraction that provides a convenient and effective mechanism for process synchronisation.

→ It allows threads to have both mutual exclusion and the ability to wait (block) for a certain condition to become false ... a thread-safe class, object, or module that wraps around a mutex in order to safely allow access to a method or variable by more than one thread.

→ It encapsulates variables, access procedures, and initialization code within an abstract data type. In other words, i.e. it represents a software module consisting of one or more procedures, an initialization sequence, and local data.

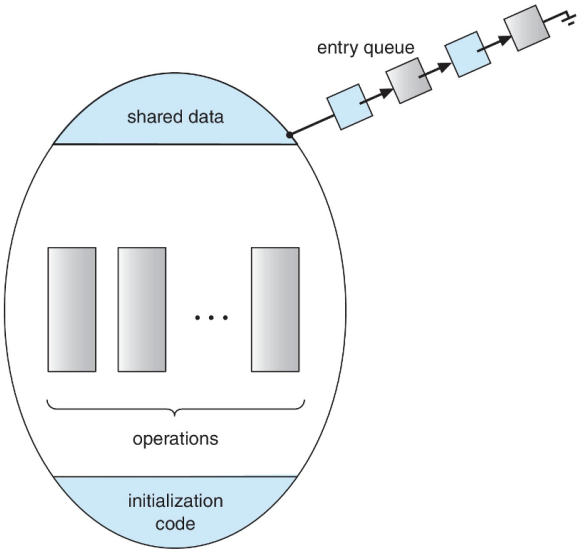

Figure 3: Schematic view of a monitor

```
monitor monitor-name {
        // shared variable declarations
        procedure P1 (...) { ... }
        procedure P2 (...) { ... }
        procedure P3 (...) { ... }
                    .
                    .
                    .
                    .
                    .
        procedure Pn (...) { ... }

        initialisation code (...) { ... }
}
```

Figure 4: Pseudocode syntax of a monitor

Problem 4

[20 points] What operations can be performed on a semaphore?

→ A Semaphore is a synchronisation tool which uses integer variables to solve the critical section problem; while also providing a sophisticated way (other than the Mutex locks) for processes to synchronise their activities.

→ Semaphore (S) can only be accessed along two inseparable atomic operations:

- **wait() or P() — assists in controlling the entry of a task into the critical section. If the value of wait is positive, then the value of the wait argument is decremented by 1. If the new value of the semaphore variable is negative or zero value, the process executing wait() is blocked (i.e., added to the semaphore's queue). Otherwise, the process continues execution, having used a unit of the resource.**

```
wait(S)
{
   while (S<=0);

   S--;
}
```

- **signal() or V() — is used to control the exit of a task from a critical section. It increments the value of semaphore variable by 1. After the increment, if the pre-increment value was negative (meaning there are processes waiting for a resource), it transfers a blocked process from the semaphore's waiting queue to the ready queue.**

```
signal(S)
{
   S++;
}
```

→ Both operations are used to implement process synchronisation and the goal is to obtain mutual exclusion.