ICS EXE 13

June 3, 2020

1

Please fill in the blanks with initial values for the three semaphores and add `P()` and `V()` semaphore operations such that the process is guaranteed to terminate. (NOTE: You can only fill in one `P(x)` or `V(x)` operations)

```
1  /* Initialize x */
2  int x = 1;
3
4  /* Initialize semaphores */
5  sem_t a, b, c;
6  sem_init(&a, 0, 0);
7  sem_init(&b, 0, 1);
8  sem_init(&c, 0, 0); /* Not used */
9
10 void thread1()
11 {
12     while (x != 12) {
13         x = x * 2;
14         V(a);
15         P(b);
16     }
17     exit(0);
18 }
19
20 void thread2()
21 {
22     while (x != 12) {
23         P(a);
24         P(a);
25         x = x * 3;
26         V(b);
27     }
28     exit(0);
29 }
```

2

The following code implements a simple stack.

```
1  typedef struct Node {
2          struct Node *next;
3          int value;
4  } Node;
5
6  void push(Node **top_ptr, Node *n) {
7          n->next = *top_ptr;
8          *top_ptr = n;
9  }
10
11 Node *pop(Node **top_ptr) {
12         if (*top_ptr == NULL)
13                 return NULL;
14         Node *p = *top;
15         *top_ptr = (*top_ptr)->next;
16         return p;
17 }
```

2.1

Is this implementation thread-safe?

No. Multiple threads may modify the `*top_ptr` simultaneously.

2.2

Use semaphore to protect the operations towards the stack.

Create a new struct – `stack` which contains a `top` pointer points to the top node of the stack and a `sem` which protects the stack operation. For all stack operation, `P(stack->sem)` before start and `V(stack->sem)` after finish.

3

Initially, counting semaphore S is initialized with value 2. What is the maximum possible value of x after all the processes complete execution.

Process 1	Process 2	Process 3	Process 4
P(S)	P(S)	P(S)	P(S)
x = *addr	x = *addr	x = *addr	x = *addr
x = x + 1	x = x + 1	x = x - 2	x = x - 2
*addr = x	*addr = x	*addr = x	*addr = x
V(S)	V(S)	V(S)	V(S)

To obtain the maximum value of x, the processes must execute in such a way that only the impact of the processes 1 and 2 remains on the value of x. The impact of processes 3 and 4 gets lost on the value of x.

One Possible solution is:

1. Block P1 after execute `x = *addr`.
2. Allow P3 and P4 finish their execution.
3. P1 can continue execute and `*addr = x + 1`.
4. P2 can execute and `*addr` now is the old value + 2.