LE19.2.1: Mutual Exclusion

0.0/1.0 point (ungraded)

You are experimenting with a pair of simple processes, A and B, that share an integer variable X as well as a semaphore S:

```
Process A

int Y;
wait(S);
Y = X + 1;
X = Y;
signal(S)
```

```
Shared Memory

int X = 1;
Semaphore S = 1;
```

```
Process B

int Z;
wait(S);
Z = X * 5;
X = Z;
signal(S);
```

The processes run on a timeshared system whose scheduler promises no constraints on process scheduling other than those imposed by the use of semaphores by the processes. Note that each process uses a local variable, in addition to the shared variable X. The two processes are run simultaneously; execution of their statements may be interleaved. You may assume that the execution of each line of code is atomic (that is, un-interrupted by the scheduler). Note that the initial values of X and S are both 1.

1. After execution of the above pair of processes, what final values of the variable X are possible? Select all possible values. Select “None” if one or both processes will never run to completion.

   Possible final values of X, or “None”:

   - [ ] 0
   - [ ] 1
   - [ ] 2
   - [ ] 3
   - [ ] 4
   - [ ] 5

- [x] 6
- [ ] 7
- [ ] 8
- [ ] 9
- [x] 10
- [ ] None

2. Suppose the initial value of the semaphore in the above code is changed to 2. What final values are now possible for X?

Possible final values of X, or "None":

- [ ] 0
- [ ] 1
- [x] 2
- [ ] 3
- [ ] 4
- [x] 5
- [x] 6
- [ ] 7
- [ ] 8
- [ ] 9
- [x] 10

- [ ] None

Finally, you use 0 as the initial value of S, and delete two lines of code, resulting in:

```
  Process A

  int Y;
  wait(S);
  Y = X + 1;
  X = Y;
  // Line
removed
```

```
  Shared Memory

  int X = 1;
  Semaphore S = 0;  // Semaphore
initialized to 0
```

```
  Process B

  int Z;
  // Line
removed
  Z = X * 5;
  X = Z;
  signal(S);
```

3. After execution of the above pair of processes, what final values of the variable X are possible? Select all possible values. Select “None” if one or both processes will never run to completion.

Possible final values of X, or “None”:

- [ ] 0
- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] 5
- [x] 6
- [ ] 7
- [ ] 8
- [ ] 9

- [ ] 10
- [ ] None

Submit

LE19.2.2: Using Semaphores

0.0/1.0 point (ungraded)

In lecture, you saw the use of semaphores to mediate a communication stream between a Producer and a Consumer process. In this problem, we assume the existence of three asynchronous processes: a Producer process, producing a stream of characters; a Consumer process, which consumes a stream of characters; and a Filter process spliced between the Consumer and Producer processes. The Filter process takes characters from the producer, processes them (via a **translate** function), and passes the result to the Consumer process. The Producer and Consumer processes each communicate directly only with the Filter process.

The following is in Shared Memory (shared among Producer, Filter, and Consumer processes):

```
Semaphore charsA=???, spaceA=???, charsB=???, spaceB=???;
char buf[100];
char indata;
int in=0, out=0;
```

and the following code runs in the Filter Process:

```
while (1) {                  /* loop forever… */
   char temp;                /* local variable */

   wait(charsA);
   temp = indata;
   signal(spaceA);
   temp = translate(temp);   /* do the actual translation */
   wait(spaceB);
   buf[in] = temp;
   in = (in+1)%100;          /* increment 'in' modulo 100 */
   signal(charsB);
}
```

1. What is the maximum number of characters that can be produced by the Producer process but not yet processed by the Filter process?

Maximum unprocessed characters produced: Answer: 1

2. What are appropriate initial values for each of the semaphores?

Initial value for charsA: Answer: 0

Initial value for spaceA: Answer: 1

Initial value for charsB: Answer: 0

Initial value for spaceB: Answer: 100

Explanation
By closely examining the filter code, you can deduce that the Producer will put a character into shared memory location indata which is a single character buffer. The filter process needs to wait for a character to be available in indata, it then copies it to its local variable temp, and then signals to the producer that it can produce another character by calling signal(spaceA). This behavior tells us that at most 1 character can be produced by the processor but not yet be processed.
The filter process then translates the value received by the producer and makes it available to the consumer by copying the result into the consumers buffer. From the code we see that the consumer's buffer is a 100 character array named buf. Before sending a value to the consumer, the filter process must ensure that there is space available in the consumer's buffer. After adding a value to the buffer, it needs to signal to the consumer that a value is available by calling signal(charsB).
The initial value of the four semaphores needs to indicate how much space and characters are available for the producer and consumer. We saw that the producer uses a 1 character buffer, so initially it has 1 space available and 0 characters. The consumer, however, uses a 100 character buffer so it initially has 100 spaces available and 0 characters.

Submit

Answers are displayed within the problem

Discussion

Topic: 19. Concurrency and Synchronization / LE19.2

Hide Discussion

Add a Post

Show all posts by recent activity

[STAFF] Setting up LE19.2.2, Producer/Filter/Consumer

Community TA

10