OS 3/31

Reagan Shirk

March 31, 2020

Chapter 6: Synchronization

Prodcuer-Consumer Problem

- You have one or more processes that are generating data and placing the data in a buffer
- In the case discussed last week, you have a single consumer that is taking items out of the buffer, one item at a time
- Constraint: Only one producer or consumer can access the buffer at a time
- You could make a pretty penny if you solve this problem, it's a big one
- Example
  - Let's say your buffer is an infintely large array, how would you solve the prodcuer/consumer problem in this case?
    - One approach: you have an index `in` that tells you where the producer going to put the next item. The consumer also has a pointer `out` that tells you where the consumer should consume
      - Constraints: if `out` $\geq$ `in` you have an issue, we need to make sure `out` $<$ `in`
      - Not how the real world works
    - Another approach: you now have a bounded array, but it is otherwise the same as above
      - Constraints: we need to make sure that `in` remains less than or equal to the buffer size, aka you have limited space- you can get stuck if the consumer ends up right behind the producer at the end of the buffer. We don't end up maximizing the array size
    - Another approach: make it circular!
      - Constraints: we need to make sure that `in` % `b-1` $<$ `out` % `b-1` (where `b` is the maximum length of the array)
      - Pseudocode for this scenario:

```
Producer:

while(true)
{
    produce
    while (count == buffsize){}
    buffer[in] = newitem;
    in = (in + 1) % buffsize;
    count++;
}

Consumer:

while(true)
{
    while(count == 0){}
    newitem = buffer[out];
    out = (out + 1) % buffsize;
    count--;
    consume
}
```

        - `count` is now a liability, `count++/count--` isn't really how it's implemented in most processors, it's translated to `register A = count; register A = register A +/- 1; count = register A`
- Race condition: The last change to a value wins; when multiple processes change a value and the order of scheduling effects the value of the shared variable

Synchronization/Coordination

- Critical section: where you access or change shared variables
- Mutual exclusion: For a process $p \in P$, only one process may be in its critical section at a time
  - Assumes that the time spent in the critical section is finite
- Bounded waiting: Limits the number of times or length of time that a process p can spend in its critical section
- Progress: a process cannot be denied entry into its critical section indefinitely