CIS 452 - Operating Systems Concepts

Nathan Bowman

Images taken from Silberschatz book

Critical Sections

Recall the **race condition** example where two concurrent processes modify shared variable `counter`

Process A: `counter++`

Process B: `counter--`

Final result can change based on interrupts

Let us generalize this idea

`N` processes (`P0`, `P1`, ..., `P(N-1)`) are executing concurrently

Each process has a section of code in which it is accessing shared data (e.g, modifying shared memory, writing to shared file, ...)

This code is the **critical section**

To run safely, must ensure that *only one process can be in its critical section at once*

If process A is in its critical section, process B cannot be in its own critical section

Designing a protocol that processes can follow to ensure this condition is called the **critical section problem**

Each process must determine whether it is safe before entering its critical section and must somehow indicate when it has left its critical section

```
do {
```

entry section

critical section

exit section

remainder section

```
} while (true);
```

Solution must satisfy three criteria:

- **Mutual exclusion**
- **Progress**
- **Bounded waiting**

Mutual exclusion: no two processes may be in their critical sections at the same time

Progress: no process in its remainder section may influence the selection of which process enters its critical section next, and the selection cannot be postponed indefinitely

Bounded waiting: there exists an upper bound on the number of times another process (process B) can enter its critical section after a process (process A) has requested access to its critical section before the requesting process (process A) is granted access

Assumptions: each process is operating at nonzero speed, but *we can make no assumption about the relative speeds of the processes*

Software-based solution known as **Peterson's solution**

Works for two processes (though there is a generalization to more)

Requires that processes share

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

in addition to whatever variables they share in their critical sections

`flag[i]` indicates that process `i` wishes to enter critical section

`turn` indicates which process is allowed to enter its critical section *if both processes wish to enter*

Remember that we must ensure: mutual exclusion, progress, and bounded waiting

```
do {
        flag[i] = true;
        turn = j;
        while (flag[j] && turn == j);

            critical section

        flag[i] = false;

            remainder section

} while (true);
```

Code for process `i`

We solved the critical section problem! Wait, why did we do that again...?

Remember: all accesses to shared resources (variables, files, etc.) must be in the critical section

This solution is limited in its scope, and it is not effective on modern multiprocessor architectures

Hardware support will be necessary to allow us to effectively solve this problem