Critical Regions

CS 447
Monday 3:30-5:00
Tuesday 2:00-3:30
What are the drawbacks of the algorithmic solutions?

- i.e. solutions with shared variables and atomic read and write?
  - Scalability: No of processes is to be known statically
  - Busy wait
  - Responsibility of implementation is with user

- Pointers to OS-supported solution?
Care to be taken with Semaphores (drawbacks)

- User programs must still use P and V correctly
- A for gotten P, or a misplaced V
- Possibility of deadlocks-

\[
P(S1) \rightarrow P(S2) \\
P(S2) \rightarrow P(S1)
\]
Better Higher level synchronization primitives?

- Critical Regions
- Conditional Critical Regions
- Monitors

- These were supported in concurrent programming languages
- Today’s semaphore system calls allow monitor type synchronization as well
Critical Regions

Shared variable v

region v do .... done

region v do .... done
Producer-consumer code with CRs

- **Producer:**
  - While (true) region buff if (!full) produce done

- **Consumer**
  - While (true) region buff if (!empty) consume done
Conditional Critical Regions

Region v when C
Do......done

Region v when C
Do......done

ability to block

Try the Producers and Consumers problem
With conditional critical regions
Producer-consumer code with CCRs

- Producer:
  - While (true) region buff when (!full) do produce done

- Consumer
  - While (true) region buff when (!empty) do consume done
Readings

- Hoare: Towards a theory of parallel programming, 1971
- Hansen: Structured multiprogramming, 1993