CSE 421 Recitation Week 6

October 7th, 2014

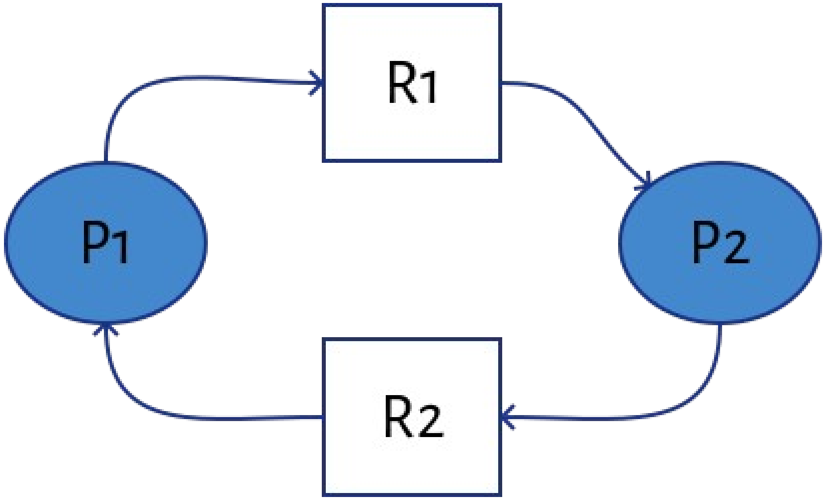
Kyungho Jeon
(kyunghoj@buffalo.edu)

Agenda

- Topics
  - Semaphore
  - Deadlock
  - Dining Philosophers Problem
- Homework #2
  - Problem #6 ~ #8

Deadlock

- We say…
  - A set of processes is in a deadlock state when every process in the set is waiting for an event that can be caused only by another process in the set



Resource Allocation Graph

- $G=\{V, E\}$
- Two types of V:
  - Process: $P = \{P1, P2, \ldots, Pn\}$
  - Resource type: $R = \{R1, R2, \ldots, Rm\}$
- Pi → Rj: Resource Request
- Ri->Pj: Resource Assignment

Four Necessary Conditions for Deadlocks

- Mutual exclusion
  - Resource is non-sharable
- Hold-and-Wait
  - Holding a resource and waiting to acquire additional resource
- Non-preemptive
  - Resource can be released only voluntarily
- Circular wait
  - A set {P0, P1, …, Pn} of waiting processes must exist such that P0 is waiting for a resource that is held by P1, …

Methods for Handling Deadlocks

- Deadlock Prevention
  - Ensure that at least one of the necessary conditions cannot hold
- Deadlock Avoidance
  - Assume O/S is given resource request/usage information
  - Decide whether the current request can be satisfied or must be delayed

Problem 6

- Consider the dining-philosophers problem where the chopsticks are placed at the center of the table and any two of them could be used by a philosopher. Assume that requests for chopsticks are made one at a time. Describe ***a simple rule*** for determining whether a particular request could be satisfied without causing deadlock given the current allocation of chopsticks to philosophers.

- Do not satisfy the request only if there's no philosopher with two chopsticks and if there is only one chopstick remaining

Problem 7

- Show that, if the `wait()` and `signal()` semaphore operations are not executed atomically, then mutual exclusion may be violated.

(Classical Definition of) Semaphore

```
wait(S) {
  while (S<=0)
    ; // no-op
  S--;
}
```

```
signal(S) {
  S++;
}
```

(A Better) Semaphore – 1/2

```
typedef struct {
  int val;
  struct process *L;
} semaphore;
```

- Assume a C-style struct for semaphores
- Each semaphore has a list of processes as well as a integer value

(A Better) Semaphore – 2/2

```
wait(S) {
   S.val--;
   if (S.val < 0){
      Add this P to S.L;
      block()
   }
}
```

```
signal(S) {
   S.val++;
   if (S.val <= 0) {
      Remove P from S.L;
      wakeup(P);
   }
}
```

Atomicity Requirement for Semaphores

- "No two processes can execute **wait()** and **signal()** operations on the ***same semaphore*** at the ***same time***."
  - A critical-section problem, again.

Problem 8

- Write a monitor that implements an alarm clock that enables a calling program to delay itself for a specified number of time units (*ticks*). You may assume the existence of a real hardware clock that invokes a procedure tick in your monitor at regular intervals.

```
monitor alarm {
  condition c;

  void delay ( int ticks ) {
    int begin_time = read_clock ();
    while (read_clock () < begin_time + ticks)
      c.wait();
  }

  void tick () {
    c.broadcast();
  }
}
```

Deadlock Avoidance

- Resource-allocation Graph Algorithm
  - Only one instance of each resource type
  - Maintain wait-for graph

Banker's Algorithm

- Resource allocation system with multiple instance of each resource type
- Keeps the following data structures
  - Available[m]
    - Number of available resources of each type
  - Allocation[n][m]
    - Number of resources of each type currently allocated to each P
  - Request[n][m]
    - The current request of each process.
    - e.g., Request[i][j] = k, Pi is requesting k more instances of Rj

Detection Algorithm

- Step 1: Initialize
  - Work := Available
  - finish[n] := [0, ..., 0]
- Step 2: Find an index *i* such that both:
  - Finish[i] == false
  - Request[i] <= Work
  - If no such *i* exists, go to step 4.

Detection Algorithm

- Step 3: Update
  - Work = Work + Allocation[i]
  - Finish[i] = true
  - Go to step 2
- Step 4:
  - If Finish[i] == falsefor some I, 0 <= I <= n-1, then the system is in deadlock state.