Introduction to Operating Systems
CS 1550

Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Some slides are from **Silberschatz, Galvin and Gagne ©2013)**

Announcements

- Upcoming deadlines
  - Homework 5 is due **this Friday**
  - Project 1 is due on Friday 2/17 at 11:59 pm
  - Lab 2 is due on Tuesday 2/28 at 11:59 pm

Previous lecture …

- Dining philosophers
- Deadlock prevention
- Banker's Algorithm for deadlock detection and avoidance

Banker's Algorithm

How to detect deadlocks?

How to avoid deadlocks?

Banker’s Algorithm

We can use the same algorithm for both detecting and avoiding deadlocks

Banker's Algorithm

	A	B	C	D
Avail	2	3	0	1

Hold

Process	A	B	C	D
1	0	3	0	0
2	1	0	1	1
3	0	2	1	0
4	2	2	3	0

Want

Process	A	B	C	D
1	3	2	1	0
2	2	2	0	0
3	3	5	3	1
4	0	4	1	1

```
current=avail;
for (j = 0; j < N; j++) {
 for (k=0; k<N; k++) {
  if (finished[k])
   continue;
  if (want[k] <= current) {
   finished[k] = 1;
   current += hold[k];
   break;
  }
 }
 if (k==N) {
   printf "Deadlock!\n";
   // finished[k]==0 means process is in
   // the deadlock
   break;
 }
}
```

Note: want[j], hold[j], current, avail are arrays!

Banker's Algorithm Insights

- It is possible that some event sequences lead to a deadlock
- What we are looking for is **at least one** event sequence that can make all processes finish
  - If such sequence exists, the state is safe
  - The Banker's algorithm finds such sequence if it exists

Using the Banker's Algorithm for Deadlock Avoidance

- Call the algorithm on the following ``What-if" state instead of the current state

avoiding deadlocks

Request from Process i

avail` = avail − Request

hold`[i] = hold[i] + Request

Want`[i] −= want[i] − Request

Run algo on avail`, hold`, Want`

Problem of the Day: Sleepy Barbers

- We have two sets of processes
  - Worker processes (e.g., barbers)
  - Customer processes
- Customer processes may arrive at anytime
- Worker processes check in when they are not serving any customers
- Each worker process must wait until it gets matched with a customer process
- Each customer process must wait until it gets matched with a worker process
- The customer process cannot leave until the matched worker process finishes the work
- The worker process cannot check in for the next customer until the matched customer process leaves
- Many applications in the real-world

Rendezvous Pattern

Sleepy Barber

Worker | Customer

~~arrive~~ checks | arrives

Rendez-vous

do work | get served

Rendez-vous

checks for next customer | leaves

Solution Using Semaphores: Take 1

- One pair of semaphores per rendezvous
  - RV1a and RV1b
  - RV2a and RV2b
- Notice the flipped order of the down and up calls in the two processes

Semaphore RV1a, RV1b
(0) (0)

RV2a, RV2b
(0) (0)

Worker	Customer
arrives/checks in	arrives
down(RV1a)	up(RV1a)
up(RV1b)	down(RV1b)
does work	gets served
up(RV2a)	down(RV2a)
down(RV2b)	up(RV2b)

Solution Using Semaphores: Take 1

- This solution doesn't work for multiple workers and multiple customers
  - In that case, a customer can leave before its associated worker finishes