

CLOUD COMPUTING CONCEPTS with Indranil Gupta (Indy)

MUTUAL EXCLUSION

Lecture A

INTRODUCTION AND BASICS

WHY MUTUAL EXCLUSION?

- Bank's Servers in the Cloud: Two of your customers make simultaneous deposits of \$10,000 into your bank account, each from a separate ATM.
 - Both ATMs read initial amount of \$1000 concurrently from the bank's cloud server
 - Both ATMs add \$10,000 to this amount (locally at the ATM)
 - Both write the final amount to the server
 - What's wrong?

WHY MUTUAL EXCLUSION?

- Bank's Servers in the Cloud: Two of your customers make simultaneous deposits of \$10,000 into your bank account, each from a separate ATM.
 - Both ATMs read initial amount of \$1000 concurrently from the bank's cloud server
 - Both ATMs add \$10,000 to this amount (locally at the ATM)
 - Both write the final amount to the server
 - You lost \$10,000!
- The ATMs need *mutually exclusive* access to your account entry at the server
 - or, mutually exclusive access to executing the code that modifies the account entry

MORE USES OF MUTUAL EXCLUSION

- Distributed File systems
 - Locking of files and directories
- Accessing objects in a safe and consistent way
 - Ensure at most one server has access to object at any point of time
- Server coordination
 - Work partitioned across servers
 - Servers coordinate using locks
- In industry
 - Chubby is Google's locking service
 - Many cloud stacks use Apache Zookeeper for coordination among servers

PROBLEM STATEMENT FOR MUTUAL EXCLUSION

- *Critical Section* Problem: Piece of code (at all processes) for which we need to ensure there is at most one process executing it at any point of time.
- Each process can call three functions
 - enter() to enter the critical section (CS)
 - AccessResource() to run the critical section code
 - exit() to exit the critical section

OUR BANK EXAMPLE

```
ATM1:

enter(S);

// AccessResource()

obtain bank amount;

add in deposit;

update bank amount;

// AccessResource() end

exit(S); // exit
```

```
¦ATM2:
   enter(S);
   // AccessResource()
   obtain bank amount;
   add in deposit;
   update bank amount;
   // AccessResource() end
   exit(S); // exit
```

APPROACHES TO SOLVE MUTUAL EXCLUSION

• Single OS:

- If all processes are running in one OS on a machine (or VM), then
- Semaphores, mutexes, condition variables, monitors, etc.

Approaches to Solve Mutual Exclusion (2)

- Distributed system:
 - Processes communicating by passing messages

Need to guarantee 3 properties:

- Safety (essential) At most one process executes in CS (Critical Section) at any time
- Liveness (essential) Every request for a CS is granted eventually
- Ordering (desirable) Requests are granted in the order they were made

Processes Sharing an OS: Semaphores

- Semaphore == an integer that can only be accessed via two special functions
- Semaphore S=1; // Max number of allowed accessors

```
1. wait(S) (or P(S) or down(S)):
```

```
while(1) { // each execution of the while loop is \underline{atomic} if (S > 0) { S--; break; }
```

Each while loop execution and S++ are each atomic operations – supported via hardware instructions such as compare-and-swap, test-and-set, etc.

```
exit() 2. signal(S) (or V(S) or up(s)):
S++; // atomic
```

OUR BANK EXAMPLE USING SEMAPHORES

```
Semaphore S=1; // shared
ATM1:
   wait(S);
   // AccessResource()
   obtain bank amount;
   add in deposit;
   update bank amount;
  // AccessResource() end :
   signal(S); // exit
```

```
Semaphore S=1; // shared
¦ATM2:
   wait(S);
    // AccessResource()
    obtain bank amount;
   add in deposit;
    update bank amount;
   // AccessResource() end
   signal(S); // exit
```

NEXT

- In a distributed system, cannot share variables like semaphores
- So how do we support mutual exclusion in a distributed system?