



Chap. 8) Deadlocks

경희대학교 컴퓨터공학과

조진성

Deadlock Problems

■ Deadlock

- ✓ two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes

■ Let S and Q be two semaphores initialized to 1

P_0	P_1
<i>wait(S);</i>	<i>wait(Q);</i>
<i>wait(Q);</i>	<i>wait(S);</i>
\vdots	\vdots
<i>signal(S);</i>	<i>signal(Q);</i>
<i>signal(Q);</i>	<i>signal(S);</i>



Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously

- **Mutual exclusion:** only one process at a time can use a resource
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task
- **Circular wait:** there exists a set $\{P_0, P_1, \dots, P_n, P_0\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1 , P_1 is waiting for a resource that is held by P_2 , ..., P_{n-1} is waiting for a resource that is held by P_n , and P_n is waiting for a resource that is held by P_0



Handling Deadlocks

■ Deadlock prevention

- ✓ Restrain how requests are made
- ✓ Ensure that at least one necessary condition cannot hold

■ Deadlock avoidance

- ✓ Require additional information about how resources are to be requested
- ✓ Decide to approve or disapprove requests on the fly

■ Deadlock detection and recovery

- ✓ Allow the system to enter a deadlock state and then recover

■ Deadlock ignorance

- ✓ Just ignore the problem altogether!
- ✓ The Ostrich algorithm



Revisited: Dining Philosopher

■ A simple solution

```
Semaphore chopstick[N]; // initialized to 1
void philosopher (int i)
{
    while (1) {
        think ();
        wait (chopstick[i]);
        wait (chopstick[(i+1) % N]);
        eat ();
        signal (chopstick[i]);
        signal (chopstick[(i+1) % N]);
    }
}
```

⇒ Problem: causes deadlock



Revisited: Dining Philosopher

■ Deadlock-free version: starvation?

```
#define N      5
#define L(i)   ((i+N-1)%N)
#define R(i)   ((i+1)%N)
void philosopher (int i) {
    while (1) {
        think ();
        pickup (i);
        eat();
        putdown (i);
    }
}
void test (int i) {
    if (state[i]==HUNGRY &&
        state[L(i)]!=EATING &&
        state[R(i)]!=EATING) {
        state[i] = EATING;
        signal (s[i]);
    }
}
```

```
Semaphore mutex = 1;
Semaphore s[N];
int state[N];

void pickup (int i) {
    wait (mutex);
    state[i] = HUNGRY;
    test (i);
    signal (mutex);
    wait (s[i]);
}
void putdown (int i) {
    wait (mutex);
    state[i] = THINKING;
    test (L(i));
    test (R(i));
    signal (mutex);
}
```

