# Algorithm 3

- Think of this algorithm as using a referee who keeps track of whose "turn" it is
  - Anytime the two disagree about whose turn it is, they ask the referee, who keeps track of whose turn it is to have priority
  - This is called Peterson's algorithm (1981)
    - The original (but more complicated) solution to this problem is Dekker's algorithm (1965)
- For n processes, we can use Lamport's Bakery algorithm (1974)
  - When a thread tries to enter the critical section, it get assigned a number higher than anyone else's number
  - Thread with lowest number gets in
  - If two threads get the same number, the one with the lowest process id gets in

# Algorithm 3 (cont.)

```
■ Code:
t1 ( ) {
```

```
ti () {
    while (true) {
        t1_in_crit = true;
        turn = 2;
        while (t2_in_crit == true && turn != 1)
            ; /* do nothing */
            ... critical section of code ...
        t1_in_crit = false;
        ... other non-critical code ...
    }
}
t2 () {
    while (true) {
        similar...
    }
}
```

#### Semaphores — OS Support for Mutual Exclusion

- Semaphores were invented by Dijkstra in 1965, and can be thought of as a generalized locking mechanism
  - A semaphore supports two <u>atomic</u> operations, P / wait and V / signal
    - The semaphore initialized to 1
    - Before entering the critical section, a thread calls "P(semaphore)", or sometimes "wait(semaphore)"
    - After leaving the critical section, a thread calls "V(semaphore)", or sometimes "signal(semaphore)"
- Too much milk:

#### Thread A

milk->P( ); if (noMilk) buy milk; milk->V( );

# milk–>P( ); if (noMilk)

Thread B

if (noMilk) buy milk; milk–>V( );

# **Details of Semaphore Operation**

- Semaphore "s" is initially 1
- Before entering the critical section, a thread calls "P(s)" or "wait(s)"
  - wait (s):

2

Fall 1998, Lecture 11

- s = s 1
- if (s < 0)

block the thread that called wait(s) on a queue associated with semaphore s

- otherwise
   let the thread that called wait(s) continue into the critical section
- After leaving the critical section, a thread calls "V(s)" or "signal(s)"
  - signal (s):
    - ∎ s = s + 1
    - if (s  $\leq$  0), then

wake up one of the threads that called wait(s), and run it so that it can continue into the critical section

3

Fall 1998, Lecture 11

# **Semaphore Operation**

- Informal description:
  - Single igloo, containing a blackboard and a very large freezer
  - Wait thread enters the igloo, checks the blackboard, and decrements the value shown there
    - If new value is 0, thread goes on to the critical section
    - If new value is negative, thread crawls in the freezer and hibernates (making room for others to enter the igloo)
  - Signal thread enters igloo, checks blackboard, and increments the value there

Semaphore Operation & Values

signal (s):

s = s + 1

if  $(s \le 0)$ 

wake up & run one of

the waiting threads

Semaphores (simplified slightly):

block the thread

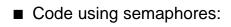
that called wait(s)

continue into CS

Semaphore values:

If new value is 0 or negative, there's a thread waiting in the freezer, so it thaws out a frozen thread, which then goes on to the critical section

### **Using Semaphores**



```
t1 () {
    while (true) {
        wait (s);
        ... critical section of code ...
        signal (s);
        ... other non-critical code ...
    }
}
t2() {
    while (true) {
        wait (s);
        ... critical section of code ...
        signal (s);
        ... other non-critical code ...
    }
}
```

Fall 1998, Lecture 11

#### Using Semaphores for Mutual Exclusion

Too much milk:

6

<u>Thread A</u> milk–>P(); if (noMilk) buy milk; milk–>V(); <u>Thread B</u> milk->P(); if (noMilk) buy milk; milk->V();

- "noMilk" is a semaphore initialized to 1
- Execution:

<u>After:</u>	<u>s</u>	<u>queue</u>	<u>A</u>	<u>B</u>
	1			
A: noM->P( );	0		in CS	
B: noM->P( );	-1	В	in CS	waiting
A: noM->V( );	0		finish	ready, in CS
B: noM->V( );	1			finish

blocked (note — there's also one in CS)
Binary semaphore has an initial value of 1

• Negative semaphore = number of threads

(additional) threads that can be allowed

Positive semaphore = number of

into the critical section

- Counting semaphore has an initial value greater than 1
  - Fall 1998, Lecture 11

8

Fall 1998, Lecture 11

5

wait (s):

s = s - 1

if (s < 0)

otherwise

#### The Coke Machine (Bounded-Buffer Producer-Consumer)

```
/* number of full slots (Cokes) in machine */
semaphore fullSlot = 0;
/* number of empty slots in machine */
semaphore emptySlot = 100;
/* only one person accesses machine at a time */
semaphore mutex = 1;
DeliveryPerson()
{
    emptySlot->P();
                             /* empty slot avail? */
    mutex->P();
                             /* exclusive access */
    put 1 Coke in machine
    mutex->V();
    fullSlot->V();
                             /* another full slot! */
}
ThirstyPerson()
{
    fullSlot->P();
                             /* full slot (Coke)? */
                             /* exclusive access */
    mutex->P();
    get 1 Coke from machine
    mutex->V();
    emptySlot->V();
                             /* another empty slot! */
}
9
                                             Fall 1998, Lecture 11
```