lecture-14
BDU

classmate
Date 20/02/18
Page

~~Semaphore~~ Semaphore

```
wait(s)                          signal(s)
{                                {
    while(s<=0);  <- busy wait       s++;
    s--;          -> out of      }
}                    while
```

when producer uses the buffer, that is its critical section.

Producer-consumer problem

```
do {
    // produce an item
    wait(empty);
    wait(mutex);
    // add item to buffer
    signal(mutex);
    ~~signal(full);~~ signal(full);
} while(1);
```

Producer process

```
do {
    wait(full);
    wait(mutex);
    //consumes an item from buffer
    signal(mutex);
    signal(empty);
} while(1);
```

Consumer process.

	mutex	empty	full
P1	1	n	0
	0	n-1	1
P2	1	n-2	2
	1	n-1	1
	0		0
	1	n	
C2	0		
	1		

Race condition: Inconsistency b/w processes while sharing the same resource.

Reader-writer problem

Reader process

```
do {
    wait(wrt);
    //writing is performed
    signal(wrt);
} while(1);
```

Reader's process

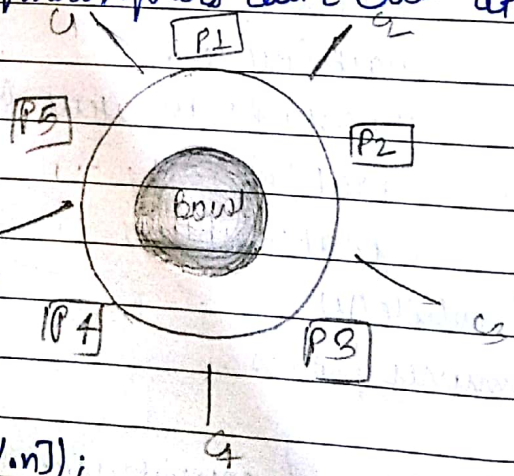
```
do {
    wait(mutex);
    readcount++;
    if (readcount == 1)
        wait(wrt);
    signal(mutex);
    // reading performed
    wait(mutex);
    readcount--;
    if (readcount == 0)
        signal(wrt);
    signal(mutex);
} while(1);
```

Semaphores

readcount	mutex	wrt
0	1	1 → wait(wrt) → 0
1	0	0 → signal(mutex) → 1
2	1	0
1	0	1 → signal(wrt)
0	1	0
1	0	1
0	1	0

Dining philosopher problem

- Can either eat or think at once.
- Each philosopher needs two chop-sticks to eat.
- So two adjacent philosophers can't eat at once.




```
do {
    wait(chopstick[i]);
    wait(chopstick[(i+1)%n]);
    // eat
    signal(chopstick[i]);
    signal(chopstick[(i+1)%n]);
    // think
} while(1);
```

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]
1	1	1	1	1	1
0	0				