

9.5 Philosophenproblem (6)

- Lösung 2: einer der Philosophen muss erst die andere Gabel aufnehmen

```
Semaphore [] forks= new Semaphore[5] { 1, 1, 1, 1, 1 };
```

```
Philosoph i, i ∈ [0,3]
while( 1 ) {
    ... /* think */

    forks[i].P();
    forks[(i+1)%5].P();

    ... /* eat */

    forks[i].V()
    forks[(i+1)%5].V()
}
```

```
Philosoph 4
while( 1 ) {
    ... /* think */

    forks[0].P();
    forks[4].P();

    ... /* eat */

    forks[0].V()
    forks[4].V()
}
```



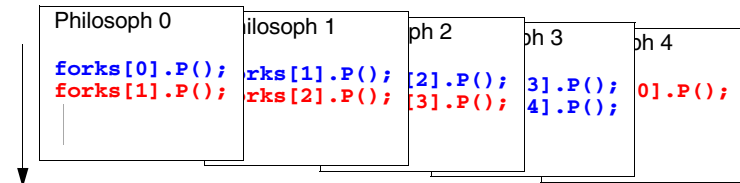
10 Monitore

- Einfaches Konzept zur Koordinierung (B. Hansen/Hoare 1975)
 - ◆ Ziel: Vermeidung von Koordinierungsfehlern durch falsche Platzierung von Semaphoren
- Monitor als programmiersprachliches Konstrukt mit
 - ◆ privaten Variablen
 - ◆ Prozeduren
 - laufen immer im gegenseitigen Ausschluss
 - ◆ Condition-Variables zum Blockieren
 - Aufruf von **wait(c)** führt zur Blockade an Condition-Variable **c**
 - Aufruf von **signal(c)** führt zur Deblockade eines Prozesses blockiert an Condition-Variable **c**
 - Blockieren und Deblockieren unter gegenseitigem Ausschluss
 - Freigabe der Sperre bei Blockieren
 - Belegen der Sperre vor Deblockierung



9.5 Philosophenproblem (7)

- ◆ Ablauf der asymmetrischen Lösung im ungünstigsten Fall



- ◆ System verklemmt sich nicht



10.1 Beispiel: Erzeuger-Verbraucher

```
monitor
{
    char buffer[N];
    int inslot= 0, outslot= 0, count= 0;
    condition notFull, notEmpty;

    void put( char c )
    {
        while( count == N )
            wait( notFull );

        buffer[inslot]= c;
        inslot= (inslot+1)%N;
        count++;
        signal( notEmpty );
    }

    char get( void )
    {
        char c;

        while( count == 0 )
            wait( notEmpty );

        c= buffer[outslot];
        outslot= (outslot+1)%N;
        count--;
        signal( notFull );
        return c;
    }
}
```



10.2 Beispiel: Java

■ Threaderzeugung

```
class MyThread extends Thread
{
    public void run()
    {
        ... /* thread code */
    }
}
```

```
...
Thread t= new MyThread();
...
```

- ◆ weitere Möglichkeiten ohne Vererbung (siehe Java-Dokumentation)



10.2 Beispiel: Java (3)

- Jedes Objekt stellt Monitor dar
 - ◆ nur für **synchronized** Methoden
 - ◆ expliziter Eintritt in den Monitor (lock) durch **synchronized(obj) { ... }**
- Nur eine implizite Condition-Variable
 - ◆ **wait** und **notify** statt **wait** und **signal**
 - ◆ **notifyAll** deblockiert alle am Objekt (Monitor) blockierten Threads
- ★ Näheres: siehe Java-Dokumentation



10.2 Beispiel: Java (2)

```
class buffer
{
    char buffer[]= new char[N];
    int inslot= 0, outslot= 0, count= 0;

    synchronized public void put( char c )
    {
        while( count == N )
            try { wait(); }
            catch( ... ) {}

        buffer[inslot]= c;
        inslot=(inslot+1)%N;
        count++;
        notifyAll();
    }

    synchronized public char get( void )
    {
        char c;

        while( count == 0 )
            try { wait(); }
            catch( ... ) {}

        c= buffer[outslot];
        outslot= (outslot+1)%N;
        count--;
        notifyAll();
        return c;
    }
}
```

