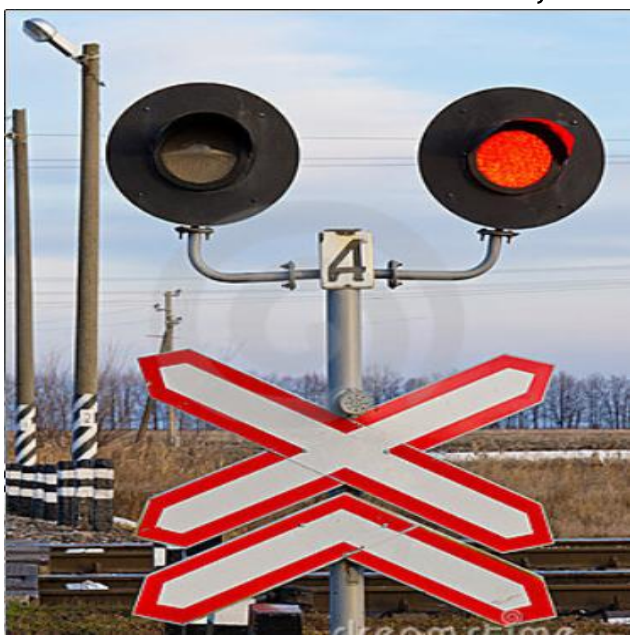
# Bendrų kintamųjų apsauga ir sinchronizavimas naudojant semaforus

# Semaforas

- Sema + phoros = ženklas + nešantis (gr.) – stacionarus signalinis geležinkelio įtaisas, kuris leidžia arba draudžia važiuoti traukiniams;




- programavime: 1968 m.,
autorius E.W.Dijkstra (1930-2002).

# Galimi veiksmai (Dijkstra)

- `s.P()` operacija:
  jei `s` reikšmė `= 0`, blokuoti procesą ties `s`, priešingu atveju – sumažinti `s` reikšmę;
- `s.V()` operacija:
  jei yra blokuotų procesų ties `s`, išblokuoti vieną iš jų, priešingu atveju – padidinti `s` reikšmę;
- `s.init(a)` operacija:
  semaforui `s` suteikti pradinę reikšmę `a`.

# Semaforų savybės

- blocked waiting laukimas:
  - FIFO: P – į eilės galą, V – eilės pr.,
  - RANDOM: P – atsitikt., V – atsitikt.;
- daugumoje kalbų realizacijų: $s \geq 0$;
- $s = s0 + \#V - \#P$;

# Galimi tipai

- priklausomai nuo reikšmių rinkinio:
  - bendrieji (*general, counting semaphore*);
  - *dvejetainiai (binary semaphore*);
- priklausomai nuo veikimo:
  - "šališki"(*unfair*)
  - "beveik bešališki"(*weakly fair*)
  - "bešališki"(*fair*)

# Kritinės sekcijos apsauga

```
class G extends Thread {
  void run() {
    while (true) {
      NKS1;
      veiksmai prieš KS;
      KS;
      veiksmai už KS;
      NKS2;
    }
  }
}
```

# KS apsauga: semaforas (1)

- semaforo reikšmė kiekvienu laiko momentu:
  $s = s0 + \#V - \#P$;
- tarpusavio išskyrimo sąlyga:
  $\#KS + s = 1; s0 = 1$.

# KS apsauga: semaforas (2)

```
class G extends Thread {
  void run() {
    while (true) {
      NKS1;
      s.P();   // veiksmai prieš KS;
      KS;
      s.V();   // veiksmai už KS;
      NKS2;
    }
  }
}
```

**s.init(???)**

# Sinchronizavimo sąlygos

- vykdyti veiksmus tada, kai tenkinama tam tikra sąlyga (sąlygos);
- nevykdyti veiksmų, kol nebus tenkinama tam tikra sąlyga (sąlygos).

# Sąlyginė sinchronizacija: semaforas

```
// thread                          | // thread
class Siuntėjas {                  | class Gavėjas {
  int m;                           |   int d;
  void run() {                     |   void run() {
    Formuoti(m);                   |     ??? // while (!BK2.t);
    ??? // while (BK2.t);          |     d=BK2.v;
    BK2.v=m;                       |     ??? // BK2.t=false;
    ??? // BK2.t=true;             |     Naudoti(d);
} }                                | } }
class BK2 {
  int v; ??? // boolean t = false;
  void main() {
    ...
} }
```

# Java semaforai

# Java Semaphore klasė (1)

- Java versija: nuo 1.5.0 (**5.0**)
- **import java.util.concurrent.Semaphore;**
- `Semaphore s = new Semaphore(p);`
  Creates **s** with the given number of permits (int **p**) and nonfair fairness setting.
- `Semaphore s = new Semaphore(p, f);`
  Creates **s** with the given number of permits (int **p**) and the given fairness (boolean **f**) setting.

# Java Semaphore klasė (2)

- `s.acquire();`
  Acquires a permit from **s**, blocking until one is available, or the thread is interrupted.
- `s.acquire(p);`
  Acquires the given number of permits (int **p**) from **s**, blocking until all are available, or the thread is interrupted.
- `s.release();`
  Releases a permit, returning it to the **s**.

# Java Semaphore klasė (3)

- `s.release(p);`
  Releases the given number of permits (int **p**), returning them to the **s**.
- `s.isFair();`
  Returns **true**, if **s** has fairness set **true**.
- `s.availablePermits();`
  Returns the current number (int) of permits available in **s**.

# KS apsauga: Java semaforas

```
class G extends Thread {
  void run() {
    while (true) {
      NKS1;
      s.acquire(); // veiksmai prieš KS;
      KS;
      s.release(); // veiksmai už KS;
      NKS2;
    }
  }
}
```

```
Semaphore s = new Semaphore(1, true);
Semaphore s = new Semaphore(1, false);
```

# Sąl. sinchr.: Java semaforas (1)

```
// thread
class Siuntėjas {
  int m;
  void run() {
    Formuoti(m);
    BK2.galimaRašyti.acquire(); // while (BK2.t);
    BK2.v=m;
    BK2.galimaSkaityti.release(); // BK2.t=true;
  }
}
```

# Sąl. sinchr.: Java semaforas (2)

```
// thread
class Gavėjas {
  int d;
  void run() {
    BK2.galimaSkaityti.acquire(); // while (!BK2.t);
    d=BK2.v;
    BK2.galimaRašyti.release(); // BK2.t=false;
    Naudoti(d);
  }
}
```

# Sąl. sinchr.: Java semaforas (3)

```
class BK2 {
 int v;
 Semaphore galimaRašyti = new Semaphore(1,true);
 Semaphore galimaSkaityti = new Semaphore(0,true);
      // boolean t = false;
 void main() {
   ...
 }
}
```

Ar galima padaryti su vienu semaforu?

# Gamintojas – Vartotojas (1)

```
class RibotasBuferis{ ...  }
class Gamintojas implements Runnable { ...  }
class Vartotojas implements Runnable { ...  }

public class Gamykla {
  public static RibotasBuferis buffer =
                                new RibotasBuferis();
  public static void main(String args []) {
    Thread gam = new Thread(new Gamintojas(10));
    Thread vart = new Thread(new Vartotojas(10));
    vart.start();
    gam.start();
  }
}
```

# G–V. RibotasBuferis (2a)

```
class RibotasBuferis {
  private static final int buferioDydis = ..;
  private int[] buferis;
  private int į, iš;
  private Semaphore ksApsauga;
  private Semaphore laisva;
  private Semaphore užimta;
  public RibotasBuferis() { ...  }
  public void įterpti(int daiktas) { ...  }
  public int paimti() { ...  }
}
```

# G–V. RibotasBuferis (2b)

```
public RibotasBuferis() {
  į = 0; iš = 0;
  buferis = new int[buferioDydis];
  ksApsauga = new Semaphore(1);
  laisva = new Semaphore(buferioDydis);
  užimta = new Semaphore(0);
}
```

# G–V. RibotasBuferis (2c)

```
public void įterpti(int daiktas) {
  try {
    laisva.acquire();
    ksApsauga.acquire();
    buferis[į] = daiktas;
    į = (į + 1) % buferioDydis;
    ksApsauga.release();
    užimta.release();
  } catch (InterruptedException e) { }
}
```

# G–V. RibotasBuferis (2d)

```
public int paimti() {
  int daiktas = 0;
  try {
    užimta.acquire();
    ksApsauga.acquire();
    daiktas = buferis[iš];
    iš = (iš + 1) % buferioDydis;
    ksApsauga.release();
    laisva.release();
  } catch (InterruptedException e) { }
  return daiktas;
}
```

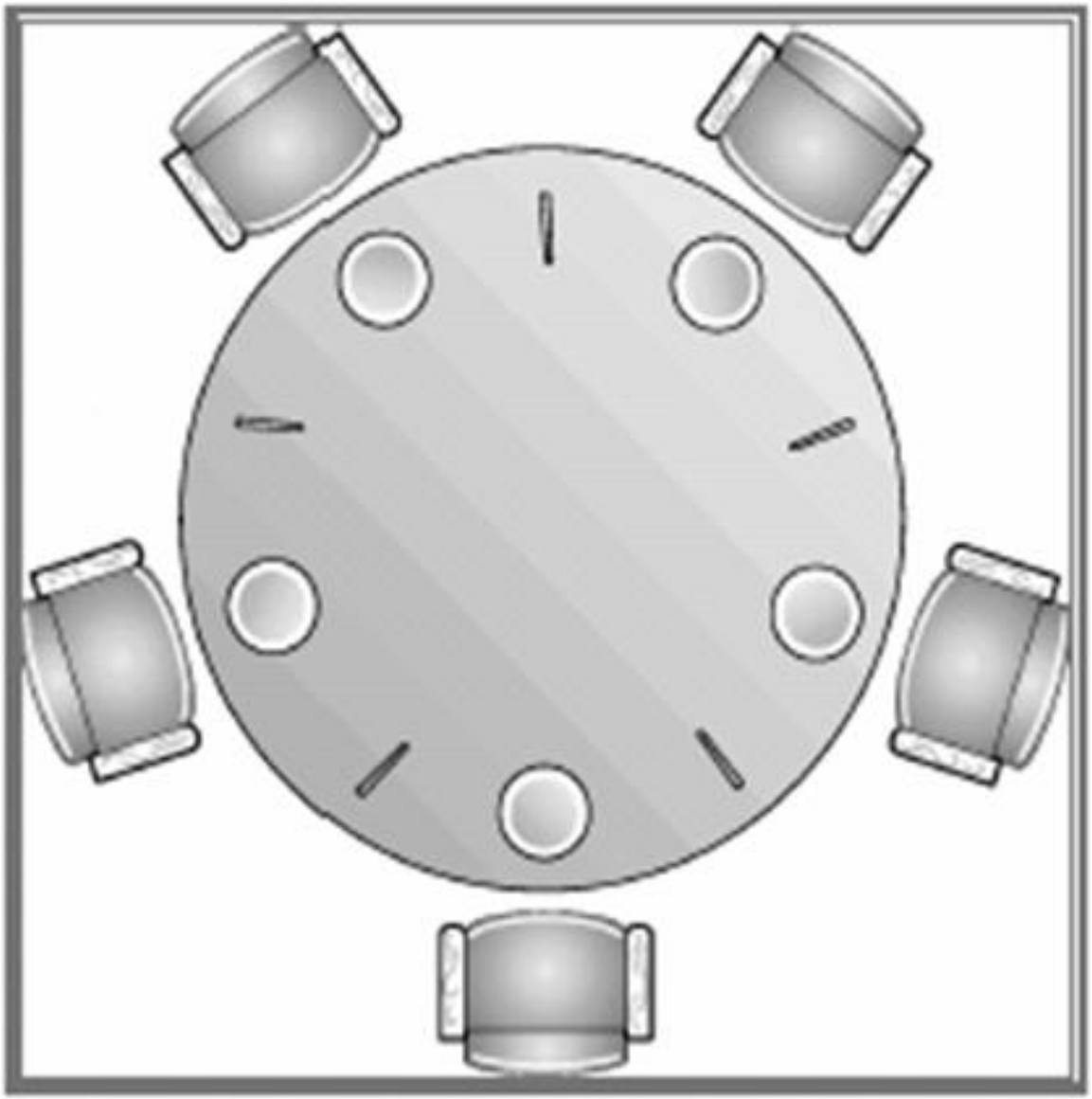
# G–V. Gamintojas

```
class Gamintojas implements Runnable {
  private int kiekGaminti;
  private int gaminys;
  public Gamintojas(int kiekGaminti) {
    this.kiekGaminti = kiekGaminti;
  }
  public void run() {
    for (int i = 0; i<kiekGaminti; i++) {
    // while (true)
      Gaminti(gaminys);
      Gamykla.buferis.įterpti(gaminys);
    }
  }
}
```

# G–V. Vartotojas

```
class Vartotojas implements Runnable {
  private int kiekVartoti;
  private int gaminys;
  public Vartotojas(int kiekVartoti) {
    this.kiekVartoti = kiekVartoti;
  }
  public void run() {
    for (int i = 0; i<kiekVartoti; i++) {
    // while (true)
      gaminys = Gamykla.buferis.paimti();
      Vartoti(gaminys);
    }
  }
}
```

# Pietaujantieji filosofai (1)



# Pietaujantieji filosofai (2)

```
class Filosofas extends Thread { ... }
public class Valgykla {
  int filosN = 5, gyvT = 500;
  Semaphore[] šakutė = new Semaphore[filosN];
  Filosofas[] filos= new Filosofas[filosN];
  public static void main(String args []) {
    for (int i=0; i<filosN; i++)
      filos[i]=new Filosofas(i, gyvT);
    for (int i=0; i<filosN;i++)
      šakutė[i] = new Semaphore(1);
    for (int i=0; i<filosN;i++) filos[i].start();
  }
}
```

# Pietaujantieji filosofai (3)

```
class Filosofas extends Thread {
  private int numeris, gyvTrukmė;
  public Filosofas(int numeris, int gyvTrukmė) {
    this.numeris = numeris;
    this.gyvTrukmė = gyvTrukmė;
  }
  private void valgyti() { ... }
  private void mastyti() { ... }
  public void run() { ... }
}
```

# Pietaujantieji filosofai (4)

```
public void run() {
  for (int i=0;i<gyvTrukmė;i++) { // while (true)
    try {
      Valgykla.šakutė[numeris].acquire();
      Valgykla.šakutė[(numeris+1)%
              Valgykla.filosN].acquire();
      valgyti();
      Valgykla.šakutė[numeris].release();
      Valgykla.šakutė[(numeris+1)%
              Valgykla.filosN].release();
      mastyti();
    } catch (InterruptedException e) {}
  }
}
```

# Pietaujantieji filosofai (5)

- Kokiu atveju galima aklavietės situacija?
- Kaip pašalinti aklavietės galimybę?

- https://www.youtube.com/watch?v=H33eWKOiUJE
- http://users.erols.com/ziring/diningAppletDemo.html
- http://www.ssw.uni-linz.ac.at/General/Staff/DB/Private/DiningPhilosophers/

# Bendrieji ir dvejetainiai semaforai (1)

- Kritinėms sekcijoms saugoti reikalingi dvejetainiai semaforai.
- Sąlyginei sinchronizacijai reikalingi bendrieji semaforai.

- Ar dvejetainį semaforą galima modeliuoti bendruoju semaforu?
- Ar bendrąjį semaforą galima modeliuoti dvejetainiu semaforu?

# Bendrieji ir dvejetainiai semaforai (2)

```
class BendrasisSemaforas {
  private int n;
  private Semaphore s1, s2;   // dvejetainiai
  public BendrasisSemaforas (int n) {...}
  public void acquire(){...}
  public void release(){...}
}
```

# Bendrieji ir dvejetainiai semaforai (3)

```
public BendrasisSemaforas (int n) {
  this.n = n;
  if (n == 0) s1 = new Semaphore(0);
  else        s1 = new Semaphore(1);
  s2 = new Semaphore(1);
}
```

# Bendrieji ir dvejetainiai semaforai (4)

```
public void acquire() {
   s1.acquire();
   s2.acquire();
   n--;
   if (n > 0) s1.release();
   s2.release();
}
```

# Bendrieji ir dvejetainiai semaforai (5)

```
public void release() {
   s2.acquire();
   n++;
   if (n > 1) s1.release();
   s2.release();
}
```

# Semaforai be *Semaphore* (1)

```
class Semaphore {
 private int count;
 public Semaphore (int count) {
   this count = count;
 }
 public synchronized void acquire(){...}
 public synchronized void release(){...}
}
```

# Semaforai be *Semaphore* (2)

```
public synchronized void acquire(){
  while (count == 0) {
    try { wait(); }
    catch (InterruptedException e) { }
  }
  count--;
}
public synchronized void release() {
  count++;
  notify();
}
```

# Semaforas ar monitorius: kurį panaudoti?

- galima užrašyti monitorių, modeliuojantį tiek bendrojo, tiek dvejetainio semaforo darbą;
- panaudojant semaforus galima modeliuoti bet kurio monitoriaus darbą.

# Klausimai pakartojimui

1. Kokie yra semaforų tipai?
2. Kaip apibrėžiama teisingo tarpusavio išskyrimo sąlyga?
3. Kiek reikia semaforų vienai kritinei sekcijai saugoti ar sąlyginei sinchronizacijai realizuoti?
4. Kokia turi būti semaforo, saugančio kritinę sekciją, pradinė reikšmė?
5. Apibūdinkite Java semaforus (tipas, inicializavimas, keitimas).
6. Kuo skiriasi bendrieji ir dvejetainiai semaforai?
7. Nurodykite pagrindinius Java semaforų ir OpenMP užraktų panašumus bei skirtumus.