Lösningar, EDAF85 Realtidssystem

2024–10–25, 14.00–19.00

1. Deadline Monotonic Scheduling används i system där schemaläggningen sker strikt enligt fasta prioriteter för trådarna. Den tråd som har kortast deadline får högst prioritet och sedan sätts lägre prioriteter i takt med längre deadline. Om vi i stället använder Earliest Deadline First har trådarna ingen fast prioritet utan det är hela tiden den tråd som har *närmast till sin nästa deadline* som får köra. En tråd med lång deadline kan alltså prioriteras framför en som har kort deadline om den råkar ha sin kommande deadline närmare i framtiden.

2. a) B_A: $0\mu s$ – varken direkt eller indirekt blockering
   B_B: $4 + 5 = 9\mu s$ – direkt blockering i M1 och M2
   B_C: $max(3, 5) = 5\mu s$ – antingen direkt blockering i M1 *eller* indirekt blockering i M2
   B_D: $0\mu s$ – inga lägre prioriterade trådar

   b) B_A: $0\mu s$ – varken direkt eller indirekt blockering
   B_B: $5\mu s$ – direkt blockering i M pga `d2()`
   B_C: $5\mu s$ – direkt blockering i M pga `d2()`
   B_D: $0\mu s$ – inga lägre prioriterade trådar

3. Prioritetsordning: A, D, B, C

$$R_A^0 = 1{,}5 + 0{,} = 2{,}1$$
$$R_A^1 = 1{,}5 + 0{,} = 2{,}1$$

$$R_D^0 = 2 + 0{,}4 = 2{,}4$$
$$R_D^1 = 2 + 0{,}4 + \left\lceil \frac{2{,}4}{5} \right\rceil \cdot 1{,}5 = 3{,}9$$
$$R_D^2 = 1 + 0{,}4 + \left\lceil \frac{3{,}4}{5} \right\rceil \cdot 1{,}5 = 3{,}9$$

$$R_B^0 = 4 + 0{,}3 = 4{,}3$$
$$R_B^1 = 4 + 0{,}3 + \left\lceil \frac{4{,}3}{5} \right\rceil \cdot 1{,}5 + \left\lceil \frac{4{,}3}{10} \right\rceil \cdot 2 = 7{,}8$$
$$R_B^2 = 4 + 0{,}3 + \left\lceil \frac{7{,}8}{5} \right\rceil \cdot 1{,}5 + \left\lceil \frac{7{,}8}{10} \right\rceil \cdot 2 = 9{,}3$$
$$R_B^3 = 4 + 0{,}3 + \left\lceil \frac{9{,}3}{5} \right\rceil \cdot 1{,}5 + \left\lceil \frac{9{,}3}{10} \right\rceil \cdot 2 = 9{,}3$$

$$R_C^0 = 2 + 0 = 1$$
$$R_C^1 = 2 + 0 + \left\lceil \frac{2}{5} \right\rceil \cdot 1{,}5 + \left\lceil \frac{2}{10} \right\rceil \cdot 2 + \left\lceil \frac{2}{12} \right\rceil \cdot 4 = 9{,}5$$
$$R_C^2 = 2 + 0 + \left\lceil \frac{9{,}5}{5} \right\rceil \cdot 1{,}5 + \left\lceil \frac{9{,}5}{10} \right\rceil \cdot 2 + \left\lceil \frac{9{,}5}{12} \right\rceil \cdot 4 = 11$$
$$R_C^3 = 2 + 0 + \left\lceil \frac{11}{5} \right\rceil \cdot 1{,}5 + \left\lceil \frac{11}{10} \right\rceil \cdot 2 + \left\lceil \frac{11}{12} \right\rceil \cdot 4 = 14{,}5$$
$$R_C^4 = 2 + 0 + \left\lceil \frac{14{,}5}{5} \right\rceil \cdot 1{,}5 + \left\lceil \frac{14{,}5}{10} \right\rceil \cdot 2 + \left\lceil \frac{14{,}5}{12} \right\rceil \cdot 4 = 18{,}5$$
$$R_C^5 = 2 + 0 + \left\lceil \frac{18{,}5}{5} \right\rceil \cdot 1{,}5 + \left\lceil \frac{18{,}5}{10} \right\rceil \cdot 2 + \left\lceil \frac{18{,}5}{12} \right\rceil \cdot 4 = 20$$
$$R_C^6 = 2 + 0 + \left\lceil \frac{20}{5} \right\rceil \cdot 1{,}5 + \left\lceil \frac{20}{10} \right\rceil \cdot 2 + \left\lceil \frac{20}{12} \right\rceil \cdot 4 = 20$$

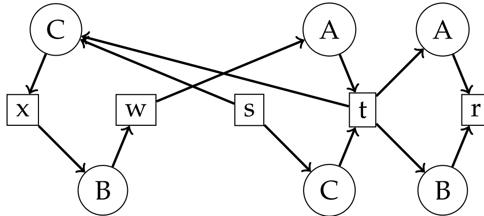
Svar: $R_A = 2{,}1ms$, $R_B = 9{,}3ms$, $R_C = 20ms$ och $R_D = 3{,}9ms$

4. a) Inget skrivs ut eftersom programemt blockerar i det första `mutex.acquire()`. Det händer eftersom semaforens värde är 0 från början.

   b) ″1″ skrivs ut. Eftersom semaforens startvärde var 1 kan programmet passera det första `mutex.acquire()`, men fastnar i det andra i början av `print2()` eftersom semaforens värde då raknats ner till 0.

   c) ″1″ och ″2″ skrivs ut. Eftersom en och samma tråd kan låsa ett `ReentrantLock` flera gången blockeras programmet inte i början av `print2()`.

5. a)



   b) A: 14
   B: 37
   C: 52

   c) Flera lösningar finns, men ett enkelt sätt är att flytta rad 52 (`x.acquire()`) till direkt efter rad 49. Då fås en strikt allokeringsordning för semaforerna (x–w–s–t–r eller möjligen x–s–w–t–r) som garanterar att inga loopar i resursallokeringsgrafen uppstår.

6.
```
public class Barrier {

    // number of threads expected in each cycle
    private final int n;

    // current cycle
    private int cycle = 0;

    // number of threads arrived in the current cycle
    private int arrived = 0;

    public Barrier(int n) {
        if (n < 2) {
            throw new IllegalArgumentException();
        }

        this.n = n;
    }

    public synchronized int await() throws InterruptedException {
        int c = cycle;
        int k = arrived;

        arrived++;
        notifyAll();

        // wait until all threads in the current cycle
        // have arrived
        while (c == cycle && arrived < n) {
            wait();
        }

        // the first thread that leaves the barrier resets
        // it for the next cycle
        if (c == cycle) {
            cycle++;
            arrived = 0;
        }

        return k;
    }
}
```

Alternativ implementation av `await()`*:*

```
    public synchronized int await() throws InterruptedException {
        int c = cycle;
        int k = arrived;
        arrived++;
        if (arrived<n) {
            while (c==cycle) {
                wait();
            }
        } else {
            cycle++;
            arrived = 0;
            notifyAll();
        }
        return k;
    }
```

7.
```
public class Barrier {

    private final int n;
    private int arrived = 0;
    private Semaphore holdSem = new Semaphore(0);
    private Lock mutex = new ReentrantLock();

    public Barrier(int n) {
        if (n < 2) {
            // a barrier for fewer than
            // 2 threads makes no sense
            throw new IllegalArgumentException();
        }
        this.n = n;
    }

    public int await() throws InterruptedException {
        mutex.lock();
        Semaphore mySem = holdSem;
        int k = arrived;
        arrived++;
        if (arrived==n) {
            holdSem.release(n);  // Signal all threads
                                 // in the current cycle

            // Alternative code for signaling:
            // for(int i=0;i<n;i++) {
            //     holdSem.release();
            // }

            holdSem = new Semaphore(0); // Reset semaphore and
            arrived = 0;                // count for next cycle
        }
        mutex.unlock();
        mySem.acquire();  // Ensure we have permission to proceed
        return k;
    }
}
```