Tentamen
EDAF85 – Realtidssystem

2024-08-29, 14.00–19.00

Det är tillåtet att använda Java snabbreferens och miniräknare. Det går bra att använda både engelska och svenska i svaren.

Den här tentamen består av två delar: *teori*, fråga 1-5 (15 poäng), och en *programmeringsuppgift*, fråga 6 (15 poäng). För godkänd (betyg 3) krävs preliminärt sammanlagt hälften av alla 30 möjliga poäng samt en tredjedel av poängen (5) på varje del för sig.

Formelsamling

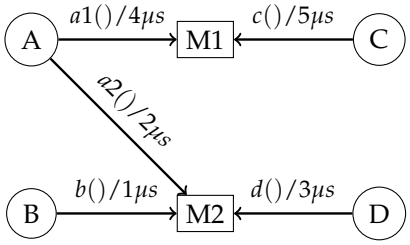
$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

$$\begin{gathered} 2 \cdot (2^{1/2} - 1) \approx 0,828427 \\ 3 \cdot (2^{1/3} - 1) \approx 0,779763 \\ 4 \cdot (2^{1/4} - 1) \approx 0,756828 \\ 5 \cdot (2^{1/5} - 1) \approx 0,743492 \\ \ldots \\ \lim_{n \to \infty} n(2^{1/n} - 1) \approx 0,693147 \end{gathered}$$

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

1. I Java finns olika sorters semaforer, bland annat `ReentrantLock` och `Semaphore`. Förklara med några få meningar vad den principiella skillnaden mellan dessa två typer av semaforer är. *(1p)*

2. Ett realtidssystem med dynamisk prioritetsbaserad schemaläggning och prioritetsarv (basic inheritance protocol) består av fyra periodiska trådar (A, B, C och D). Trådarna A, B, C och D kommunicerar med varandra genom att anropa monitoroperationerna `a1()`, `a2()`, `b()`, `c()`, och `d()` i monitorerna M1 och M2 och med maximala exekveringstider (mikrosekunder) enligt nedanstående figur. Monitormetoderna anropas en i taget (inga nästlade anrop). A har högst prioritet, B näst högst, C näst lägst och D har lägst prioritet. Vi förutsätter att trådarna exekverar på en dator med en CPU-kärna.



Ange för var och en av de fyra trådarna (A, B, C och D) den maximala tid tråden kan bli blockerad av lägre prioriterade trådar under en och samma körning (dvs deras blockeringsfaktorer, B_i). *(3p)*

3. Ett realtidssystem består av fyra periodiska trådar med maximal exekveringstid (C), periodtid (T), blockeringsfaktor (B) och deadline (D) enligt nedanstående tabell. Vi antar att direkt prioritetsarv tillämpas. Trådarna schemaläggs enligt principen deadline monotonic scheduling (DMS) och körs på en dator med en CPU-kärna.

	C (ms)	T (ms)	B (ms)	D (ms)
A	1	5	0,3	2
B	1	20	0	10
C	2	8	0,4	4
D	3	12	0,1	8

a) Ange vad systemets CPU-utnyttjande (*eng. CPU utilization*) är. *(1p)*

b) Ange för varje tråd vad dess värstafallssvarstid (R) blir[1]. *(4p)*

c) Är systemet schemaläggningsbart? Du måste svara med en korrekt motivering for att få poäng. Det räcker *inte* att bara svara ja eller nej. *(1p)*

[1] Uppgiftens tabell innehöll vid tentamenstillfället ett tryckfel som gjorde att den beskrev ett förhållande mellan trådarna som inte är möjligt i verkligheten (den lägst prioriterade tråden hade en blockeringsfaktor>0) och som upptäcktes först vid rättningen. I denna version är tryckfelet rättat. Uppgiften rättades generöst med hänsyn till felet.

4. Betrakta följande javaprogram. Metoderna `useRS()`, `useRV()`, `useS()`, `useST()`, `useT()`, `useTR()` och `useTV()` visas ej, men utför inga operationer på låsen. Metodnamnet anger vilka lås som förutsätts vara låsta när metoden exekveras. Exempelvis förutsätter en metod med namn `useXY()` att lås X och Y är låsta. Tyvärr hamnar programmet ibland i dödläge (deadlock).

```
1  public static void main(String[] args) {
2
3      Lock r = new ReentrantLock();
4      Lock s = new ReentrantLock();
5      Lock t = new ReentrantLock();
6      Lock v = new ReentrantLock();
7
8      new Thread(() -> {
9          while (true) {
10             r.lock();
11             s.lock();
12             useRS();
13             s.unlock();
14             v.lock();
15             useRV();
16             v.unlock();
17             r.unlock();
18         }
19     }).start();
20
21     new Thread(() -> {
22         while (true) {
23             s.lock();
24             useS();
25             t.lock();
26             useST();
27             t.unlock();
28             s.unlock();
29         }
30     }).start();
31
32     while (true) {
33         t.lock();
34         useT();
35         r.lock();
36         useTR();
37         r.unlock();
38         v.lock();
39         useTV();
40         v.unlock();
41         t.unlock();
42     }
43 }
```

a) Rita en resursallokeringsgraf för programmet. *(1,5p)*

b) Ange för var och en av trådarna vilken rad den befinner sig på när dödläge uppstår (vilken rad den försöker exekvera när den blockerar). *(1,5p)*

c) Föreslå en ändring av programmet som gör att dödläge inte kan uppstå, men samtidigt gör att inga onödiga lås är tagna när en `useXY()`-metod exekveras. Exempelvis ska endast låsen `r` och `t` vara tagna när `useRV()` körs. *(1p)*

5. Nedanstående javaprogram startar två trådar, väntar på att de båda ska exekvera färdigt och skriver sedan ut värdet av variabeln `x`.

```
public class ThreadTest {
    private static volatile int x = 0;

    public static void main(String[] args) throws InterruptedException {

        Thread t1 = new Thread(() -> {
            int s = x + 1;
            x = s;
        });

        x = 5;

        Thread t2 = new Thread(() -> {
            int t = x + 2;
            x = t;
        });

        // run both threads concurrently
        t1.start();
        t2.start();

        // wait for both threads to terminate
        t1.join();
        t2.join();

        System.out.println(x);
    }
}
```

Ange alla möjliga värden som kan komma att skrivas ut om vi kör programmet!

Ledning:

- Ett anrop av `start()` på en tråd garanterar att anropet av `start()` avslutas innan någon av instruktionerna i tråden får effekt.
- Ett anrop av `join()` på en tråd garanterar att allt som tråden gör får effekt innan `join()` returnerar.
- Kodordet `volatile` innebär här att om man tilldelar variabeln `x` ett nytt värde så garanteras det att det nya värdet är det som används vid framtida användningar av `x` – oavsett i vilken tråd det sker.

(2p)

6. Videomöte

I denna uppgift ska du vidareutveckla den del av ett system för on-line-undervisning via video som hanterar hopkoppling av studenter med handledare. En student ansluter till en server och (möjligen) efter en viss väntan får studenten träffa en handledare i ett privat möte.

Nuvarande design

Vår nuvarande design klarar bara att hantera ett enda videomöte i taget, dvs endast en student och handledare i ett möte. Du ser koden som hanterar detta i klassen `VideoMeetingApplication` nedan.

Metoden `awaitStudentConnected()` i klassen `WebServer` blockerar tills en student finns tillgänglig (har anslutit) och returnerar då ett objekt av typen `Request`. Själva mötet hanteras sedan av klassen `VideoMeeting` som är förberedd för att kunna hantera upp till fem samtidiga möten, men för tillfället använder systemet inte mer än en möteskanal åt gången.

```
public class Request {

    /** @return  the name of the connected student */
    public String getStudentName()                  { ... }

    /**
     * @return  the course code of the course the student
     *          student needs help with (e.g., "EDAF85")
     */
    public String getCourse()                       { ... }
}

public class VideoMeeting {
    /** Create a video meeting for instructor n (0 <= n < 5). */
    public VideoMeeting(int n)                          { ... }

    /**
     * Accepts the given student into the meeting. This method does not return
     * until the student has left the meeting (and the instructor is free).
     */
    public void acceptStudent(String studentName)       { ... }
}

public class WebServer {
    /**
     * Create a web server, allowing for students to connect.
     * Only one web server can exist in a single application.
     */
    public WebServer()                                  { ... }

    /**
     * Blocks until a student connects to get help with a particular
     * course, then returns a Request for that student and that course.
     *
     * If multiple students connect at the same time, their requests
     * are enqueued and returned, one by one and in the order they
     * connected, by this method.
     *
     * NOTE: this method must not be called by multiple threads.
     */
    public Request awaitStudentConnected()
        throws InterruptedException                     { ... }
}
```

```
public class VideoMeetingApplication {

    public static void main(String[] args) {
        WebServer server = new WebServer();
        VideoMeeting meeting = new VideoMeeting(0);

        new Thread(() -> {
            try {
                while (true) {
                    Request req = server.awaitStudentConnected();
                    meeting.acceptStudent(req.getStudentName());
                }
            } catch (InterruptedException unexpected) {
                throw new Error(unexpected);
            }
        }).start();
    }
}
```

Nödvändiga ändringar

Den nuvarande lösningen fungerar, men bara med en handledare. Studenter kan därför behöva vänta länge på sin tur. Dessutom läser studenterna olika kurser, vilket gör att handledaren måste vara bekant med alla möjliga kurser. Detta är opraktiskt.

Systemet behöver ändras på följande vis:

- Fem instruktörer ska kunna hjälpa studenterna samtidigt. Varje instruktör har sitt eget videomöte (representerat av separata instanser av `VideoMeeting`. Videomötesobjekten skapas genom anropet `new VideoMeeting(n)` där $0 \leq n < 5$ (motsvarande de fem handledarna). På så sätt får varje handledare sitt eget möte.
- Handledare ska bara acceptera studenter som hör till kurser de har kompetens inom. Alla handledare kan inte handleda alla kurser. Handledarna har olika kompetensområden, vilket beskrivs av listan `INSTRUCTOR_COMPETENCE`. Se den nedan givna uppdaterade versionen av `VideoMeetingApplication`.

  `INSTRUCTOR_COMPETENCE` är en lista av set. Varje listelement anger vilka kurser en viss handledare är kvalificerad att handleda. `INSTRUCTOR_COMPETENCE.get(0)` är ett set som anger till exempel vilka kurser handledare 0 kan handleda (EDAA45/A80/A85/A90/A30/F85). På samma sätt anger `INSTRUCTOR_COMPETENCE.get(2)` vilka kurser handledare 2 kan handleda (EDAA45/A30/F85).
- *Viktigt!* Notera att klassen `WebServer` inte är trådsäker; endast en tråd kan anropa metoden `awaitStudentConnected()` åt gången.

Din uppgift

Komplettera den nedan påbörjade nya versionen av systemet så att det fungerar med fem samtidiga handledare med varierande kompetens enligt beskrivningen ovan.

Instruktioner och tips:

- Använd endast ett `WebServer`-objekt i din lösning och låt endast en tråd använda detta (eftersom det inte är trådsäkert).
- Skapa ett separat `VideoMeeting`-objekt för varje handledare.
- Så fort en handledare är ledig, och det finns en väntande student som handledaren är kompetent att handleda, ska handledaren acceptera studenten till sitt möte. Studenter som väntat länge ska prioriteras före de som kommit senare.

- Inför nya trådar efter behov. Tänk på att hanteringen av anslutande studenter och handledarnas enskilda aktiviteter måste hanteras parallellt (metoderna `awaitStudentConnected()` samt `acceptStudent()` är båda blockerande).
- Använd en monitor för signalering och för att synkronisera trådarna. En tom monitorklass, `Monitor`, har skapats för detta. Välj en lämplig datastruktur, som t.ex. en `LinkedList`, för att hålla reda på kön av väntande studenter. Inför nödvändiga monitoroperationer för att hantera denna kö.
- För att se om ett set innehåller ett visst värde kan du använda metoden `contains()`.
- Svara med klassen `Monitor`, `main()`-metoden från `VideoMeetingApplikation` och övriga klasser/trådar/attribut/metoder du introducerat.

```
class Monitor {
    // add attributes and methods as you see fit
}

public class VideoMeetingApplication {

    // competence profiles for five instructors
    private static final List<Set<String>> INSTRUCTOR_COMPETENCE = List.of(
        Set.of("EDAA45", "EDAA80", "EDAA85", "EDAA90", "EDAA30", "EDAF85"),
        Set.of("EDAA80", "EDAA85", "EDAA90"),
        Set.of("EDAA45", "EDAA30", "EDAF85"),
        Set.of("EDAA80", "EDAA85", "EDAA30"),
        Set.of("EDAA90", "EDAA30", "EDAF85")
    );

    public static void main(String[] args) {
        // your updated main method
    }
}
```

(15p)

Preliminär poängguide

Uppgiften är förvisso stor poängmässigt men kan brytas ner i flera enskilda delar. Försök därför att lösa enskilda delar av uppgiften även om du inte känner att du får ihop helheten. Preliminärt kommer poängen att fördelas enligt följande mellan de olika delarna:

- Huvudprogram som skapar/startar nödvändiga trådar och datastrukturer – 2 poäng
- Klassen `Monitor` – 4 poäng
- Funktionalitet för att ta emot anslutande studenter och ställa dem i kö – 3 poäng
- Funktionalitet för att hantera de fem handledarna – 6 poäng

Poängfördelningen kan komma att justeras.