אוניברסיטת בן-גוריון

מדור בחינות

מספר נבחן:_____

רשמו תשובותיכם בגיליון התשובות בלבד
תשובות מחוץ לגיליון לא יבדקו.
שימו לב:

על תשובות ריקות בשאלות הפתוחות יינתן 20% מהניקוד!

תאריך הבחינה: 5.2.2018
שם המורה: ד"ר מני אדלר
ד"ר ערן טרייסטר
מר בני לוטטי
פרופ' אנדרי שרף
שם הקורס: תכנות מערכות
מספר הקורס: 202-1-2031
מיועד לתלמידי: מדעי המחשב, הנדסת תוכנה
שנה: תשע"ח
סמסטר: א'
מועד: א'
משך הבחינה: שלש שעות
חומר עזר: אסור

בהצלחה!

שאלה 1: ניהול זיכרון (30 נקודות)

ברצוננו לבנות מערכת של הרצת סרטי וידאו, בשחור לבן או בצבע. לשם כך נכתב הקוד הבא:

```
class Image {

public:
        Image(int n):n_pixels(n) {};
        virtual void show() = 0;

protected:
        int n_pixels;
};

class BWImage : public Image {

public:
        BWImage(int n):Image(n){data = new short[n_pixels];}
        virtual ~BWImage(){ delete[] data;}
        virtual void show(){/*iterate over data and print*/}
private:
        short* data;
};
```

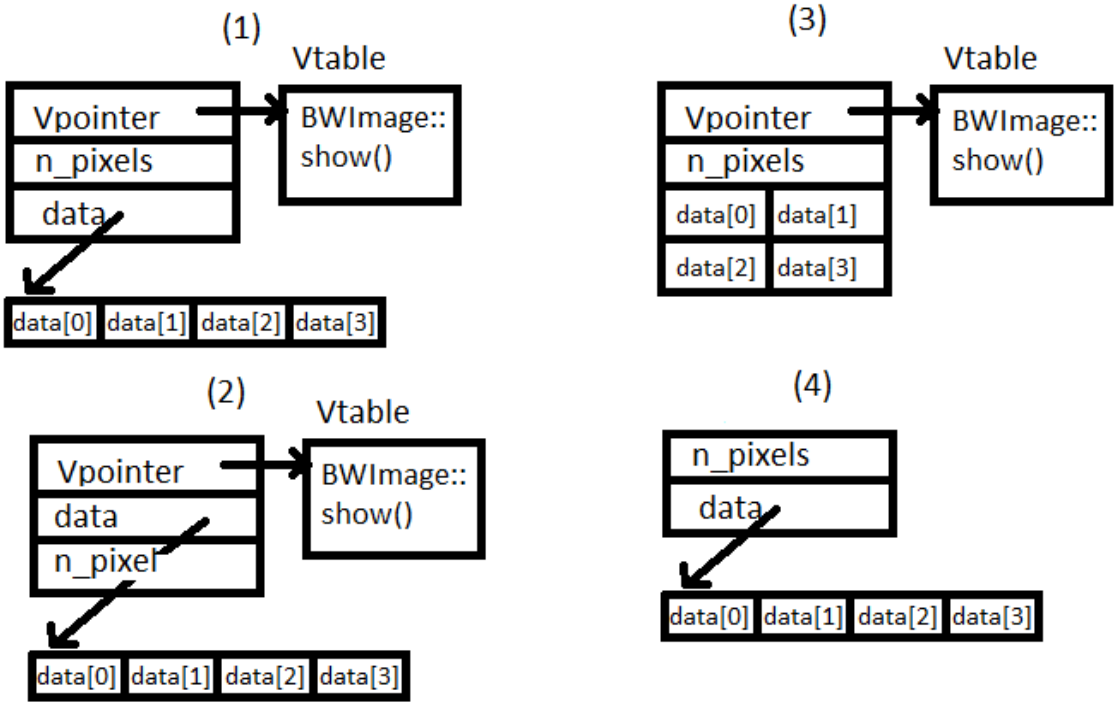
```
class RGB {
  short data[3];
};

class ColorImage : public Image{

public:
        ColorImage(int n):Image(n){
                data = new RGB[n_pixels];
        }
        virtual ~ColorImage(){
                delete[] data;
        }
        ColorImage(const ColorImage& rhs):Image(rhs.n_pixels){
                data = new RGB[n_pixels];
                copyData(rhs);
        }
        ColorImage& operator=(const ColorImage& rhs) {
                if (&rhs != this) {
                        n_pixels = rhs.n_pixels;
                        delete[] data;
                        data = new RGB[n_pixels];
                        copyData(rhs);
                }
                return *this;
        }
        virtual void show(){/*iterate over data and print*/}

 private:
        RGB* data;
        void copyData(const ColorImage& rhs){/*copies data*/}
};
```

א. אילו מהציורים הבאים מתארים נכונה את מבנה הזיכרון של המחלקה BWImage כפי שנלמד בקורס, בהתייחסות לשדות המחלקה ולפונקציה ()show בלבד. (הניחו כי מערך data באורך 4).



(5) אף אחד מהציורים אינו נכון, חסר לכולם השדה this.

ב. נתונה התכנית:

```
int main() {
        Image *C = new ColorImage(2);
        C -> show();
        delete C;
       return 0;
}
```

סמנו את הטענה הכי נכונה מהבאות לגבי ניהול הזיכרון בתכנית:

1. המערך data במחלקה ColorImge אינו משוחרר כמו שצריך ב-destructor. כיוון שבמחלקה RGB גם כן יש מערך, חייבים לעבור על מערך data איבר איבר ולשחרר את הזיכרון.
2. ישנה זליגת זיכרון הנובעת מכך שה-destructor של המחלקה ColorImg בכלל אינו נקרא. התכנית main() שגוייה, והשורה השנייה חייבת להשתנות ל: ;ColorImage *C = new ColorImage(2)
3. הפתרון בסעיף 2 אמנם יעבוד בתכנית הזו, אבל באופן כללי אינו נכון. כדי לפתור את הבעיה בצורה נכונה חייבים להוסיף למחלקה Image את השורה: ;virtual ~Image(){}
4. הטענה בסעיף 1 נכונה, אבל מעבר לכך חייבים לממש destructor למחלקה RGB, כיוון שיש גם שם מערך שחייבים לשחרר בעזרת delete.
5. אין זליגת זיכרון בתכנית.

כעת נניח בנוסף את התכנית הבאה (סעיפים ג,ד מתייחסות לפונקציה זו):

```
int main() {
        ColorImage *C = new ColorImage(2);
        BWImage *B = new BWImage(2);
        BWImage B2(2);
        B2 = *B;
        ColorImage C2 = *C;
        delete C;
        delete B;
        return 0;
}
```

ג. סמנו את הטענה הכי נכונה (עבור הפונקציה ()main בראש העמוד):

1. המשתנה C מצביע לערך של מחלקה המוקצה דינאמית על ה-heap, ומשוחרר ע"י delete בסוף התכנית.
2. המשתנה C2 הינו ערך של מחלקה שמוקצה סטאטית על ה-stack. מערך ה-data בתוכו מוקצה דינאמית על ה-heap. ערך המחלקה ישוחרר בסוף הפונקציה main אך מערך ה-data בתוכו לא ישוחרר.
3. המשתנה C2 הינו ערך של מחלקה שמוקצה סטאטית על ה-stack. מערך הdata בתוכו מוקצה דינאמית על ה-heap. בסוף הפונקציה main ייקרא ה-destructor של המחלקה ColorImage והמערך data ישוחרר.
4. גודל הזיכרון ב-stack של ערך המשתנה B2 לאחר יצירתו בשורה השלישית ישתנה אם נאתחל את B2 (נקרא ל-constructor) עם ערך שונה מ-2.
5. טענות 1 ו-3 נכונות.

ד. סמנו את הטענה הכי נכונה מהבאות (עבור הפונקציה ()main בראש העמוד):

1. ישנה זליגת זיכרון - המערך data שהוקצה בבניית המשתנה B2 אינו משוחרר, כיוון שהמחלקה BWImage איננה ממלאת את "חוק ה-3". בתכנית נקרא ה-copy assignment operator שממומש כברירת מחדל עבור BWImage, ומבוצעת העתקה רדודה של המצביע ל-data.
2. ישנה זליגת זיכרון כיוון שהמשתנה C2 אינו משוחרר. יש להוסיף את השורה: delete &C2
3. טענה 1 נכונה, אך בנוסף התכנית לא יעילה בכלל. ביצירת המשתנה C2, מופעל תחילה ה-constructor של המחלקה ColorImage, ואז מופעל ה-Copy Assignment Operator ומעתיק את הערך של C אל תוך C2, תוך כדי שחרור והקצאה מיותרים של זיכרון. כדי לפתור את המקרה הזה צריך לממש פונקציות נוספות על פי 'חוק ה-5'.
4. המחלקה ColorImage מממשת את 'חוק ה-3' נכון. ביצירת C2 מופעל ה-copy constructor שמעתיק את המערך data למערך חדש. בסוף התכנית הזיכרון של המשתנים C2 ו-C משוחרר.
5. טענות 1 ו-4 נכונות.

בהנחה שהקוד לעיל תוקן, המשיכו הסטודנטים לממש את תכנית הרצת הווידאו:

```
class Movie {
   int n_frames;
   Image** frames; // an array of pointers to Image
   private:
            Movie(int n_f,int n_p, Image** f):n_frames(n_f),frames(f){}
   public:
            static Movie loadMovie(string filename){
                /*Loads a movie from a file.
                  All frames are dynamically allocated.*/
           }
            Movie(const Movie& rhs):n_frames(rhs.n_frames){
                        /*performs deep copy of frames*/
           }
            Movie& operator=(const Movie& rhs) {
                      /*deeply deletes frames and performs deep copy of rhs*/
           }
            virtual void show(){/*shows the movie*/}
            virtual ~Movie(){/* a deep delete of frames*/}
};

int main(){
            Movie mov = Movie::loadMovie("MivtsaSavta");
            mov.show();
           return 0;
}
```

ה. סטודנט הריץ את התכנית וטען סרט וידאו גדול, אך ראה שימוש מוגזם לדעתו בזיכרון. איזו מהטענות נכונה:

1. הפעלת constructor בתוך פונקציה סטאטית (Factory Design) היא אינה יעילה וחייבים לממש constructor 'רגיל'.
2. האובייקט המוחזר מ loadMovie() מועתק שלא לצורך, ולכן יש שימוש בהרבה זיכרון. ניתן לפתור זאת ע"י מימוש move constructor אשר 'יגנוב' את מערך התמונות מהסרט שנוצר, ללא העתקה מיותרת.
3. טענה 2 נכונה למעט העובדה שהפונקציה הרלוונטית פה היא move assignment operator.
4. אם נשנה את הפונקציה ב-main להיות:

```
int main() {
          Movie* M = &(Movie::loadMovie("MivtsaSafta.mov"));
          M->show();
          delete M;
}
```

אז הכל יהיה בסדר ותיפתר בעיית העתקת הזיכרון, כיוון שנעבוד ישירות רק עם המצביעים.

5. טענות 2 ו-4 נכונות.

שאלה 2: מקביליות (30 נקודות)

נתון הממשק Stack, המגדיר מחסנית בגודל חסום לשמירת אובייקטים (כולל כאלה שהם null):

```
interface Stack<T> {
        void push(T data);  // the pushed data can be null
        T pop();
        int size();         // return the current number of items in the stack
        int capacity();   // returns the maximal number of items which can be stored in the stack
}
```

א. הגדירו תכונה נשמרת לממשק (5 נקודות)

המחלקה LinkedStack מממשת את הממשק Stack באופן הבא:

```
class Link<T> {

   public Link<T> next;
   public T data;

   public Link(Link<T> next, T data) {
      this.next = next;
      this.data = data;
   }
}

class LinkedStack<T> implements Stack<T> {

   private Link<T> head;
   private int size;
   private final int capacity;

   LinkedStack(int capacity) throws Exception {
        if (capacity < 1)
                throw new Exception("Illegal capacity: " + capacity);
        this.capacity = capacity;
        this.head = null;
        this.size = 0;
   }
```

```
    public void push(T data) {
        if (size < capacity) {
            head = new Link<>(head, data);
            size++;
        }
    }

    public T pop() {
        if (size > 0) {
            T ret = head.data;
            head = head.next;
            size--;
            return ret;
        } else
            return null;
    }

    public int size () {
        return size;
    }

    public int capacity () {
        return capacity;
    }

}
```

נתונים שני ת'רדים T1,T2 הניגשים למופע משותף של המחלקה LinkedSatck.
נניח כי אין בעיית visibility.

ב. תארו בקצרה תרחיש בו הרצת הת'רדים אינה בטוחה [מבחינת השמירה על האינוריאנטה] (5 נקודות)

ג. תארו בקצרה תרחיש בו הרצת הת'רדים בטוחה אך אינה נכונה (5 נקודות)

ד. עדכנו את המימוש כך שהוא יהיה בטוח ונכון. אין להשתמש במילה השמורה synchronized, ולא במחלקות הממומשות בעזרת synchronized. (15 נקודות)

שאלה 3: תקשורת (30 נקודות)

בנספח למבחן מופיע קוד ה-Reactor כפי שנלמד בכיתה וכפי שניתן בתרגיל 3.

בחברת ACME המשתמשת ב-Reactor גילו שכמות התעבורה העוברת בשרת שלהם גורמת לעומס על הת'רד שמטפל ב-selector (ה-selector thread)

א. שרגא, ראש צוות הIT, הציע להוסיף עוד worker threads (הת'רדים שנמצאים ב-ActorThreadPool) ולהעביר את השרת למכונה עם יותר מעבדים כדי להפחית מהעומס על ה-selector thread. האם מה ששרגא מציע יכול לעבוד? הסבירו בפירוט. (6 נקודות)

ב. נחום, ראש צוות הפיתוח, שלא שמע את מה ששרגא הציע החליט לשדרג את ה-reactor על ידי הוספת תמיכה בשני selector threads בדרך הבאה:

a. לכל selector thread יהיה Selector משלו
b. ה-selector thread הראשון יהיה אחראי על ה-ServerSocketChannel
c. כאשר client חדש יתחבר הוא יגיע ל-selector thread הראשון והוא יבחר את ה-selector thread שמטפל בהכי פחות clients להיות אחראי על אותו ה-SocketChannel (מבין השניים שיש, אם שניהם עמוסים באותו המידה הוא יבחר את עצמו)
d. מאותו הרגע רק ה-selector thread הנבחר יטפל באותו ה-SocketChannel

הסבירו, מדוע הצעתו של נחום תעבוד? (4 נקודות)

ג. ממשו את הצעתו של נחום, אין צורך לרשום את כל הקוד של ה-reactor ניתן רק לציין באילו מחלקות אילו שינויים ידרשו ולרשום את הקוד שבשינויי בלבד. (20 נקודות)

שאלה 4: בסיסי נתונים (10 נקודות)

בקורס השנתי אסקיו-אל הצטברו במשך השנים הרבות שאלות ממבחנים. כחלק מתהליך התייעלות בקורס, רוצים לארגן את השאלות במסד נתונים רלציוני.
כל שאלה מאופיינת בטקסט שלה, שנים בה השאלה ניתנה במועד א' והשאלה שניתנה במקומה במועד ב' באותה השנה (ולהיפך). יש להניח כי שאלה מסוימת יכולה לחזור על עצמה בשנים שונות אך לא תינתן פעמיים באותה שנה.

א. בחרו את השלמת 2 פקודות הSQL אשר מייצרת את המסד הנתונים הרלציוני הנכון והיעיל:

```
CREATE TABLE questions (
id              INT     PRIMARY KEY,
text            TEXT    NOT NULL
);
CREATE TABLE questions2years (
question_id_A  INT      NOT NULL,
question_id_B  INT      NOT NULL,
year            INT     NOT NULL,
FOREIGN KEY(question_id_A)  REFERENCES questions(id),
FOREIGN KEY(question_id_B)  REFERENCES questions(id),
PRIMARY KEY (_________________)
);

CREATE TABLE grades (
question_id     INT     NOT NULL,
grade           INT     NOT NULL,
year            INT     NOT NULL,
FOREIGN KEY(question_id)  REFERENCES questions(id),
PRIMARY KEY (_____________)
);
```

1.
None
question_id, year

2.
question_id_A, question_id_B
question_id, year

3.
question_id_A, question_id_B, year
question_id, year

4.
question_id_A, question_id_B
question_id, grade

5.
question_id_B, year
None

ב. בחרו את השאילתה היעילה המחזירה את כל השאלות (טקסט) שניתנו במועד ב' עבור שאלות שניתנו במועד א' של 2018 ושהציון הממוצע בהם (בשאלות במועד א') קטן מ-56?

1. SELECT questions.text FROM grades JOIN questions ON grades.question_id = questions.id WHERE grades.year=2018 AND grades.grade<56;

2. SELECT questions.text FROM questions2years JOIN questions ON questions2years.question_id_B = questions.id JOIN grades ON questions.id = grades.question_id WHERE questions2years.year=2018 AND grades.grade<56;

3. SELECT questions.text FROM questions JOIN grades ON questions.id = grades.question_id WHERE grades.year=2018 AND grades.grade<56;

4. SELECT questions.text FROM questions2years JOIN questions ON questions2years.question_id_B = questions.id JOIN grades ON questions2years.question_id_A = grades.question_id WHERE questions2years.year=2018 AND grades.grade<56;

5. אף תשובה לא נכונה

נספח: קוד ה Reactor, כפי שנלמד בכיתה וניתן בתרגיל 3:

```
public class Reactor {

    private final int port;
    private final Supplier<MessagingProtocol<T>> protocolFactory;
    private final Supplier<MessageEncoderDecoder<T>> readerFactory;
    private ActorThreadPool<NonBlockingConnectionHandler> pool;
    private Selector selector;

    private Thread selectorThread;
    private final ConcurrentLinkedQueue<Runnable> selectorTasks = new ConcurrentLinkedQueue<>();

    public Reactor(
            int numThreads,
            int port,
            Supplier<MessagingProtocol<T>> protocolFactory,
            Supplier<MessageEncoderDecoder<T>> readerFactory) {

        this.pool = Executors.newFixedThreadPool(numThreads);
        this.port = port;
        this.protocolFactory = protocolFactory;
        this.readerFactory = readerFactory;
    }

    @Override
    public void serve() {
        selectorThread = Thread.currentThread();

        try (   Selector selector = Selector.open();
                ServerSocketChannel serverSock = ServerSocketChannel.open()) {

            this.selector = selector; //just to be able to close

            serverSock.bind(new InetSocketAddress(port));
            serverSock.configureBlocking(false);
            serverSock.register(selector, SelectionKey.OP_ACCEPT);

            while (!selectorThread.isInterrupted()) {

                selector.select();
                runSelectionThreadTasks();
```

```
            for (SelectionKey key : selector.selectedKeys()) {

                if (!key.isValid()) {
                    continue;
                } else if (key.isAcceptable()) {
                    handleAccept(serverSock, selector);
                } else {
                    handleReadWrite(key);
                }
            }

            selector.selectedKeys().clear(); //clear the selected keys set so that we can know about new events

        }

    } catch (ClosedSelectorException ex) {
        //do nothing - server was requested to be closed
    } catch (IOException ex) {
        //this is an error
        ex.printStackTrace();
    }

    System.out.println("server closed!!!");
    pool.shutdown();
}

void updateInterestedOps(SocketChannel chan, int ops) {
    final SelectionKey key = chan.keyFor(selector);
    if (Thread.currentThread() == selectorThread) {
        key.interestOps(ops);
    } else {
        selectorTasks.add(() -> {
            key.interestOps(ops);
        });
        selector.wakeup();
    }
}


private void handleAccept(ServerSocketChannel serverChan, Selector selector) throws IOException {
    SocketChannel clientChan = serverChan.accept();
    clientChan.configureBlocking(false);
    final NonBlockingConnectionHandler handler = new NonBlockingConnectionHandler(
```

```
            readerFactory.get(),
            protocolFactory.get(),
            clientChan,
            this);

        clientChan.register(selector, SelectionKey.OP_READ, handler);
    }

private void handleReadWrite(SelectionKey key) {
        NonBlockingConnectionHandler handler = (NonBlockingConnectionHandler) key.attachment();
        if (key.isReadable()) {
            Runnable task = handler.continueRead();
            if (task != null) {
                pool.submit(handler, task);
            }
        } else {
            handler.continueWrite();
        }
    }

    private void runSelectionThreadTasks() {
        while (!selectorTasks.isEmpty()) {
            selectorTasks.remove().run();
        }
    }

    @Override
    public void close() throws IOException {
        selector.close();
    }
}
```

```
public class NonBlockingConnectionHandler {
private static final int BUFFER_ALLOCATION_SIZE = 1 << 13; //8k
    private static final ConcurrentLinkedQueue<ByteBuffer> BUFFER_POOL = new ConcurrentLinkedQueue<>();

    private final MessagingProtocol<T> protocol;
    private final MessageEncoderDecoder<T> encdec;
    private final Queue<ByteBuffer> writeQueue = new ConcurrentLinkedQueue<>();
    private final SocketChannel chan;
    private final Reactor reactor;
```

```
public NonBlockingConnectionHandler(
        MessageEncoderDecoder<T> reader,
        MessagingProtocol<T> protocol,
        SocketChannel chan,
        Reactor reactor) {
    this.chan = chan;
    this.encdec = reader;
    this.protocol = protocol;
    this.reactor = reactor;
}

public Runnable continueRead() {
    ByteBuffer buf = leaseBuffer();

    boolean success = false;
    try {
        success = chan.read(buf) != -1;
    } catch (ClosedByInterruptException ex) {
        Thread.currentThread().interrupt();
    } catch (IOException ex) {
        ex.printStackTrace();
    }

    if (success) {
        buf.flip();
        return () -> {
            try {
                while (buf.hasRemaining()) {
                    T nextMessage = encdec.decodeNextByte(buf.get());
                    if (nextMessage != null) {
                        T response = protocol.process(nextMessage);
                        if (response != null) {
                            writeQueue.add(ByteBuffer.wrap(encdec.encode(response)));
                            reactor.updateInterestedOps(chan, SelectionKey.OP_READ | SelectionKey.OP_WRITE);
                        }
                    }
                }
            } finally {
                releaseBuffer(buf);
            }
```

```
        };
    } else {
        releaseBuffer(buf);
        close();
        return null;
    }

}

public void close() {
    try {
        chan.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}

public void continueWrite() {
    while (!writeQueue.isEmpty()) {
        try {
            ByteBuffer top = writeQueue.peek();
            chan.write(top);
            if (top.hasRemaining()) {
                return;
            } else {
                writeQueue.remove();
            }
        } catch (IOException ex) {
            ex.printStackTrace();
            close();
        }
    }

    if (writeQueue.isEmpty()) {
        if (protocol.shouldTerminate()) close();
        else reactor.updateInterestedOps(chan, SelectionKey.OP_READ);
    }
}

private static ByteBuffer leaseBuffer() {
    ByteBuffer buff = BUFFER_POOL.poll();
```

```
        if (buff == null) {
            return ByteBuffer.allocateDirect(BUFFER_ALLOCATION_SIZE);
        }

        buff.clear();
        return buff;
    }

    private static void releaseBuffer(ByteBuffer buff) {
        BUFFER_POOL.add(buff);
    }
}
```

```
public class ActorThreadPool<T> {

    private final Map<T, Queue<Runnable>> acts;
    private final ReadWriteLock actsRWLock;
    private final Set<T> playingNow;
    private final ExecutorService threads;

    public ActorThreadPool(int threads) {
        this.threads = Executors.newFixedThreadPool(threads);
        acts = new WeakHashMap<>();
        playingNow = ConcurrentHashMap.newKeySet();
        actsRWLock = new ReentrantReadWriteLock();
    }

    public void submit(T act, Runnable r) {
        synchronized (act) {
            if (!playingNow.contains(act)) {
                playingNow.add(act);
                execute(r, act);
            } else {
                getActsQueue(act).add(r);
            }
        }
    }

    public void shutdown() {
        threads.shutdownNow();
    }
```

```
    private Queue<Runnable> pendingRunnablesOf(T act) {

        actsRWLock.readLock().lock();
        Queue<Runnable> pendingRunnables = acts.get(act);
        actsRWLock.readLock().unlock();

        if (pendingRunnables == null) {
            actsRWLock.writeLock().lock();
            acts.put(act, pendingRunnables = new LinkedList<>());
            actsRWLock.writeLock().unlock();
        }
        return pendingRunnables;
    }

    private void execute(Runnable r, T act) {
        threads.submit(() -> {
            try {
                r.run();
            } finally {
                complete(act);
            }
        });
    }

    private void complete(T act) {
        synchronized (act) {
            Queue<Runnable> pending = pendingRunnablesOf(act);
            if (pending.isEmpty()) {
                playingNow.remove(act);
            } else {
                execute(pending.poll(), act);
            }
        }
    }

}
```