1612918
张志毅

1612919
钟震宇

九公元能  日新月異

JavaScript 多线程同步机制 研究成果展示

操作系统课程作业

PART 1

工作内容及分工

完成的工作

- 选定项目及准备工作 | 张志毅
- 代码基础部分及总体结构搭建 | 张志毅
- 信号量与互斥锁的实现 | 钟震宇
- 展示的准备工作 | 钟震宇

PART 2

选题依据

我们为什么做这个课题

选题依据

我们为什么做这个课题

JavaScript 是一种很高级的语言，几乎不提供与硬件直接交互的接口和能力，乍看起来与《操作系统》这门课程驴唇不对马嘴。

但是随着 JavaScript 在越来越多领域的应用，它已经从单纯的网页脚本发展为横跨前端后端客户端的全栈开发语言。因此，我们对 JavaScript 的要求也越来越高。

选题依据

我们为什么做这个课题

本课程设计作业利用了《操作系统》中的多线程锁机制相关的知识，实现了 JavaScript 的多线程同步，包括信号量与互斥锁的实现，是操作系统的底层知识在高级编程语言的应用。

PART 3

项目背景

历史遗留问题

多线程支持

线程同步机制引入

项目可行性

1

项目背景

历史遗留问题

JavaScript 设计之初，开发者考虑到了冲突问题。

如果运行 JavaScript 使用多线程，那么可能出现不同进程同时操作 DOM 中同一个元素的问题，会由此引发混乱，必须引入线程同步机制。

但由于使用浏览器进行硬件相关的原子性操作极为困难，且彼时计算机性能不强，所以开发者在解决这一问题时采用的策略是使 JavaScript 只支持单线程，完全规避了线程同步这一问题。

2

项目背景

多线程支持

由于计算机性能的快速发展，以及部分场景下需要 JavaScript 支持异步，HTML5 标准提出后，JavaScript 开始支持多线程。

但严格意义来说，此时的 JavaScript 仍是单线程的，多线程是由主线程开启多个 worker， worker 与主线程之间不直接共享数据，数据传递采用拷贝，因此仍然不需要引入线程同步机制。

3

项目背景

线程同步机制的引入

拷贝传值在某些应用场景下仍不能满足需求。因此，在 JavaScript ES9 标准中， JavaScript 可以使用 SharedArrayBuffer 实现主线程和 worker 之间的数据共享（读写同一块内存），这种方法在解决需求的同时引入了数据访问冲突的问题，需要引入线程同步机制。

4

项目背景

项目可行性

JavaScript ES8 中提供了一套名为 Atomics 的基础 API，可以针对共享的内存提供原子操作。例用 Atomics，可以实现一套 JavaScript 的多线程同步机制。

PART 4

具体实现

信号量与互斥锁

semaphore.js 信号量部分

```
class Semaphore {
    constructor(value, buffer) {
        if (buffer) {
            this.buffer = buffer
        } else {
            this.buffer = new SharedArrayBuffer(4 * value)
        }
        this.value = value
        this.current_value = -1
        this.view = new Int32Array(this.buffer)
    }

    static fromSharedBuffer(value, buffer) {
        return new Semaphore(value, buffer)
    }

    p() {
        while (true) {
            for (let i = 0; i < this.value; i++) {
                if (Atomics.compareExchange(this.view, i, 0, 1) === 0) {
                    this.current_value = i
                    return;
                }
            }
        }
    }

    v() {
        Atomics.store(this.view, this.current_value, 0)
        this.current_value = -1
    }
}

```

semaphore.js 互斥锁部分

```
34  class Mutex extends Semaphore {
35      constructor(buffer) {
36          super(1, buffer)
37      }
38
39      static fromSharedBuffer(buffer) {
40          return new Mutex(buffer)
41      }
42
43      waitOne() {
44          this.p()
45      }
46
47      releaseMutex() {
48          this.v()
49      }
50  }
51
```

main.js 主线程

```
1  const mutex = new Semaphore(1)
2  // const mutex = new Mutex()
3
4  workers = []
5
6  for (let i = 0; i < 4; i++) {
7      workers.push(new Worker('./worker.js'))
8  }
9
10 for (let i = 0; i < 4; i++) {
11     workers[i].postMessage(
12         {
13             index: i,
14             shared_buffer: mutex.buffer
15         }
16     )
17 }
18
```

worker.js 子线程

```
importScripts('./semaphore.js')

onmessage = msg => {
    const index = msg.data.index
    const buffer = msg.data.shared_buffer
    const mutex = Semaphore.fromSharedBuffer(1, buffer)
    // const mutex = Mutex.fromSharedBuffer(buffer)

    mutex.p()
    // mutex.waitOne()

    for (let i = 0; i < 5; i++) {
        console.log(index + '-' + i)
    }

    mutex.v()
    // mutex.releaseMutex()
}

```

非同步

Elements Console Sources Network Performance

top Filter Default levels

```
0-0                                                    worker.js:11
1-0                                                    worker.js:11
0-1                                                    worker.js:11
1-1                                                    worker.js:11
0-2                                                    worker.js:11
1-2                                                    worker.js:11
1-3                                                    worker.js:11
0-3                                                    worker.js:11
1-4                                                    worker.js:11
0-4                                                    worker.js:11
3-0                                                    worker.js:11
2-0                                                    worker.js:11
3-1                                                    worker.js:11
3-2                                                    worker.js:11
3-3                                                    worker.js:11
3-4                                                    worker.js:11
2-1                                                    worker.js:11
2-2                                                    worker.js:11
2-3                                                    worker.js:11
2-4                                                    worker.js:11
```

左边的数字（线程号）有交错

>

同步

Elements Console Sources Network Performance

top Filter Default levels

```
0-0                                                    worker.js:11
0-1                                                    worker.js:11
0-2                                                    worker.js:11
0-3                                                    worker.js:11
0-4                                                    worker.js:11
1-0                                                    worker.js:11
1-1                                                    worker.js:11
1-2                                                    worker.js:11
1-3                                                    worker.js:11
1-4                                                    worker.js:11
3-0                                                    worker.js:11
3-1                                                    worker.js:11
3-2                                                    worker.js:11
3-3                                                    worker.js:11
3-4                                                    worker.js:11
2-0                                                    worker.js:11
2-1                                                    worker.js:11
2-2                                                    worker.js:11
2-3                                                    worker.js:11
2-4                                                    worker.js:11
```

左边的数字（线程号）无交错

>

九公元能  日新月異

JavaScript 多线程同步机制 研究成果展示

操作系统课程作业