به نام خدا

امیر ارسلان صنعتی ۹۹۲۴۳۰۴۷

گزارش پروژه سیستم عامل

ابتدا به نحوه پیاده سازی الگوریتم های page-replacement میپردازیم و سپس نحوه پیاده سازی thread ها و socket ها را بیان میکنیم.

LRU:

نحوه پیدا کردن least recently used page با استفاده از clock بود اما نه دقیقا clock سیستم. هر عددی که سرور وارد میکند ، یک متغییر داریم که بیانگر تعداد اعداد وارد شده توسط سرور است که هر بار سرور عددی وارد میکند این متغییر یک واحد زیاد می شود. از این رو می توان با این متغییر بیان کرد که کدام page در چه زمانی وارد شده (اگر مقدار count یک page بیشتر باشد یعنی در بین اعدادی که سرور وارد کرده جلو تر از بقیه است).

در ابتدای اجرای تابع چک میکنیم که data از قبل وجود داشته یا نه که اگر وجود داشت time آن آپدیت میشود. سپس اگر از قبل وجود نداشت ، اگر جا داشتیم این عدد را جا می دهیم، اگر جا نداشتیم min بین time ها را پیدا کرده و با آن replace میکنیم.

```
    private void LRU(int data, int count) {
        for (int i = 0; i < lruIndex; i++) {
            if(lruObjects[i] != null) {
                if(lruObjects[i].data == data) {
                    lruObjects[i].clock = count;
                    return;
                }
            }
        }
        if(lruIndex < size) {
            lruObjects[lruIndex] = new LRUObject();
            lruObjects[lruIndex].data = data;
            lruObjects[lruIndex].clock = count;
            ++lruIndex;
        } else {
            int minClk = lruObjects[0].clock;
            int minI = 0;
            for (int i = 1; i < size; i++) {
                if(lruObjects[i].clock < minClk) {
                    minI = i;
                    minClk = lruObjects[i].clock;
                }
            }
            lruObjects[minI].clock = count;
            lruObjects[minI].data = data;
        }
        ++lruPageFault;
    }
```

```
class LRUObject {
    public int data;
    public int clock;
}
```

Second-chance:

در این روش هم مانند LRU ، بین داده ها سرچ میکنیم تا اگر وجود داشت reference bit را یک کند. در ادامه باز، اگر داده پیدا نشد و جا داشتیم، مانند یک queue جا را به آن اختصاص میدهیم. اگر جا نبود مطابق الگوریتم second-chance جلو میرویم تا یک reference bit = 0 پیدا کنیم. و reference bit های یک

را هم صفر میکنیم. هر داده ای که تازه وارد آرایه می شود (چه با replace کردن چه اگر جا داشتیم) reference bit اش صفر است.

```
private void secondChance(int data) {
    for (int i = 0; i < secondSize; i++) {
        if(secondChanceObjects[i] != null) {
            if(secondChanceObjects[i].data == data) {
                secondChanceObjects[i].referenceBit = 1;
                return;
            }
        }
    }
    if(secondSize < size) {
        secondChanceObjects[secondFirst] = new SecondChanceObject();
        secondChanceObjects[secondFirst].data = data;
        secondChanceObjects[secondFirst].referenceBit = 0;
        secondFirst = (secondFirst+1)%size;
        ++secondSize;
    } else {
        while(secondChanceObjects[secondFirst].referenceBit != 0) {
            secondChanceObjects[secondFirst].referenceBit = 0;
            secondFirst = (secondFirst+1)%size;
        }
        secondChanceObjects[secondFirst].data = data;
        secondChanceObjects[secondFirst].referenceBit = 0;
        secondFirst = (secondFirst+1)%size;
    }
    ++secondPageFault;
}
```

```
class SecondChanceObject {
    public int data;
    public int referenceBit = 0;
}
```

FIFO:

در اینجا هم اول سرچ میکنیم، اگر جا داشتیم جا میدهیم اگر نداشتیم به ترتیب FIFO ، replace میکنیم.

```
    private void FIFO(int data) {
        for (int i = 0; i < fifoLast; i++) {
            if (fifo[i] == data)
                return;
        }
        if(fifoLast < size) {
            fifo[fifoLast++] = data;
        } else {
            fifo[fifoFirst] = data;
            fifoFirst = (fifoFirst+1)%size;
        }
        ++fifoPageFault;
    }
```

Thread implementation and semaphores and socket:

طوری که مسئله مدل شده مانند مسئله producer consumer است به نحوی که کلاس اصلی (کلاسی که به سرور متصل می شود) producer است برای consumer که این consumer کار page replacement را انجام میدهد. یک بافر بین producer و consumer وجود دارد . producer اول چک میکند که بافر پر نباشد (semaphore empty) بعد باید یک قفل برای دسترسی به بافر(منبع مشترک) بگیرد (semaphore buffer) داده مورد نظر که از سرور گرفته بود را داخل buffer بگذارد. تعداد خانه های پر buffer را یکی زیاد کند (semaphore full) و در نهایت قفلی که برای نوشتن داده در بافر گرفته بود را آزاد کند. اگر هم از سرور صفر دریافت کرد، آن داده را در بافر قرار میدهد تا consumer را مطلع سازد و در آخر هم برای consumer صبر میکند تا کارش تمام شود بعد از همه اینها کار برنامه تمام میشود.

```
int[] bufferArray;
int buffer_index = 0, data, n;
Semaphore full, empty, buffer;
try(Socket socket = new Socket( host: "localhost",  port: 8080);
    DataInputStream din = new DataInputStream(socket.getInputStream())) {
    n = din.readInt();
    System.out.println("n is : " + n);
    full = new Semaphore( permits: 0);
    empty = new Semaphore(n);
    buffer = new Semaphore( permits: 1);
    bufferArray = new int[n];
    Consumer consumer = new Consumer(full, empty, buffer, n, bufferArray);
    consumer.start();
    while(true) {
        data = din.readInt();
        System.out.println("producer received the data");
        empty.acquire();
        buffer.acquire();
        //System.out.println("buffer index : " + buffer_index);
        bufferArray[buffer_index] = data;
        buffer_index = (buffer_index+1)%n;
        //System.out.println("new buffer index : " + buffer_index);
        full.release();
        buffer.release();
        if(n == 0) {
            consumer.join();
            break;
        }
    }
}catch (Exception e) {
    System.out.println("process finished");
```

Consumer Thread implementation:

```java
import java.util.concurrent.Semaphore;

public class Consumer extends Thread {
    private final Semaphore full;
    private final Semaphore empty;
    private final Semaphore buffer;
    private final LRUObject[] lruObjects;
    private final SecondChanceObject[] secondChanceObjects;
    private final int[] fifo;
    private final int[] bufferArray;
    private final int size;
    private int fifoFirst = 0;
    private int fifoLast = 0;
    private int secondFirst = 0;
    private int secondSize = 0;
    private int bufferIndex = 0;
    private int dataCount = 0;
    private int lruIndex = 0;
    private int lruPageFault = 0, fifoPageFault = 0, secondPageFault = 0;

    public Consumer(Semaphore full, Semaphore empty, Semaphore buffer, int size, int[] bufferArray) {
        super();
        this.full = full;
        this.empty = empty;
        this.buffer = buffer;
        this.size = size;
        this.bufferArray = bufferArray;
        lruObjects = new LRUObject[size];
        secondChanceObjects = new SecondChanceObject[size];
        fifo = new int[size];
    }
```

```java
    @Override
    public void run() {
        while(true) {
            try {
                full.acquire();
                buffer.acquire();
                int data = bufferArray[bufferIndex];
                bufferIndex = (bufferIndex+1)%size;
                empty.release();
                buffer.release();
                System.out.println("consumer received the data");
                if(data != 0) {
                    LRU(data, ++dataCount);
                    secondChance(data);
                    FIFO(data);
                    printTables(data);
                } else {
                    System.out.println("LRU : " + lruPageFault + ", FIFO : " + fifoPageFault + ", Second-chance : " + secondPageFault);
                    break;
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
```

Consumer هم اول چک میکند بافر خالی نباشد، سپس قفل میگیرد، داده را میخواند، تعداد خانه های خالی را اضافه میکند، قفل را رها میکند و سپس به الگوریتم های گفته شده میپردازد (برای این که هرچه سریع تر قفل آزاد بشه اجرای الگوریتم ها خارج از محدوده قفل آورده شده).

Print tables:

```
private void printTables(int data) {
    System.out.println("for data : " + data);
    /* printing LRU results : */
    System.out.print("LRU :\n[");
    for (int i = 0; i < size-1; i++) {
        if(lruObjects[i] != null)
            System.out.print(" " + lruObjects[i].data + ",");
        else
            System.out.print(" free table,");
    }
    if(lruObjects[size-1] != null)
        System.out.println(" " + lruObjects[size-1].data + "]");
    else
        System.out.println(" free table]");
    /* printing Second-chance results : */
    System.out.print("Second-chance :\n[");
    for (int i = 0; i < size-1; i++) {
        if(secondChanceObjects[i] != null)
            System.out.print(" " + secondChanceObjects[i].data + ",");
        else
            System.out.print(" free table,");
    }
    if(secondChanceObjects[size-1] != null)
        System.out.println(" " + secondChanceObjects[size-1].data + "]");
    else
        System.out.println(" free table]");
    /* printing FIFO results : */
    System.out.print("FIFO :\n[");
    for (int i = 0; i < size-1; i++) {
        if(fifo[i] != 0)
            System.out.print(" " + fifo[i] + ",");
```

```
        for (int i = 0; i < size-1; i++) {
            if(fifo[i] != 0)
                System.out.print(" " + fifo[i] + ",");
            else
                System.out.print(" free table,");
        }
        if(fifo[size-1] != 0)
            System.out.println(" " + fifo[size-1] + "]");
        else
            System.out.println(" free table]");

    }
}
```

پایان :)