گزارش پروژه سیستم عامل

مائده دهقان - ۹۹۲۴۳۰۳۴

ابتدا با in داده ها را از سمت سرور می‌گیریم.

سه ترد fifo/lru/sch را از سه نوع الگوریتم خواسته شده می‌سازیم.

سپس درخواست ها را برای هرکدام از آنها میفرستیم.

```java
public class client {
    static final int PORT = 8080;
    public static void main(String[] args) throws IOException {
        Socket socket = new Socket("localhost",PORT);
        DataInputStream in = new DataInputStream(socket.getInputStream());
        int next;
        int capacity = in.readInt();
        FIFO fifo = new FIFO(capacity);
        LRU lru = new LRU(capacity);
        secondChance sch = new secondChance(capacity);
        new Thread(fifo).start();
        new Thread(lru).start();
        new Thread(sch).start();
        do{

            next = in.readInt();
            fifo.requests.add(next);
            lru.requests.add(next);
            sch.requests.add(next);

        }while(next!=0);
        System.out.println("FIFO:");
        System.out.println(fifo.pageFaults);
        System.out.println("LRU:");
        System.out.println(lru.pageFaults);
        System.out.println("Second Chance:");
        System.out.println(sch.pageFaults);
    }
}
```

الگوریتم FIFO:
این الگوریتم سه صف دارد.
Queue: صفی برای نگهداری ترتیب ورود و خروج اعداد. الگوریتم روی این صف پیاده سازی می‌شود.
Tables: مشتری های رستوران را با میز آنها نگهداری می‌کند.
Request: برای ارتباط با برنامه client است. به طوری که داده‌ها از کلاینت به این صف فرستاده می‌شوند. چون بین دو ترد مشترک است باید از concurrenthashmap استفاده کنیم.
متغیرها:
Capacity: تعداد میزها را نگه می‌دارد.
Counter:برای شماره گذاری میزها استفاده می‌شود.

```
public class FIFO implements Runnable {
    public int new_data;
    public Queue<Integer> queue = new LinkedList<Integer>();
    public Queue<Integer> requests = new ConcurrentLinkedQueue<Integer>();
    public LinkedList<Customer> tables = new LinkedList<Customer>();
    public int capacity;
    public int pageFaults=0;
    public int counter = 1;
    public FIFO(int capacity) {
        this.capacity = capacity;
```

در پیاده سازی متد run، با استفاده از یک حلقه(while(true، همواره چک می‌کنیم که اگر ریکوستی آمد به آن جواب دهیم.
اول چک می‌کنیم که آیا صفمان دارای عنصر مورد نظر هست یا خیر. اگر بود فقط این ریکوئست را از صف درخواست ها پاپ می‌کنیم.
سپس چک می‌کنیم اگر ظرفیت بزرگتر از صفر بود، عنصر را در صف قرار می‌دهیم. همچنین مستری را به لیست مشتری ها اضافه می کنیم و شماره میز مورد نظر را به آن می‌دهیم.
اینجا یک پیج فالت رخ می‌دهد.
اگر جای خالی وجود نداشت عنصر سر صف را پاپ می‌کنیم. سپس از لیست مشتری ها میز مربوط به مشتری بیرون انداخته شده از رستوران را پیدا کرده و آن را به مشتری جدید می‌دهیم. مشتری جدید را به لیست مشتری ها اضافه می‌کنیم و شماره مشتری جدید(data) را به صفمان اضافه می‌کنیم. ایمجا نیز یک پیج فالت رخ می‌دهد.

```
while(iterator.hasNext()) {
    int data = iterator.next();
    if(queue.contains(data))
        requests.poll();
    else if(capacity>0){
        pageFaults++;
        queue.add(data);
        tables.add(new Customer(data, counter));
        counter++;
        capacity--;
        requests.poll();
    } else {
        pageFaults++;
        int page = queue.poll();
        int freeTable = 0;
        int removeIndex = 0;
        for (int i = 0; i < tables.size(); i++) {
            if(tables.get(i).name == page){
                freeTable = tables.get(i).table;
                removeIndex = i;
                break;
            }
        }
        tables.remove(removeIndex);
        tables.add(new Customer(data, freeTable));
        requests.poll();
        queue.add(data);
    }
}
```

الگوریتم LRU:

map: شماره مشتری ها و مدت زمان بین شروع کار رستوران تا درخواست آنها را در سک مپ نگه می‌داریم. کلید مپ شماره مشتری و مقدار آن مدت زمان گذشته پس از شروع کار رستوران تا آخرین درخواست آن مشتری است. وس هرچه عدد بزرگتر باشد، این مشتری در زمان نزدیکتری درخواست داده است.

```
import java.util.concurrent.TimeUnit;
public class LRU implements Runnable {
    public int new_data;
    public HashMap<Integer, Long> map = new HashMap<Integer, Long>();
    public Queue<Integer> requests = new ConcurrentLinkedQueue<Integer>();
    public LinkedList<Customer> tables = new LinkedList<Customer>();
    public int capacity;
    public int pageFaults=0;
    public int counter=1;

    public LRU(int capacity) {
        this.capacity = capacity;
    }
```

ابتدا start_time را محاسبه کردیم که زمان شروع کار رستوران(الگوریتم) را محاسبه کردیم.

در پیاده سازی متد run، با استفاده از یک حلقه(while(true، همواره چک می‌کنیم که اگر ریکوستی آمد به آن جواب دهیم. هر درخواستی که می‌آید ما زمان سیستم را دوباره چک می‌کنیم تا فاصله بین زمان شروع کار رستوران و زمان آمدن درخواست را اندازه گیری کنیم.

اول چک می‌کنیم که آیا مپ ما دارای عنصر مورد نظر هست یا خیر. اگر بود فقط این ریکوئست را از صف درخواست ها پاپ می‌کنیم و زمان درخواست مشتری را اپدیت می‌کنیم.

سپس چک می‌کنیم اگر ظرفیت بزرگتر از صفر بود، عنصر را در مپ قرار می‌دهیم

همچنین مشتری را به لیست مشتری ها اضافه می کنیم و شماره میز مورد نظر را به آن می‌دهیم.

اینجا یک پیج فالت رخ می‌دهد.

```java
@Override
public void run() {
    long startTime = System.currentTimeMillis();
    Iterator<Integer> iterator = requests.iterator();
    System.out.println(capacity);
    while(true){
        iterator = requests.iterator();
        while(iterator.hasNext()) {
            long endTime = System.currentTimeMillis();
            int data = iterator.next();
            if(map.containsKey(data)){
                map.put(data,endTime-startTime);
                requests.poll();
            }
            else if(capacity>0){
                pageFaults++;
                map.put(data,endTime-startTime);
                tables.add(new Customer(data, counter));
                counter++;
                capacity--;
                requests.poll();
            } else {
```

اگر به اندازه کافی میز خالی نداشتیم، باید یکی را از رستوران بیرون کنیم. کسی که آخرین درخواست آن دورتر بوده را باید بیرون کنیم. پس بین مقدار مپ ها باید آن کلیدی که کمترین مقدار را دارد پیدا کنیم، سپس از لیست مشتری ها نیز آن را پیدا کنیم و میزش را به مشتری جدید بدهیم. در این مرحله نیز بدیهتا پیج فالت رخ میدهد.

```
} else {
    Long min=9999999999999999L;
    int min_data = 0;
    for (Map.Entry<Integer,Long> entry : map.entrySet()){
        System.out.println(entry.getValue());
        if(entry.getValue()<=min){
            min_data = entry.getKey();
            min = entry.getValue();
        }
    }
    int page = min_data;
    int freeTable = 0;
    int removeIndex = 0;
    for (int i = 0; i < tables.size(); i++) {
        if(tables.get(i).name == page){
            freeTable = tables.get(i).table;
            removeIndex = i;
            break;
        }
    }
    map.remove(min_data);
    pageFaults++;
    map.put(data,endTime-startTime);\
    tables.remove(removeIndex);
    tables.add(new Customer(data, freeTable));
    requests.poll();
```

الگوریتم SecondChance:

list: صفی برای نگهداری ترتیب ورود و خروج صفحه ها است. الگوریتم روی این صف پیاده سازی می‌شود.

```
public int new_data;
public LinkedList<Page> list = new LinkedList<Page>();
public Queue<Integer> requests = new ConcurrentLinkedQueue<Integer>();
public LinkedList<Customer> tables = new LinkedList<Customer>();
public int capacity;
public int pageFaults =0 ;
public int counter = 1;

public secondChance(int capacity) {
    this.capacity = capacity;
}
```

صفحه موجودیتی است که دارای نام مشتری و اینکه آیا به آن رفرنسی داده شده است یا نه می‌باشد.

```
public class Page{
    int name;
    boolean used;
    public Page(int name, boolean used){
        this.name=name;
        this.used = used;
    }
}
```

در پیاده سازی متد run، با استفاده از یک حلقه(while(true، همواره چک می‌کنیم که اگر ریکوستی آمد به آن جواب دهیم. با آمدن درخواست چک می‌کنیم که آیا در لیستمان پیجی با این شماره وجود دارد یا خیر. اگر داشت used آن را true می‌کنیم.
اگر مشتری را در رستوران نداشتیم و ظرفیت خالی در رستوران داشتیم،، مشتری را در صف(رستوران) قرار می‌دهیم. همچنین مشتری را به لیست مشتری‌ها اضافه می‌کنیم و شماره میز مورد نظر را به آن می‌دهیم.
اینجا یک پیج فالت رخ می‌دهد.

```java
@Override
public void run() {
    Iterator<Integer> iterator = requests.iterator();
    while(true){
        iterator = requests.iterator();
        while(iterator.hasNext()) {
            int data = iterator.next();
            boolean in = false;
            for (Page p : list) {
                if(p.name == data){
                    p.used = true;
                    in = true;
                    requests.poll();
                    break;
                }
            }
            if(in)
                break;
            else if(capacity>0){
                list.add(new Page(data, false));
                pageFaults++;
                tables.add(new Customer(data, counter));
                counter++;
                capacity--;
                requests.poll();
```

اگر میز خالی نداشتیم یعنی باید یک مشتری را بیرون بیندازیم. در لیستمان به پیج ها از سر صف نگاه می‌کنیم. اگر used آنها true بود آن را false می‌کنیم و به آنها شانس دوباره می‌دهیم و به صف goToTail اضافه می‌کنیم که بعدا آن ها را به انتهای صف بفرستیم. زمانی به پیجی رسیدیم که used آن false بود، نام آن مشتری را از لیست مشتری ها پیدا می‌کنیم و شماره میز آن را ذخیره می‌کنیم. همچنین ایندکس آنرا در لیست اصلی مشخص می‌کنیم.

```
else {
  int pageName = 0;
  int freeTable = 0;
  int removeIndexTable = 0;
  int removeIndexList =0;
  boolean flag = true;
  int i = 0;
  LinkedList<Page> goToTail = new LinkedList<Page>();
  while (flag){
      i = i % (list.size());
      if(list.get(i).used == true){
          list.get(i).used = false;
          goToTail.add(list.get(i));
      } else {
          pageName = list.get(i).name;
          for (int j = 0; j < tables.size(); j++) {
              if(tables.get(j).name == pageName){
                  freeTable = tables.get(j).table;
                  removeIndexTable = j;
                  break;
              }
          }
          tables.remove(removeIndexTable);
          tables.add(new Customer(data, freeTable));
          removeIndexList =i;
          flag = false;
          break;
      }
      i++;
  }
```

سپس مشتری قربانی را از لیست بیرون میکنیم و آن پیج هایی که درgoToTail بودند را به آخر صفمان اضافه می‌کنیم.

```
//remove from list
list.remove(removeIndexList);
for (int k=0;k<goToTail.size();k++) {
    list.remove(goToTail.get(k));
    list.add(goToTail.get(k));
}
//add new data
list.add(new Page(data, false));
pageFaults++;
```