操作系统(D)

第 6 次作业

李子龙 518070910095

2021 年 3 月 22 日

6.8 Race conditions are possible in many computer systems. Consider an online auction system where the current highest bid for each item must be maintained. A person who wishes to bid on an item calls the bid(amount) function, which compares the amount being bid to the current highest bid. If the amount exceeds the current highest bid, the highest bid is set to the new amount. This is illustrated below:

```
void bid(double amount){
    if (amount > highestBid)
        highestBid = amount;
}
```

Describe how a race condition is possible in this situation and what might be done to prevent the race condition from occurring.

解. 因为 `highestBid` 是共享变量，一旦有两个出价不同但都高于原有价格的人并发地执行比较程序，出价较低的那一方可能会成为最高的那个：由于两者均比原有的 `highestBid` 高，导致两者均进入临界区的修改部分，高价的那方刚修改完 `highestBid`，低价的那方就进行了覆盖，从而导致 `highestBid` 被赋予了低价方，程序运行出现了错误。

解决这个问题只需要添加一个互斥锁即可：

```
int available = 1;

void aquire(){
    while(!available)
        ;   /* busy wait */
    available = 0;
}

void release(){
    available = 1;
}

void bid(double amount){
    aquire();
    if (amount > highestBid)
        highestBid = amount;
    release();
}
```

6.13 The first known correct software solution to the critical-section problem for two processes was developed by Dekker. The two processes, P_0 and P_1, share the following variables:

```
boolean flag[2]; /* initially false */
int turn;
```

The structure of process P_i(i == 0 or 1) is shown in Figure 6.18. The other process is P_j(j == 1 or 0). Prove that the algorithm satisfies all three requirements for the critical-section problem.

```
while (true){
    flag[i] = true;
    while (flag[j]){
        if (turn == j){
            flag[i] = false;
            while (turn == j); /* do nothing */
            flag[i] = true;
        }
    }

        /* critical section */

    turn = j;
    flag[i] = false;

        /* remainder section*/
}
```

解.

互斥 假如 P_i 在临界区执行，那么 `flag[j]==false`，`flag[i]==true`，`turn==i`，如果此时 P_j 想要进入临界区执行，修改 `flag[j]=true` 后会被卡在第 6 行的 `while` 循环中。满足互斥条件。

进步 第 13 行决定了下一个进程应该是另一个进程运行临界区，这样卡在第 6 行的另一个进程就会继续运行。

有限等待 P_i 在 P_j 进入临界区后最多一次就能进入临界区（`turn == i`）。

6.21 A multithreaded web server wishes to keep track of the number of requests it services (known as *hits*). Consider the two following strategies to prevent a race condition on the variable `hits`. The first strategy is to use a basic mutex lock when updating `hits`:

```
int hits;
mutex_lock hitlock;
hit_lock.acquire();
hits++;
hit_lock.release();
```

A second strategy is to use an atomic integer:

```
atomic_t hits;
atomic_inc(&hits);
```

Explain which of these two strategies is more efficient.

解. 第二种使用原子整数的方法更高效。因为第一种使用互斥锁也就是自旋锁，当锁不可用时需要不停地旋转，忙等待会消耗CPU资源，这对于服务器的资源是一种浪费。而原子整数并没有加锁机制的开销，对于这种整型变量更新特别有效。