بسم الله الرحمن الرحیم

راه‌حل تمرین‌های سیستم عامل

فصل ۵ – همگام سازی

۱- سه فرآیند همزمان به صورت زیر در حال اجرا هستند. در این فرآیندها از سه سمافور باینری استفاده شده است که مقادیر اولیه آنها به ترتیب عبارتند از : S0=1, S1=0, S2=0. در این حالت ، پردازه P0 چند بار مقدار 0 را چاپ می‌کند؟

```
Proccess P0
while(true){
    wait(S0);
     print '0';
     release(S1);
     release(S2);
}
```

```
Proccess P1
wait(S1);
release(S0);
```

```
Proccess P2
wait(S2);
release(S0);
```

۲ تا سه بار

P0 → rel(s1) → P1 → P0 → P2 (3 times)
P0 → P1 → P2 → P0 (2 times)

---

۲. راجع به امکان فراخوانی یک مانیتور دیگر داخل یک مانیتور بحث کنید.
با توجه به اینکه فراخوانی قفل و انتظار در شرط قفل مانیتور دوم زمانی است که ریسمان قفل اول را در اختیار دارد ممکن است این شرایط منجر به بن‌بست شود

۳. پیاده سازی توابع wait و signal سمافور را در مدل busy waiting با دستورهای test&set و compareAndSwap بنویسید.

```
int initial_value;    //semaphor initial value
int sem;
int done = 0;
int lock = 0;

void Wait() {
    int done = 0;
    do {
        while (testAndSet(lock));
        if (sem > 0) { sem--, done = 1; }
        lock = 0;
    } while (!done);
}

void signal() {
    while (testAndSet(lock));
    sem++;
    if (sem > initial_value)
        sem = initial_value;
    lock = 0;
}

//With CAS instruction

void wait() {
    int val;
    do {
        val = sem;
    } while (val == 0 || !CAS(sem, val, val - 1));
}

void signal() {
    int val;
    do {
        sem;
    } while (val < initial_value || !CAS(sem, val, val + 1));
}
```

۲. سیستمی شامل n فرایند را در نظر بگیرید. یک برنامه با مکانیزم مانیتور بنویسید که سه عدد چاپگر خطی را به سه فرایند تخصیص دهد.
۱-۲- این بار برنامه را برای حالتی بنویسید که هر فرآیند یک عدد اولویت منحصر به فرد داشته باشد. از عدد اولویت برای تصمیم‌گیری ترتیب تخصیص استفاده نمایید.

پاسخ قسمت اول

```
int available = 3;
Lock lock;
Condition cond=lock.newCondition();

int request_printer(int processId) {
    int printerId;

    lock.lock();

    while (available == 0)
        cond.wait();

    available--;
    printerId = free_printer_id();

    lock.unlock();
    return printerId;
}

void release_printer(int printerId) {
    lock.lock();
    set_printer_free(printerId);
    available++;
    cond.signal();
    lock.unlock();
}
```

قسمت دوم – راهکار مدیر مرکزی

در این راه حل فرض میشود که درخواستها به یک ریسمان داور که مثل یک داور مرکزی رفتار می‌کند ارسال میشود. این داور درخواستها را در صف گذاشته و به ترتیب اولویتشان به آنها پرینتر اختصاص می‌دهد. هدف اصلی سوال این روش **نبوده** و در امتحانها هم از نوشتن پاسخ با این شکل اجتناب کنید.

```
PriorityQueue q;
int available = 3;


void printer_arbiter(Thread t) {
    lock.lock();

    if (available > 1) {
        available--;
        assign_free_printer(t);
    }
    else {
        q.enqueue(t);
        put_thread_sleep(t);
    }

    lock.unlock();
}

void printer_release(Thread t) {
    lock.lock();

    if (available == 0 & q.size() > 0) {
        t = q.deque();
        wake_up_thread(t)
    }
    else
        available++;

    lock.unlock();
}
```

قسمت دوم – راهکار توزیع شده ۱

در راهکارهای توزیع شده ریسمانها بدون وجود یک ریسمان مرکزی برای داوری بین آنها به شکل همکارانه پرینترها را دریافت و استفاده می‌کنند.

در راهکار اول تمامی ریسمانها مستقل از تقدمشان بیدار شده و سپس تمامی آنها به غیر از ریسمانی که بالاترین تقدم را دارد دوباره به حالت انتظار می‌روند.

```
int request_printer_dist1(Thread t) {

      int printerId;

      lock.lock()

      if (avail == 0) {
          q.enqueue(t)
          while (avail == 0 && q.head().Id != t.Id)
              cond.wait();
      }
      available--;
      printerId = free_printer_id();
      lock.unlock();

      return printerId;
}

void release_printer_dist1(int procId) {
    lock.lock();
    available++;
    //resume all waiting threads
    cond.signalAll();
    lock.unlock();
}
```

قسمت دوم – راهکار توزیع شده ۲

این راهکار مشابه قبلی است با این تفاوت که به جای بیدار کردن همه ریسمانها فقط ریسمانی که دارای بالاترین تقدم است بیدار می‌شود.

```
const int N;
Lock lock;
Thread thread_map[N];
Condition cond[N];
Queue queue;

int request_printer_dist2(Thread t) {
    int printerId;

    lock.lock();
    if (queue.size() == N)
        return -1;

    if (available == 0) {
        queue.enqueue(t);

        // put thread in any empty cell of thread_map array
        int index = find_empty_cell(thread_map)
        thread_map[index] = t;

        //wait for a specific condition
        while (available == 0)
            cond[index].wait();

        thread_map[index] = NULL;
    }

    available--;
    printerId = free_printer_id();
    lock.unlock();

    return printerId;
}

void release_printer_dist2() {
    lock.lock();

    available++;

    if(queue.size() > 0) {
        // get the thread with the highest priority
        Thread t = queue.dequeue();
        int index = find_in_array(thread_map, t);

        // signal only one thread with the highest priority
        cond[index].signal();
    }
    lock.unlock();
}
```

Bridge with one car for each side – Semaphore solution

```
upper_line Semaphor(1, 1)
lower_line Semaphor(1, 1)

Car(direction)
   if(direction == RTL)
      wait(upper_line)
      //pass over bridge
      signal(upper_line)
   else
      wait(lower_line)
      //pass over bridge
      signal(lower_line)
```

```
Truck()
   wait(upper_line)
   wait(lower_line)

   //pass over bridge

   signal(upper_line)
   signal(lower_line)
```

Bridge with one car for each side – Monitor solution

```
Lock right_lock, left_lock;

Car(direction)
  if(direction == RTL)
    right_lock.lock()

    //pass over bridge

    right_lock.unlock()

  else
    left_lock.lock()

    //pass over bridge

    left_lock.unlock()
```

```
Truck()
  right_lock.lock()
  left_lock.lock()

  //pass over bridge

  right_lock.unlock()
  left_lock.unlock()
```

۷- روی پل برای عبور n ماشین جا وجود دارد. این بسیار مشابه مساله readers-writers است و پاسخ هم در سبک پاسخ همان مساله نوشته شده است. در راهکار مانیتور از مانیتورهای تودرتو استفاده شده است. راهکار ارائه شده به مشکل بن بست نمیخورد چرا که هیچگاه انتظارهای چرخشی بین ریسمانها نیست (ماشینهای مسیر راست-به-چپ هیچ منبع مشترکی با ماشینهای چپ به راست ندارند)

Bridge with N car for each side – Semaphore solution

```
Car(direction)
   if(direction == RTL)
      wait(rfull) //wait if line if full

      wait(rmutex)
      if(rcount == 0)
         wait(rempty)

      rcount++
      signal(rmutex)

      pass_over_bridge()

      signal(rfull)

      wait(rmutex)
      rcount--
      if(rcount == 0)
        signal(rempty)  //give chance to truck

      signal(rmutex)
   else
      // same code as right for left
```

```
rmutex,lmutex Semaphor(1,1)
rempty,lempty Semaphor(1,1)
rfull , lfull Semaphore(N,N)
int rcount = lcount = 0

Truck()
   wait(rempty)
   wait(lempty)

   pass_over_bridge()

   signal(rempty)
   signal(lempty)
```

Bridge with N car for each side – Monitor solution

```
Car(direction)
  if(direction == RTL)
      rlock.lock()
      while(rcount == N)
         rfull.await()

      rcount++
      rlock.unlock()

      pass_over_bridge()

      rlock.lock()
      rcount--
      rfull.signal()
      if(rcount == 0)
         rempty.signal()
      rlock.unlock()
else
      // the same code for left side
```

```
Lock rlock, llock;
int rcount=0, lcount=0;
Condition rfull,lfull;
Condition rempty, lempty;


Truck ()
   rlock.lock()
   while(rcount>0)
     rempty.await()

   llock.lock()
   while(lcount>0)
     lempty.await()

   pass_over_bridge()

   rlock.unlock()
   llock.unlock()
```

۸- با ورود کامیون ماشینهای جدید حق ورود ندارند

پیاده‌سازی سمافور دقیقا از راه‌حل عادلانه برای مساله readers-writers گرفته شده است

Semaphore solution

```
Car(direction)
   if(direction == RTL)
      wait(rfull) //wait if line if full

      rservice.wait()
      rmutex.wait()
      if(rcount == 0)
        raccess.wait()
      rcount++
      rmutex.signal()

      pass_over_bridge()

      rmutex.wait()
      rcount--
      if(rcount == 0)
        raccess.signal()
      rmutex.signal()
   else
      // same code as right for left
```

```
rmutex,lmutex     Semaphor(1,1)
rservice,lservice Semaphor(1,1)
rfull , lfull Semaphore(N,N)
int rcount = lcount = 0
Condition noTruck;

Truck()
    rservice_mutex.wait()
    raccess.wait()
    rservice_mutex.signal()

    lservice_mutex.wait()
    laccess.wait()
    lservice_mutex.signal()

    pass_over_bridge()

    signal(raccess)
    signal(laccess)
```

Bridge with N car for each side – Monitor solution

```
Car(direction)
  if(direction == RTL)
    comLock.lock()
    while(truck == 1)
      noTruck.await()
    comLock.unlock()

    rlock.lock()
    while(rcount == N)
      rfull.await()

    rcount++
    rlock.unlock()

    pass_over_bridge()

    rlock.lock()
    rcount--
    rfull.signal()
    if(rcount == 0)
      rempty.signal()
    rlock.unlock()
else
    // the same code for left side
```

```
int rcount=0, lcount=0
Lock rlock, llock
Condition rfull,lfull
Condition rempty, lempty
int truck = 0
Lock commLock


Truck ()
   comLock.lock()
   truck=1
   comLock.unlock()

   rlock.lock()
   while(rcount > 0)
     rempty.await()

   llock.lock()
   while(lcount > 0)
     lempty.await()

   pass_over_bridge()

   rlock.unlock()
   llock.unlock()

   comLock.lock()
   truck=0
   noTruck.signalAll()
   comLock.unlock()
```