بخش a

برای این که دو تولید کننده و مصرف کننده به صورت همزمان به بافر دسترسی پیدا نکنند mutex را به کار میگیریم و برای این که مصرف کننده از بافر خالی استفاده نکند هم از سمافور full استفاده میکنیم که تنها فقط قبل از کار های مصرف کننده wait میشود و بعد از تولید signal میخورد.

```
semaphore full = 0 , mutex = 1 ;
void P() {
  while (true) {
    // do somethings  for producing
    wait(mutex);
    addToBuff();
    signal(mutex);
    signal(full);
  }
}

void C() {
  while (true) {
    wait(full);
    wait(mutex);
    takeFromBuff();
    signal(mutex);
    // do somethings for consuming
  }
}
```

بخش b

سمافورهای سوال قبلی سرجای خود باقی میماند و برای این که تولید کننده برای بافر پر دیگر چیزی تولید نکند یک سمافور جدید میسازیم (مشابه همان چیزی که در سر کلاس بحث شد) به نام empty که تعداد خانه های خالی مقدار آن است و دقیقا برعکس سمافور full کار میکند.

```
semaphor full = 0, empty = n, b = 1 ;
void P() {
  while (true) {
    // do somthings for producing
    wait (empty)
    wait(mutex);
    addToBuff();
    signal(mutex);
    signal(full);
```

```
  }
}

void C() {
  while (true) {
    wait(full);
    wait(mutex);
    takeFromBuff();
    signal(mutex);
    signal(empty);
    // do somthings for consuming
  }
}
```

بخش c

هر نویسنده ای که بخواهد وارد شود، چنانچه اولین نویسنده ای باشد که میخواهد وارد شود؛ دیگر به خواننده ای اجازه نمیدهد که وارد شود؛ و بعد از آخرین خواننده خود شروع به کار میکند. و هنگام خروج بررسی میشود که آیا آخرین نویسنده است یا خیر؛ اگر آخرین نویسنده باشد یک سیگنال برای خوانندگان زده میشود که بتوانند وارد شوند. وخوانندگان چنانچه بخواهند وارد شوند وقتی نویسنده ای نباشد وارد میشوند و اگر وارد شدند اجازه ورود را از نویسندگان میگیرند.

```
semaphore mutex1 = 1 , mutex2 = 1 , mutex3 = 1 , w = 1 , R = 1;

int w_counter = 0 ,r_counter = 0;
void writer() {
  while (true) {
    wait(mutex2);
    w_counter++;
    if(w_count == 1)
      wait (R);
    signal(mutex2);
    wait(w);
    //do somethings to write
    signal(w);
    wait(mutex2);
    w_count--;
    if(w_count == 0)
      signal (R);
    signal(mutex2);
  }
}
```

```
void reader() {
  while (true) {
    wait(mutex3);
    wait(R);
    wait(mutex1);
    r_count++;
    if(r_count == 1)
      wait (w);
    signal(mutex1);
    signal(R);
    signal(mutex3);
    //do somethings to read
    wait(mutex1);
    r_count--;
    if(r_count == 0)
      signal (w);
    signal(mutex1);
  }
}
```

بخش d

پشت خوانندگان wait و signal روی سمافور queue میگذاریم که پشت صف نویسندگان قرار بگیرند. در واقع ایده این مسئله همان ایده اولویت مطلق خوانندگان است.

```
semaphore  queue = 1, w = 1 , m = 1;
int r_counter = 0;
void reader() {
  while(true){
    wait(queue);
    signal(queue);
    wait(m);
    if(r_counter == 1)
      wait(w);
    signal(m);
    // do somethings for reading
    wait(m);
    r_counter--;
    if(r_counter == 0)
      signal(w);
    signal(m);
  }
```

```
}
void writer() {
  while(true){
    wait(queue);
    wait(w) ;
    //do somethings for writing
    signal(w);
    signal(queue);
  }
}
```

بخش e

از mutex برای مراقبت از متغیر inqueue استفاده میکنیم.و خود متغیر inqueue برای تعداد افراد در صف است. سمافور server وقتی که یک باشد یعنی آرایشگر مشغول است و اگر صفر باشد یعنی آزاد است.

```
semaphore customers = 1 , server = 0 , mutex = 1;
int inqueue = 0;
const chairs = 10;
void barber(){
  while (true) {
    wait(customers);
    wait(mutex);
    inqueue--;
    signal(server);
    signal(mutex);
    // he cuts hair
  }
}

void customer(){
  wait(mutex);
  if(inqueue < chairs) {
    inqueue++;
    signal(customers);
    signal(mutex);
    wait(server);
    //he gets his hair cut
  }else{
    signal(mutex);
  }
}
```