

### ECES 338 Assignment #4 Solutions

- (1) Write a monitor for the **Readers-Writers Problem** where WRITERS STARVE in the sense that a stream of readers can arrive to “read”, and read (as long as there is already a reader reading) even when a writer has been waiting to write.

*Policy:* The signaled process is blocked until the signaling process leaves the monitor.

**type RW = monitor**

**int** numreaders; numwriters;

**bool** Busy;

**condition** r, w;

**procedure entry** open-read ( );

**begin**

numreaders++;

**if** (Busy) **then** r.wait;

**for** i = 1 **to** numreaders **do** r.signal;

//Release all blocked readers;

**end;**

**procedure entry** close-read ( );

**begin**

numreaders--;

**if** (numreaders = 0) **then** w.signal;

**end;**

**begin** numreaders=0; numwriters=0; Busy = *False* **end;**

MONITOR USE:

**Reader:**

Var RW-Instance : RW;

While *True*

{ RW-Instance.open-read ( );

READ;

RW-Instance.close-read ( );

DO-SOMETHING;

}

**procedure entry** open-write ( );

**begin**

numwriters++;

**if** (numreaders > 0 or Busy) **then** w.wait;

Busy = *True*;

**end;**

**procedure entry** close-write ( );

**begin**

numwriters--;

Busy = *False*;

**if** (numreaders > 0) **then** r.signal **else**

w.signal;

**end;**

**Writer:**

Var RW-Instance : RW;

While *True*

{ RW-Instance.open-write ( );

WRITE;

RW-Instance.close-write ( );

DO-SOMETHING;

}

(2) Write a monitor for the **Readers-Writers Problem** where READERS STARVE in the sense that a stream of writers can arrive and “write” one after another even when a reader has been waiting to read.

*Policy:* The signaled process is blocked until the signaling process leaves the monitor.

**type RW = monitor**

**int** numreaders; numwriters; BlockedReaders;

**bool** Busy;

**condition** r, w;

**procedure entry** open-read ( );

**begin**

**if** (Busy **or** numwriters > 0) **then**  
    {BlockedReaders++; r.wait};  
    numreaders++;  
    **while** (BlockedReaders > 0) {r.signal;  
BlockedReaders--};

**end;**

**procedure entry** close-read ( );

**begin**

    numreaders--;  
    **if** (numreaders = 0) **then** w.signal;

**end;**

**procedure entry** open-write ( );

**begin**

    numwriters++;  
    **if** (numreaders > 0 **or** Busy) **then** w.wait;  
    Busy = *True*;

**end;**

**procedure entry** close-write ( );

**begin**

    numwriters--;  
    Busy = *False*;  
    **if** (numwriters > 0) **then** w.signal **else**  
r.signal;

**end;**

**begin** numreaders=0; numwriters=0; BlockedReaders =0; Busy= *False* **end;**

MONITOR USE:

**Reader:**

Var RW-Instance : RW;

While *True*

{ RW-Instance.open-read ( );  
  READ;  
  RW-Instance.close-read ( );  
  DO-SOMETHING;  
}

**Writer:**

Var RW-Instance : RW;

While *True*

{ RW-Instance.open-write ( );  
  WRITE;  
  RW-Instance.close-write ( );  
  DO-SOMETHING;  
}

**(3) The Cookie Jar Problem.** A cookie jar that never becomes empty (the best type!) is being shared by two sisters, Tina and Judy, using the following (not-so-good) rule: Judy can get a cookie from the jar only after Tina (being the older sister) gets a cookie in at least two separate occasions, whereas Tina gets a cookie from the jar whenever she wants to. Treating Tina and Judy as two independently executing processes and the Cookie Jar as a shared resource, in assignment #3, you have given a conditional critical region-based solution to this problem. Solve the same problem, but this time, use only monitors.

```
type CookieJar = monitor
```

```
int TinaCount;
```

```
bool Busy;
```

```
condition tina, judy;
```

```
procedure entry Tina-OpenJar ( );
```

```
begin
```

```
    if (Busy) then tina.wait;
```

```
    Busy = True;
```

```
    TinaCount++;
```

```
end;
```

```
procedure entry Tina-CloseJar ( );
```

```
begin
```

```
    Busy = False;
```

```
    if (TinaCount > 1) then judy.signal;
```

```
end;
```

```
begin TinaCount = 0; Busy = False end;
```

```
procedure entry Judy-OpenJar ( );
```

```
begin
```

```
    if (Busy or TinaCount < 2) then judy.wait;
```

```
    Busy = True;
```

```
    TinaCount = 0;
```

```
end;
```

```
procedure entry Judy-CloseJar ( );
```

```
begin
```

```
    Busy = False;
```

```
    tina.signal;
```

```
end;
```

MONITOR USE:

**Tina:**

```
Var CookieJar-Instance : CookieJar;
```

```
While True
```

```
{ CookieJar-Instance.Tina-OpenJar ( );
```

```
  GET-COOKIE;
```

```
  CookieJar-Instance.Tina-CloseJar ( );
```

```
  EAT-COOKIE;
```

```
  DO-SOMETHING;
```

```
}
```

**Judy:**

```
Var CookieJar-Instance : CookieJar;
```

```
While True
```

```
{ CookieJar-Instance.Judy-OpenJar ( );
```

```
  GET-COOKIE;
```

```
  CookieJar-Instance.Judy-CloseJar ( );
```

```
  EAT-COOKIE;
```

```
  DO-SOMETHING;
```

```
}
```