```
/* PROCESS 1 */

void P1
{
   while (true)
   {
      /* preceding code */;
      entercritical (Ra);
      /* critical section */;
      exitcritical (Ra);
      /* following code */;
   }
}
```

```
/* PROCESS 2 */

void P2
{
   while (true)
   {
      /* preceding code */;
      entercritical (Ra);
      /* critical section */;
      exitcritical (Ra);
      /* following code */;
   }
}
```

• • •

```
/* PROCESS n */

void Pn
{
   while (true)
   {
      /* preceding code */;
      entercritical (Ra);
      /* critical section */;
      exitcritical (Ra);
      /* following code */;
   }
}
```

Figure 5.1 Illustration of Mutual Exclusion

```
/* program mutualexclusion */
const int n = /* number of processes */;
int bolt;
void P(int i)
{
   while (true)
   {
      while (!testset (bolt))
          /* do nothing */;
      /* critical section */;
      bolt = 0;
      /* remainder */
   }
}
void main()
{
   bolt = 0;
   parbegin (P(1), P(2), . . . ,P(n));

}
```

(a) Test and set instruction

```
/* program mutualexclusion */
int const n = /* number of processes**/;
int bolt;
void P(int i)
{
   int keyi = 1;
   while (true)
   {
      do exchange (keyi, bolt);
      while (keyi != 0)
      /* critical section */;
      exchange (keyi, bolt);
      /* remainder */
   }
}
void main()
{
   bolt = 0;
   parbegin (P(1), P(2), . . ., P(n));
}
```

(b) Exchange instruction

Figure 5.2 Hardware Support for Mutual Exclusion

```
semWait(s)
{
   while (!testset(s.flag))
      /* do nothing */;
   s.count--;
   if (s.count < 0)
   {
      place this process in s.queue;
      block this process (must also set s.flag to 0)
   }
   else
      s.flag = 0;
}

semSignal(s)
{
   while (!testset(s.flag))
        /* do nothing */;
   s.count++;
   if (s.count <= 0)
   {
      remove a process P from s.queue;
      place process P on ready list
   }
   s.flag = 0;
}
```

(a) Testset Instruction

```
semWait(s)
{
      inhibit interrupts;
      s.count--;
      if (s.count < 0)
      {
            place this process in s.queue;
            block this process and allow interrupts
      }
      else
         allow interrupts;
}

semSignal(s)
{
      inhibit interrupts;
      s.count++;
      if (s.count <= 0)
      {
            remove a process P from s.queue;
            place process P on ready list
      }
      allow interrupts;
}
```

(b) Interrupts

Figure 5.14   Two Possible Implementations of Semaphores

```
void reader(int i)
{
   message rmsg;
      while (true)
      {
         rmsg = i;
         send (readrequest, rmsg);
         receive (mbox[i], rmsg);
         READUNIT ();
         rmsg = i;
         send (finished, rmsg);
      }
 }
void writer(int j)
{
   message rmsg;
   while(true)
   {
      rmsg = j;
      send (writerequest, rmsg);
      receive (mbox[j], rmsg);
      WRITEUNIT ();
      rmsg = j;
      send (finished, rmsg);
   }
}
```

```
void  controller()
{
      while (true)
      {
         if (count > 0)
         {
            if (!empty (finished))
            {
               receive (finished, msg);
               count++;
            }
            else if (!empty (writerequest))
            {
               receive (writerequest, msg);
               writer_id = msg.id;
               count = count – 100;
            }
            else if (!empty (readrequest))
            {
               receive (readrequest, msg);
               count--;
               send (msg.id, "OK");
            }
         }
         if (count == 0)
         {
            send (writer_id, "OK");
            receive (finished, msg);
            count = 100;
         }
         while (count < 0)
         {
            receive (finished, msg);
            count++;
         }
      }
}
```

Figure 5.24 A Solution to the Readers/Writers Problem Using Message Passing

```
char   rs, sp;
char inbuf[80];
char outbuf[125];
void read()
{
  while (true)
  {
    READCARD (inbuf);
    for (int i=0; i < 80; i++)
    {
        rs = inbuf [i];
        RESUME squash
    }
    rs = " ";
    RESUME squash;
  }
}
void print()
{
  while (true)
  {
    for (int j = 0; j < 125; j++)
    {
        outbuf [j] = sp;
        RESUME squash
    }
    OUTPUT (outbuf);
  }
}
```

```
void squash()
{
  while (true)
  {
    if (rs != "*")
    {
        sp = rs;
        RESUME print;
    }
    else
    {
      RESUME read;
      if (rs == "*")
      {
          sp = "↑";
          RESUME print;
      }
      else
      {
        sp = "*";
        RESUME print;
        sp = rs;
        RESUME print;
      }
    }
    RESUME read;
  }
}
```

Figure 5.25   An Application of Coroutines