```
wait(s):
        repeat { nothing } until testset(s.flag);
        s.count := s.count - 1;
        if s.count < 0
            then begin
                place this process in s.queue;
                block this process (must also set s.flag to 0)
                end
            else s.flag := 0;

signal(s):
        repeat { nothing } until testset(s.flag);
        s.count := s.count + 1;
        if s.count ≤ 0
            then begin
                remove a process P from s.queue;
                place process P on ready list
                end;
        s.flag := 0;
```

(a) Testset Instruction

```
wait(s):
        inhibit interrupts;
        s.count := s.count - 1;
        if s.count < 0
            then begin
                place this process in s.queue;
                block this process and allow interrupts
                end
            else allow interrupts;

signal(s):
        inhibit interrupts;
        s.count := s.count + 1;
        if s.count ≤ 0
            then begin
                remove a process P from s.queue;
                place process P on ready list
                end;
        allow interrupts;
```

(b) Interrupts

Figure 5.18 Two Possible Implementations of Semaphores

```
var rs, sp: character;
        inbuf: array [1 . . 80] of character;
        outbuf: array [1 . . 125] of character;
procedure read;
begin
    repeat
        READCARD (inbuf);
        for i=1 to 80 do
            begin
                rs := inbuf [i];
                RESUME squash
            end;
        rs := " ";
        RESUME squash
    forever
end;
procedure print;
begin
    repeat
        for j = 1 to 125
            begin
                outbuf [j] := sp;
                RESUME squash
            end;
        OUTPUT (outbuf)
    forever
end;
```

```
procedure squash;
begin
    repeat
        if rs ≠ "*" then
            begin
                sp := rs;
                RESUME print
            end
        else begin
            RESUME read;
            if rs = "*" then
                begin
                    sp := "↑";
                    RESUME print
                end
            else begin
                sp := "*";
                RESUME print;
                sp := rs;
                RESUME print
            end
        end
        RESUME read
    forever
end.
```

Figure 5.31 An Application of Coroutines

```
program   mutualexclusion;
const  n = . . . ; (*number of processes*);
var      bolt: integer;
procedure P(i: integer);
begin
    repeat
        repeat { nothing } until testset (bolt);
        < critical section >;
        bolt := 0;
        < remainder >
    forever
end;
begin (* main program *)
    bolt := 0;
    parbegin
        P(1);
        P(2);
        . . .
        P(n)
    parend
end.
```

(a) Test and set instruction

```
program   mutualexclusion;
const  n = . . . ; (*number of processes*);
var      bolt: integer;
procedure P(i: integer);
var      key_i: integer;
begin
    repeat
        key_i := 1;
        repeat exchange (key_i, bolt) until key_i = 0;
        < critical section >;
        exchange (key_i, bolt);
        < remainder >
    forever
end;
begin (* main program *)
    bolt := 0;
    parbegin
        P(1);
        P(2);
        . . .
        P(n)
    parend
end.
```

(b) Exchange instruction

Figure 5.7 Hardware Support for Mutual Exclusion

```
procedure readeri;
    var rmsg: message;
    begin
        repeat
            rmsg := i;
            send (readrequest, rmsg);
            receive (mboxi, rmsg);
            READUNIT;
            rmsg := i;
            send (finished, rmsg)
        forever
    end;
procedure writerj;
    var rmsg: message;
    begin
        repeat
            rmsg := i;
            send (writerequest, rmsg);
            receive (mboxj, rmsg);
            WRITEUNIT;
            rmsg := i;
            send (finished, rmsg)
        forever
    end;
```

```
procedure controller;
    begin
        repeat
            if count > 0 do begin
                if not empty (finished) then begin
                    receive (finished, msg);
                    count := count + 1
                    end
                else if not empty (writerequest) then begin
                    receive (writerequest, msg);
                    writer.id := msg.id;
                    count := count - 100
                    end
                else if not empty (readrequest) then begin
                    receive (readrequest, msg);
                    count := count - 1;
                    send (msg.id, "OK")
                    end
            end;
            if count = 0 do begin
                send (writer.id, "OK");
                receive (finished, msg);
                count := 100
            end;
            while count < 0 do begin
                receive (finished, msg);
                count := count + 1
            end
        forever
    end.
```

Figure 5.30 A Solution to the Readers/Writers Problem Using Message-Passing