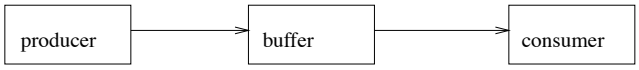
```
WHEN TRUE => alternative
```

is semantically the same as

```
alternative
```

Thus, an alternative without a guard is always open. And the basic select construct is a special case of the guarded select construct.

We take the producer-consumer problem with a bounded buffer as an example. Since the buffer is a passive object, we should represent it as a process in the message passing model. The configuration graph of the program is shown as follows:



The producer process and the consumer process are the clients processes which call the buffer process to receive and send messages respectively. They are trivial to program. We only give the solution to the buffer process below:

```
 PROCESS buffer(var put,get: channel of
integer); VAR BUFF: ARRAY[0..N] OF integer;  (*buffer size is
N+1*)
    NextIn, NextOut, CONTENTS: integer;
BEGIN
    NextIn:=0; NextOut:=0; CONTENTS:=0;
    REPEAT
       SELECT
         WHEN CONTENTS < N+1 => put ? BUFF[NextIn];
            NextIn := (NextIn+1) MOD (N+1);
            CONTENTS:=CONTENTS + 1;
       OR
         WHEN CONTENTS > 0 => get ! BUFF[NextOut];
            NextOut: NextOut:= (NextOut+1) MOD (N+1);
            CONTENTS:=CONTENTS-1
       END;
    FOREVER
END;
```

6.10 The terminate alternative

Recall that a concurrent program contains active and passive objects. With message passing, both active and passive objects are encoded as processes - generally as client processes and server processes respectively.

Active processes control their own execution and hence terminate when their internal states require them to do so. For example, the producer can stop if it does not want to produce anymore, and the consumer can stop if it does not