第24讲 管程及消息传递方法

Approaches of Mutual Exclusion

- *Software Approaches*
- Hardware Support
- Semaphores
- ***Monitors***
- Message Passing

Approaches of Mutual Exclusion

- *Software Approaches*
- Hardware Support
- Semaphores
- ***Monitors***
- Message Passing

Monitor（管程）——面向对象方法

- 用信号量实现互斥，编程容易出错（ wait 、 signal 的出现顺序和位置非常重要）
- Support at Programming-language level
- 管程是用并发 pascal 、 pascal plus 、 Modula-2 、 Modula-3 等语言编写的程序，现在已形成了许多库函数。管程可以锁定任何对象，如链表或链表的元素等。
- 用管程实现互斥比用信号量实现互斥，更简单、方便

Monitors

- Monitor is a software module，由若干过程、局部于管程的数据、初始化语句（组）组成
- Chief characteristics
  - Local data variables are accessible only by the monitor.
  - Process enters monitor by invoking one of its procedures.
  - Only one process may be executing in the monitor at a time.

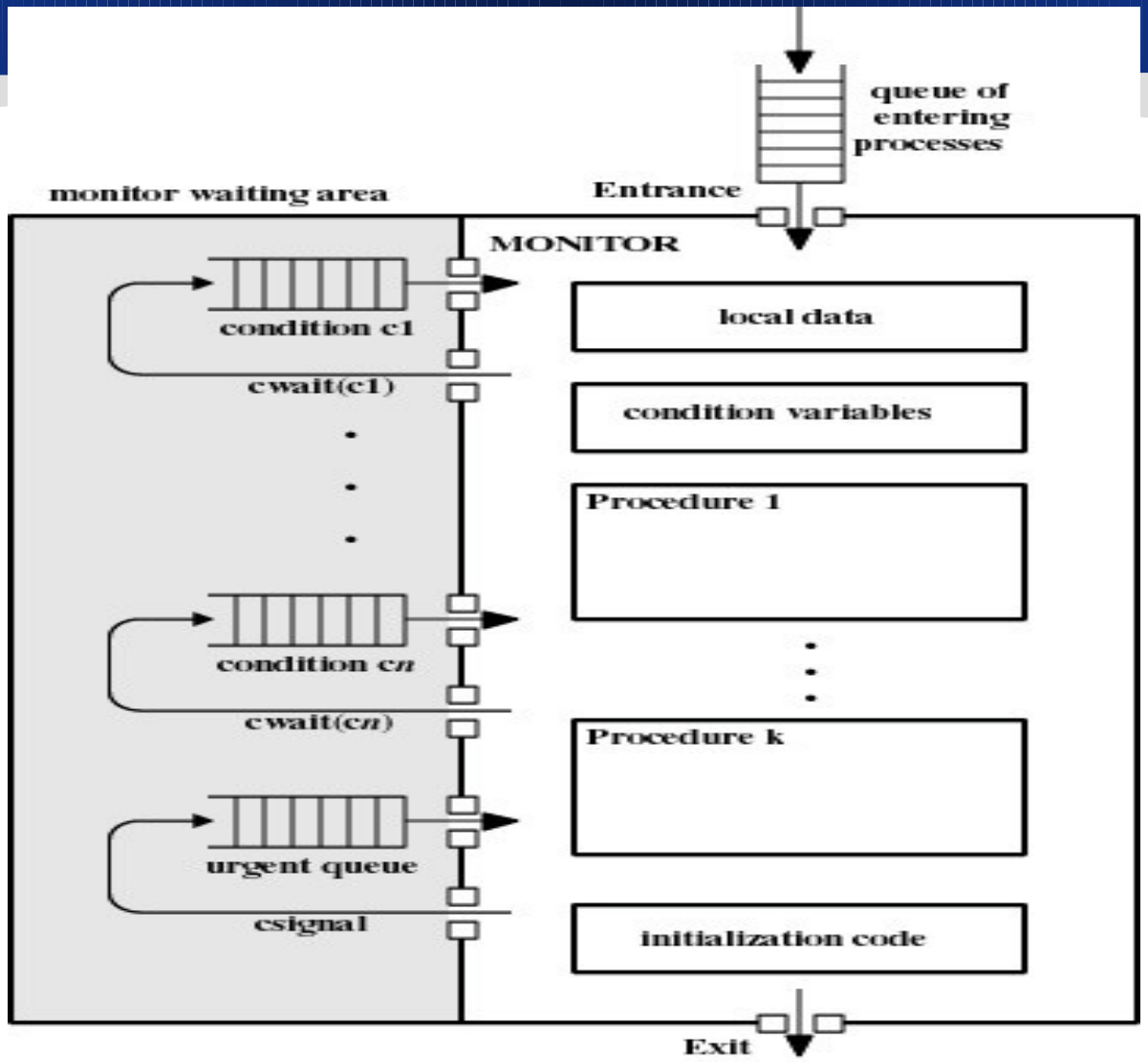

Figure 5.21 Structure of a Monitor

Approaches of Mutual Exclusion

- *Software Approaches*
- Hardware Support
- Semaphores
- Monitors
- ***Message Passing***

Message Passing

- Enforce mutual exclusion
- Exchange information

send (destination, message)

receive (source, message)

Synchronization

- Sender and receiver may or may not be blocked (waiting for message).
- Blocking send, blocking receive
  - Both sender and receiver are blocked until message is delivered.
  - Called a rendezvous(紧密同步，汇合)

Synchronization

- **Nonblocking send, blocking receive**
  - Sender continues processing such as sending messages as quickly as possible.
  - Receiver is blocked until the requested message arrives.
- **Nonblocking send, nonblocking receive**
  - Neither party is required to wait

Addressing(寻址)

- **Direct addressing**
  - **Send primitive includes a specific identifier of the destination process.**
  - **Receive primitive could know ahead of time which process a message is expected.**
  - **Receive primitive could use source parameter to return a value when the receive operation has been performed.**

Addressing

- **Indirect addressing**
  - messages are sent to a shared data structure consisting of queues.
  - queues are called **mailboxes**.
  - one process sends a message to the mailbox and the other process picks up the message from the mailbox.

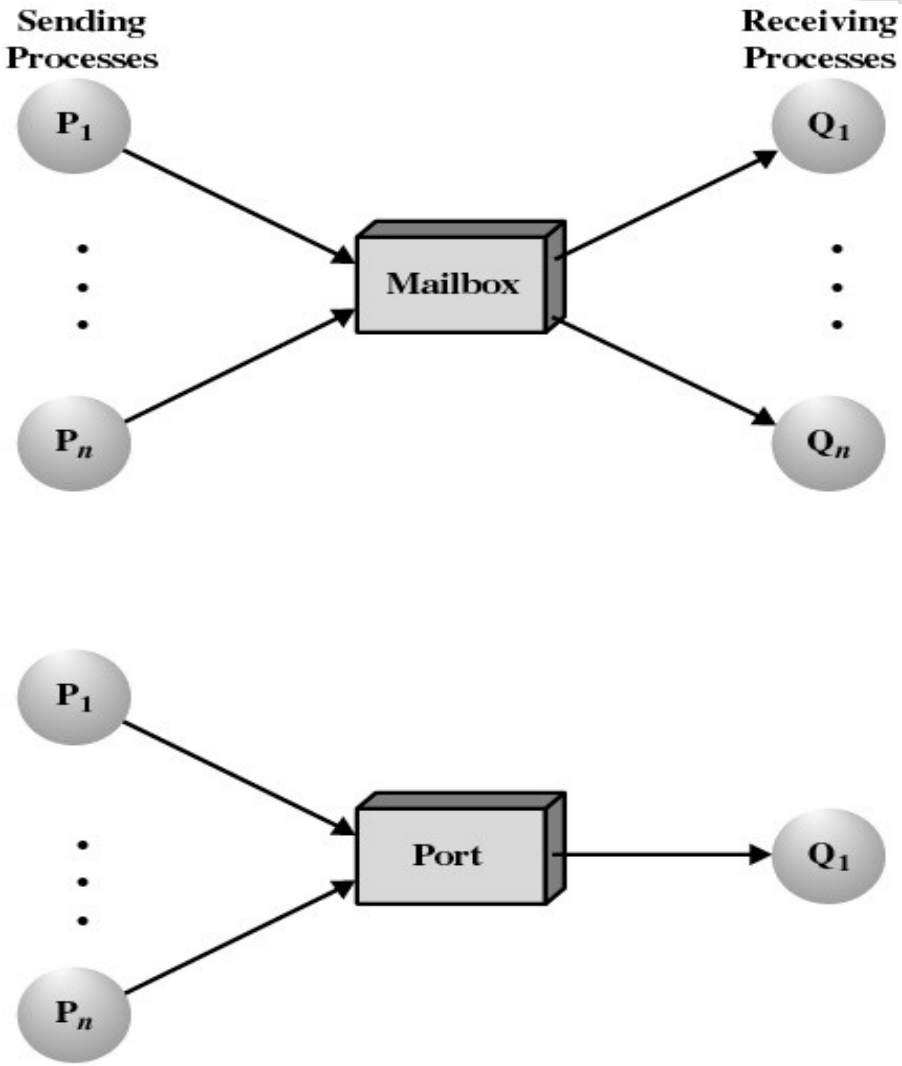

Figure 5.24 Indirect Process Communication

Message Format

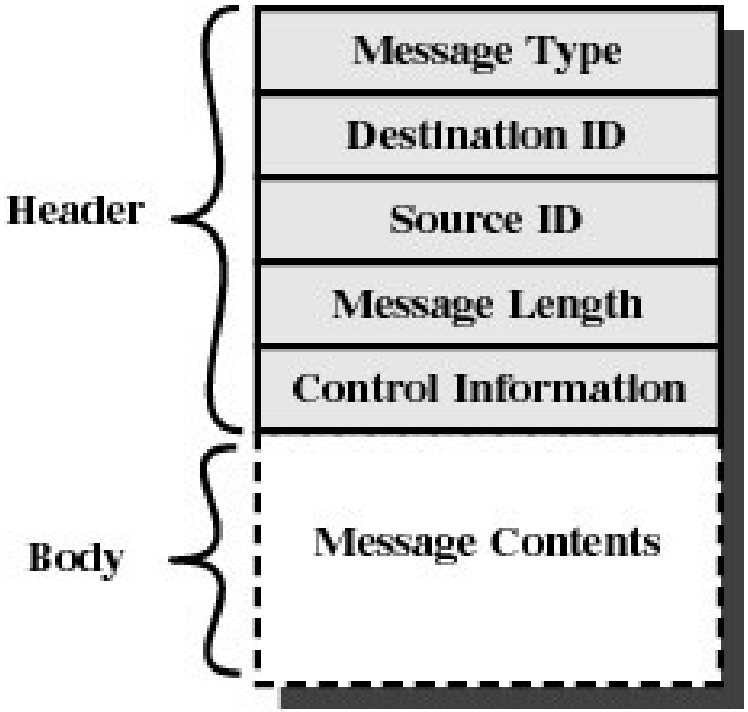

Figure 5.25 General Message Format

Message Passing (Mutual Exclusion)

- 若采用 **Nonblocking send, blocking receive**

- 多个进程共享*邮箱* ***mutex***。若进程申请进入临界区，首先申请从 mutex 邮箱中接收一条消息。若邮箱空，则该进程阻塞；若进程收到邮箱中的消息，则进入临界区，执行完毕退出，并将该消息放回邮箱 mutex。该消息 as a ***token*** 在进程间传递。

利用消息传递实现互斥的通用模式

```
program mutualexclusion;
const n=…;   /*   进程数      */
procedure P(i:integer);
var msg:message;
begin
  repeat
      receive(mutex,msg);     /*  从邮箱接收一条消息      */
    <临界区>;
     send(mutex,msg);        /*     将消息发回到邮箱     */
    <其余部分>
  forever
end;
```

```
begin    /*  主程序   */
           create_mailbox(mutex);    /*  创建邮箱   */
             send(mutex,null);          /*  初始化，向邮箱发送一条空消息   */
            parbegin
                 P(1);
                 P(2);
                       …
                 P(n)
            parend
```

Message Passing（Producer/Consumer Problem)

- 解决有限 buffer Producer/Consumer Problem
- 设两个邮箱：
  - *Mayconsume*： Producer 存放数据，供 Consumer 取走（即 buffer 数据区）
  - *Mayproduce*：存放空消息的 buffer 空间

```
program mutualexclusion;
const
capacity     =…;          /*   消息缓冲区大
小     */
    null  = …;           /*   空消息     */
procedure producer;
var pmsg:message;
begin
 while true do
  begin
     receive(mayproduce,pmsg);
     pmsg := produce;
     send(mayconsume,pmsg);
  end
end;
```

```
procedure consumer;
var cmsg:message;
begin
   while true do
   begin
     receive(mayconsume,cmsg);
      consume(cmsg);
      send(mayproduce,null);
   end
end;
```

```
begin    /*  主程序   */
  create_mailbox(mayproduce);
  create_mailbox(mayconsume);
  for i = 1 to capacity do send(mayproduce,null);
  parbegin
        producer;
       consumer;
  parend
end
```