Concurrency: Mutual Exclusion and Synchronization

Chapter 5

1

Difficulties with Concurrency

- Sharing global resources
- Management of allocation of resources
- Programming errors difficult to locate

4

Currency

- Communication among processes
- Sharing resources
- Synchronization of multiple processes
- Allocation of processor time

2

A Simple Example

```
void echo()
{
  chin = getchar();
  chout = chin;
  putchar(chout);
}
```

5

Concurrency

- Multiple applications
  - Multiprogramming
- Structured application
  - Application can be a set of concurrent processes
- Operating-system structure
  - Operating system is a set of processes or threads

3

A Simple Example

```
  Process P1        Process P2
.                   .
in = getchar();     .
.                   in = getchar();
chout = chin;       chout = chin;
putchar(chout);     .
.                   putchar(chout);
.                   .
```

6

Operating System Concerns

- Keep track of active processes
- Allocate and deallocate resources
  - Processor time
  - Memory
  - Files
  - I/O devices
- Protect data and resources
- Result of process must be independent of the speed of execution of other concurrent processes

7

Cooperation Among Processes by Sharing

- Writing must be mutually exclusive
- Critical sections are used to provide data integrity

10

Process Interaction

- Processes unaware of each other
- Processes indirectly aware of each other
- Process directly aware of each other

8

Cooperation Among Processes by Communication

- Messages are passes
  - Mutual exclusion is not a control requirement
- Possible to have deadlock
  - Each process waiting for a message from the other process
- Possible to have starvation
  - Two processes sending message to each other while another process waits for a message

11

Competition Among Processes for Resources

- Mutual Exclusion
  - Critical sections
    - Only one program at a time is allowed in its critical section
    - Example only one process at a time is allowed to send command to the printer
- Deadlock
- Starvation

9

Requirements for Mutual Exclusion

- Only one process at a time is allowed in the critical section for a resource
- A process that halts in its non-critical section must do so without interfering with other processes
- No deadlock or starvation

12

Requirements for Mutual Exclusion

- A process must not be delayed access to a critical section when there is no other process using it
- No assumptions are made about relative process speeds or number of processes
- A process remains inside its critical section for a finite time only

13

Second Attempt

- Each process can examine the other's status but cannot alter it
- When a process wants to enter the critical section is checks the other processes first
- If no other process is in the critical section, it sets its status for the critical section
- This method does not guarantee mutual exclusion
- Each process can check the flags and then proceed to enter the critical section at the same time

16

First Attempt

- Busy Waiting
  - Process is always checking to see if it can enter the critical section
  - Process can do nothing productive until it gets permission to enter its critical section

14

Third Attempt

- Set flag to enter critical section before check other processes
- If another process is in the critical section when the flag is set, the process is blocked until the other process releases the critical section
- Deadlock is possible when two processes set their flags to enter the critical section. Now each process must wait for the other process to release the critical section

17

Coroutine

- Designed to be able to pass execution control back and forth between themselves
- Inadequate to support concurrent processing

15

Fourth Attempt

- A process sets its flag to indicate its desire to enter its critical section but is prepared to reset the flag
- Other processes are checked. If they are in the critical region, the flag is reset and later set to indicate desire to enter the critical region. This is repeated until the process can enter the critical region.

18

Fourth Attempt

- It is possible for each process to set their flag, check other processes, and reset their flags. This scenario will not last very long so it is not deadlock. It is undesirable

19

Mutual Exclusion: Hardware Support

- Special Machine Instructions
  - Performed in a single instruction cycle
  - Not subject to interference from other instructions
  - Reading and writing
  - Reading and testing

22

Correct Solution

- Each process gets a turn at the critical section
- If a process wants the critical section, it sets its flag and may have to wait for its turn

20

Mutual Exclusion: Hardware Support

- Test and Set Instruction
```
boolean testset (int i) {
    if (i == 0) {
        i = 1;
        return true;
    }
    else {
        return false;
    }
}
```

23

Mutual Exclusion: Hardware Support

- Interrupt Disabling
  - A process runs until it invokes an operating-system service or until it is interrupted
  - Disabling interrupts guarantees mutual exclusion
  - Processor is limited in its ability to interleave programs
  - Multiprocessing
    - disabling interrupts on one processor will not guarantee mutual exclusion

21

Mutual Exclusion: Hardware Support

- Exchange Instruction
```
void exchange(int register,
        int memory) {
    int temp;
    temp = memory;
    memory = register;
    register = temp;
}
```

24

Mutual Exclusion Machine Instructions

- Advantages
  - Applicable to any number of processes on either a single processor or multiple processors sharing main memory
  - It is simple and therefore easy to verify
  - It can be used to support multiple critical sections

Mutual Exclusion Machine Instructions

- Disadvantages
  - Busy-waiting consumes processor time
  - Starvation is possible when a process leaves a critical section and more than one process is waiting.
  - Deadlock
    - If a low priority process has the critical region and a higher priority process needs it, the higher priority process will obtain the processor to wait for the critical region

Semaphores

- Special variable called a semaphore is used for signaling
- If a process is waiting for a signal, it is suspended until that signal is sent
- Wait and signal operations cannot be interrupted
- Queue is used to hold processes waiting on the semaphore

Semaphores

- Semaphore is a variable that has an integer value
  - May be initialized to a nonnegative number
  - Wait operation decrements the semaphore value
  - Signal operation increments semaphore value

Producer/Consumer Problem

- One or more producers are generating data and placing these in a buffer
- A single consumer is taking items out of the buffer one at time
- Only one producer or consumer may access the buffer at any one time

Producer

```
producer:
while (true) {
  /* produce item v */
  b[in] = v;
  in++;
}
```
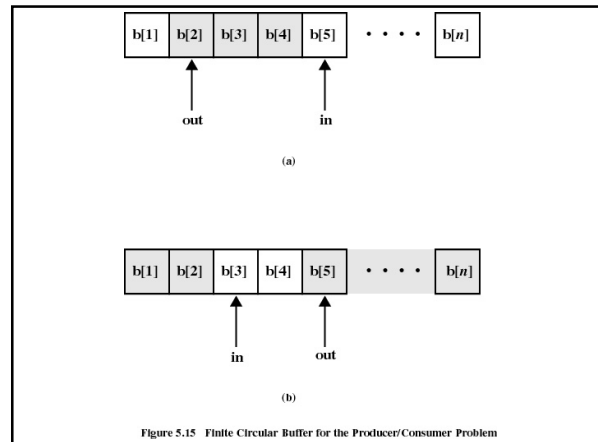
Consumer

```
consumer:
while (true) {
  while (in <= out)
     /*do  nothing */;
  w = b[out];
  out++;
  /* consume item w */
}
```

Figure 5.15 Finite Circular Buffer for the Producer/Consumer Problem

Producer with Circular Buffer

```
producer:
while (true) {
  /* produce item v */
  while ((in + 1) % n == out)  /* do
  nothing */;
  b[in] = v;
  in = (in + 1) % n
}
```

Infinite Buffer
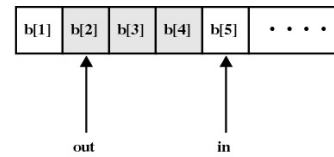

Note: shaded area indicates portion of buffer that is occupied

Figure 5.11 Infinite Buffer for the Producer/Consumer Problem

Consumer with Circular Buffer

```
consumer:
while (true) {
  while (in == out)
     /* do nothing */;
  w = b[out];
  out = (out + 1) % n;
  /* consume item w */
}
```
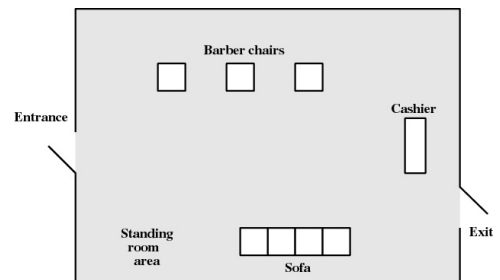
Barbershop Problem

Figure 5.18 The Barbershop

Monitors

- Monitor is a software module
- 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

Synchronization

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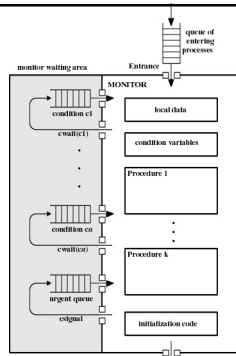 - Called a rendezvous

Figure 5.21 Structure of a Monitor

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

Message Passing

- Enforce mutual exclusion
- Exchange information

```
send (destination, message)
receive (source, message)
```

Addressing

- 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

43

Readers/Writers Problem

- Any number of readers may simultaneously read the file
- Only one writer at a time may write to the file
- If a writer is writing to the file, no reader may read it
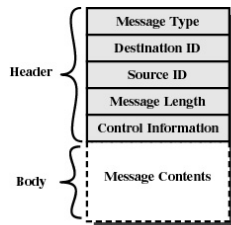
46

Figure 5.24 Indirect Process Communication
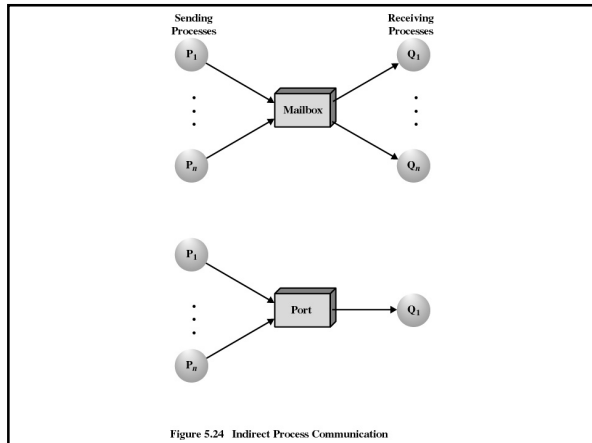
Message Format

Figure 5.25   General Message Format

45