Processes

Notice: The slides for this lecture have been largely based on those accompanying an earlier edition of the textbook Operating Systems Concepts with Java, by Silberschatz, Galvin, and Gagne. Many, if not all, the illustrations contained in this presentation come from this source.

1/29/2010 CSCI 315 Operating Systems Design
Process Concept

- Process – a program in execution; process execution must progress in sequential fashion.

- A process includes:
  - program counter,
  - stack,
  - data section.
Process Control Block (PCB)

OS bookkeeping information associated with each process:

- Process state,
- Program counter,
- CPU registers,
- CPU scheduling information,
- Memory-management information,
- Accounting information,
- I/O status information,
  :

<table>
<thead>
<tr>
<th>process id</th>
</tr>
</thead>
<tbody>
<tr>
<td>process state</td>
</tr>
<tr>
<td>program counter</td>
</tr>
<tr>
<td>registers</td>
</tr>
<tr>
<td>memory limits</td>
</tr>
<tr>
<td>list of open files</td>
</tr>
<tr>
<td>:</td>
</tr>
</tbody>
</table>

1/20/2010    CSCI 315 Operating Systems Design  3
Process State Transition Diagram

new \rightarrow\ schemer dispatch \rightarrow ready \rightarrow interrupt \rightarrow running \rightarrow exit \rightarrow terminated

I/O or event completion \rightarrow waiting \rightarrow I/O or event wait
Process Scheduling Queues

- *Job queue* – set of all processes in the system.

- *Ready queue* – set of all processes residing in main memory, ready and waiting to execute.

- *Device queues* – set of processes waiting for an I/O device.

Processes migrate between the various queues.
Processes and OS Queues

1/20/2010 CSCI 315 Operating Systems Design 6
Schedulers

- *Long-term scheduler* (or job scheduler) – selects which processes should be brought into the ready queue
- *Short-term scheduler* (or CPU scheduler) – selects which process should be executed next and allocates CPU
Schedulers

- Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slow; controls the degree of multiprogramming)
- Processes can be described as either:
  - *I/O-bound process* – spends more time doing I/O than computations, many short CPU bursts
  - *CPU-bound process* – spends more time doing computations; few very long CPU bursts
Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.

- Context-switch time is overhead; the system does no useful work while switching.

- Time dependent on hardware support.
Process Creation

- Parent process create children processes, which, in turn can create other processes, forming a tree of processes.

- Resource sharing:
  - Parent and children share all resources,
  - Children share subset of parent's resources,
  - Parent and child share no resources.

- Execution:
  - Parent and children execute concurrently,
  - Parent may wait until children terminate.
Process Creation (Cont.)

• Address space:
  – Child has duplicate of parent’s address space, or
  – Child can have a program loaded onto it.

• UNIX examples:
  – `fork` system call creates new process and returns
    with a pid (0 in child, > 0 in the parent),
  – `exec` system call can be used after a `fork` to replace
    the process’ memory space with a new program.
Process Termination

- Process executes last statement and asks the operating system to terminate it (**exit**)  
  - Output data from child to parent (via **wait**)  
  - Process' resources are deallocated by operating system
- Parent may terminate execution of children processes (**abort**) if:  
  - Child has exceeded allocated resources,  
  - Task assigned to child is no longer required,  
  - If parent is exiting (some operating system do not allow child to continue if its parent terminates)  
    - All children terminated - **cascading termination**
Cooperating Processes

• An *independent* process *cannot* affect or be affected by the execution of another process.
• A *cooperating* process *can* affect or be affected by the execution of another process.
• Advantages of process cooperation:
  – Information sharing,
  – Computation speed-up,
  – Modularity,
  – Convenience.
Producer-Consumer Problem

A paradigm for cooperating processes in which a *producer* process produces information that is consumed by a *consumer* process:
- *unbounded-buffer* places no practical limit on the size of the buffer,
- *bounded-buffer* assumes that there is a fixed buffer size.
Bounded-Buffer
(shared-memory solution)

```java
public interface Buffer {
    // producers call this method
    public abstract void insert(Object item);

    // consumers call this method
    public abstract Object remove();
}

import java.util.*;
public class BoundedBuffer implements Buffer {
    private static final int BUFFER_SIZE = 5;
    private int count; // number of items in the buffer
    private int in; // points to the next free position
    private int out; // points to the next full position
    private Object[] buffer;

    public BoundedBuffer() {
        // buffer is initially empty
        count = 0;
        in = 0;
        out = 0;
        buffer = new Object[BUFFER_SIZE];
    }

    // producers call this method
    public void insert(Object item) { // Slide 17
        if (count < BUFFER_SIZE) {
            buffer[out] = item;
           [out]
            count = count + 1;
            in = (in + 1) % BUFFER_SIZE;
        }
    }

    // consumers call this method
    public Object remove() { // Slide 18
        if (count > 0) {
            Object item = buffer[in];
            buffer[in] = null;
            [in]
            count = count - 1;
            in = (in + 1) % BUFFER_SIZE;
            return item;
        }
        return null;
    }
}
```
Bounded-Buffer
(shared-memory solution)

```java
public void insert(Object item) {
    while (count == BUFFER_SIZE); // do nothing -- no free buffers
    // add an item to the buffer
    ++count;
    buffer[in] = item;
    in = (in + 1) % BUFFER_SIZE;
}
```
Bounded-Buffer
(shared-memory solution)

```java
public Object remove() {
    Object item;
    while (count == 0); // do nothing -- nothing to consume
    // remove an item from the buffer
    --count;
    item = buffer[out];
    out = (out + 1) % BUFFER SIZE;
    return item;
}
```
Interprocess Communication (IPC)

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
  - `send(message)` – message size fixed or variable
  - `receive(message)`
- If $P$ and $Q$ wish to communicate, they need to:
  - establish a `communication link` between them
  - exchange messages via `send/receive`
- Implementation of `communication link`
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)
Implementation Questions

- How are links established?
- Can a link be associated with more than two processes?
- How many links can there be between every pair of communicating processes?
- What is the capacity of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?
Direct Communication

- Processes must name each other explicitly:
  - send \((P, \text{message})\) – send a message to process \(P\)
  - receive\((Q, \text{message})\) – receive a message from process \(Q\)

- Properties of communication link
  - Links are established automatically
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
  - The link may be unidirectional, but is usually bi-directional
Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports)
  - Each mailbox has a unique id
  - Processes can communicate only if they share a mailbox
- Properties of communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with many processes
  - Each pair of processes may share several communication links
  - Link may be unidirectional or bi-directional
Indirect Communication

- Operations:
  - create a new mailbox,
  - send and receive messages through mailbox,
  - destroy a mailbox.
- Primitives are defined as:
  \textbf{send}(A, \textit{message}) – send a message to mailbox A,
  \textbf{receive}(A, \textit{message}) – receive a message from mailbox A.
Indirect Communication

- Mailbox sharing
  - \( P_1, P_2, \) and \( P_3 \) share mailbox A
  - \( P_1 \), sends; \( P_2 \) and \( P_3 \) receive
  - Who gets the message?

- Solutions
  - Allow a link to be associated with at most two processes
  - Allow only one process at a time to execute a receive operation
  - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.
Synchronization

- *Message passing* may be either blocking or non-blocking.
- **Blocking** is considered *synchronous*:
  - *Blocking send* has the sender block until the message is received.
  - *Blocking receive* has the receiver block until a message is available.
- **Non-blocking** is considered *asynchronous*:
  - *Non-blocking send* has the sender send the message and continue.
  - *Non-blocking receive* has the receiver receive a valid message or null.
Buffering

Queue of messages attached to the link; implemented in one of three ways:

1. Zero capacity – 0 messages
   Sender must wait for receiver (rendezvous).

2. Bounded capacity – finite length of \( n \) messages. Sender must wait if link full.

3. Unbounded capacity – infinite length.
   Sender never waits.