FAA National Software Conference, June 2001
RTOS Basics

Digital Flight

FAA National Software Conference

OS Overview
Safety-Critical Implications

Presented by: Cheryl Dorsey

Digital Flight, Principal

email: dorsey@erols.com

Slide 1

Digital Flight

Topics

- Definitions
- Overview
- Common OS Components
  - -System Initialization
  - -Process Management
  - -Memory Management
  - -I/O System Management
  - -Protection
- System Services and Functions
- Processes
- Inter-process Communication
- Process Synchronization
- Memory Management
- I/O
- Safety-critical Specifics (Green Hills Software Inc.)*

Slide 2

Digital Flight

What Is An Operating System?

A program that acts as an intermediary between a user of a computer and the computer hardware.

Operating system goals:

- execute user programs and make solving user problems easier
- make the computer system convenient to use
- use the computer hardware in an efficient manner.

Slide 3

Digital Flight

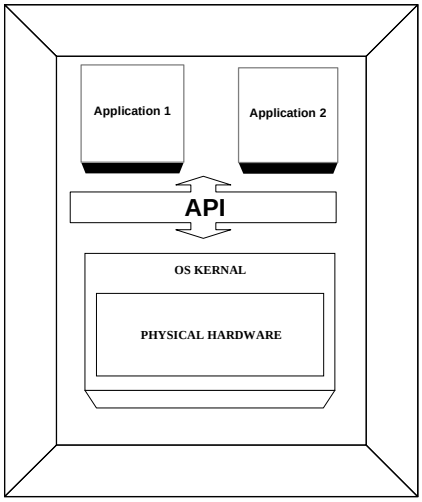
Computer System Components

1. Hardware – provides basic computing resources (CPU, memory, I/O devices).
2. Operating system – controls and coordinates the use of the hardware among the various application programs for the various users
3. Applications programs – define ways to use system resources to implement the computing needs of the users (compilers, database systems, avionics, business programs).

Slide 4

Digital Flight

Overview Of System Components



Slide 5

Digital Flight

Operating System Definitions

Resource allocator – manages and allocates resources.

Control program – controls the execution of user programs and operations of I/O devices .

Kernel – the one program running at all times (all else being application programs).

Slide 6

Digital Flight

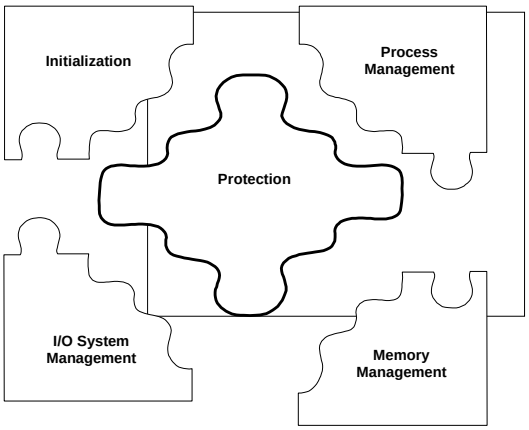
Needed OS Features For Multi -programming

- Memory management – the system must allocate the memory to several processes.
- CPU scheduling – the system must choose among several processes ready to run.
- I/O routine supplied by the system.
- Allocation of devices.

Slide 7

Digital Flight

Common OS Components



Slide 8

Digital Flight

System Initialization

- Operating systems designed to run on many machines; system must be configured for each specific target.
- Board Support Package needs specifics of the configuration of the hardware system.
- *Booting* – start a computer by loading the kernel.
- *Bootstrap program* – code stored in ROM locates kernel, loads it into memory, and execution.

Slide 9

Digital Flight

Process Management

- A *process* - program in execution. A process needs resources (CPU time, memory, files, and I/O devices) to accomplish its task.
- The operating system is responsible for the following activities in connection with process management.
  - Process creation and deletion.
  - process suspension and resumption.
  - Provision of mechanisms for:
    - process synchronization
    - process communication

Slide 10

Digital Flight

Main-Memory Management

- Memory - large array of words or bytes, each with its own address, a repository of quickly accessible data shared by CPU and I/O devices.
- Main memory is a volatile storage device. It loses its contents in the case of system failure (unless non-volatile is use).
- The operating system is responsible for the following activities in connections with memory management:
  - Keep track of which parts of memory are currently being used and by whom.
  - Decide which processes to load when memory space becomes available.
  - Allocate and de-allocate memory space as needed.

Slide 11

Digital Flight

Secondary Memory Management

- Since main memory (*primary storage*) is volatile & too small for all data and programs permanently, the computer system must provide *secondary storage* to back up main memory.
- Most modern computer systems use other types of slower memory, for both programs and data.
- The operating system is responsible for the following activities in connection with disk management:
  - Free space management
  - Storage allocation
  - Scheduling

Slide 12

Digital Flight

I/O System Management

- The I/O system consists of:
  - A buffer-caching system
  - A general device-driver interface
  - Drivers for specific hardware devices

Slide 13

Digital Flight

Protection System

- *Protection* refers to a mechanism for controlling access by programs, processes, or users to both system and user resources.
- The protection mechanism must:
  - distinguish between authorized and unauthorized usage.
  - specify the controls to be imposed.
  - provide a means of enforcement.

Slide 14

Digital Flight

Operating System Services

- Program execution – system capability to load a program into memory and to run it.
- I/O operations – user programs unable to execute I/O operations directly, OS must provides means to perform I/O.
- File-system manipulation – program capability to read, write, create, and delete files.

Slide 15

Digital Flight

Operating System Services

- Communications – exchange of information between processes executing either on the same computer or on different systems tied together by a network or bus. Implemented via *shared memory* or *message passing*.
- Error detection – ensure correct computing by detecting errors in the CPU and memory hardware, in I/O devices, or in user programs.

Slide 16

Digital Flight

Operating System Functions (Cont'd)

Additional OS functions to ensure efficient system operations:

- Resource allocation – allocating resources to multiple users or multiple jobs running at the same time.
- Accounting – keep track of and record which users use how much and what kinds of computer resources.
- Protection – ensuring that all access to system resources is controlled.

Slide 17

Digital Flight

Topics

Definitions

Overview

Common OS Components

-System Initialization

-Process Management

-Memory Management

-I/O System Management

-Protection

System Services and Functions

Processes

Inter-process Communication

Process Synchronization

Memory Management

I/O

Safety-critical Specifics (Green Hills Software Inc.)*

Slide 18

Digital Flight

Processes

- Process Concept
- Process Scheduling
- Operation on Processes
- Cooperating Processes
- Interprocess Communication

Slide 19

Digital Flight

Process Concept

- An operating system executes a variety of application programs or tasks
- Process – a program in execution; process execution progresses in sequential fashion.
- A process includes:
  - program counter
  - stack
  - data section

Slide 20

Digital Flight

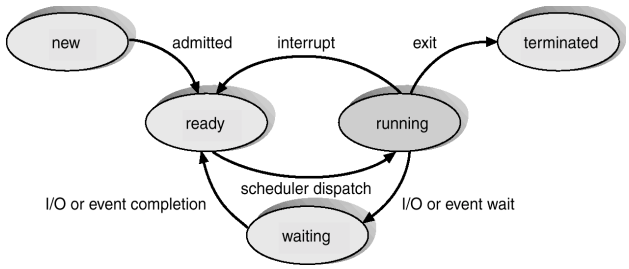
Process State

- As a process executes, it changes *state*
  - new: The process is being created.
  - running: Instructions are being executed.
  - waiting: The process is waiting for some event to occur.
  - ready: The process is waiting to be assigned.
  - terminated: The process has finished execution.

Slide 21

Digital Flight

Diagram of Process State



Slide 22

Digital Flight

Process Control Block (PCB)

Information associated with each process.

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

Slide 23

Digital Flight

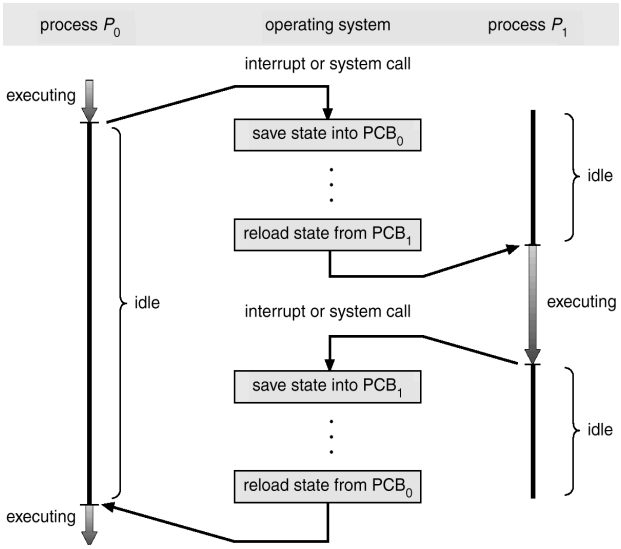
Process Control Block (PCB)

pointer	process state
process number	
program counter	
registers	
memory limits	
list of open files	
⋮	

Slide 24

Digital Flight

CPU Switch From Process to Process



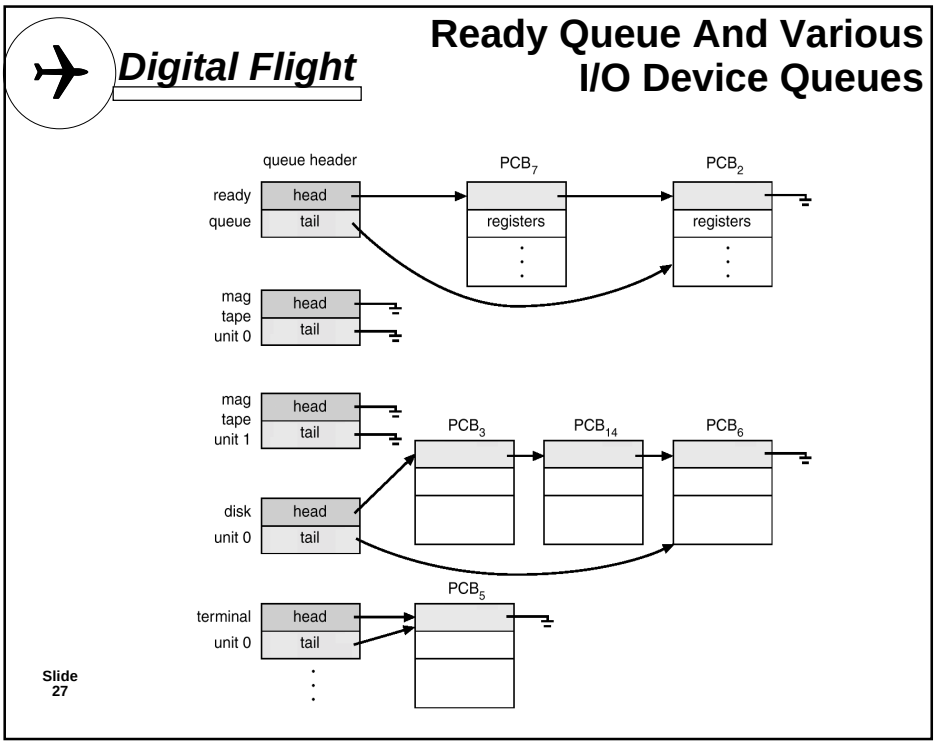
Slide 25

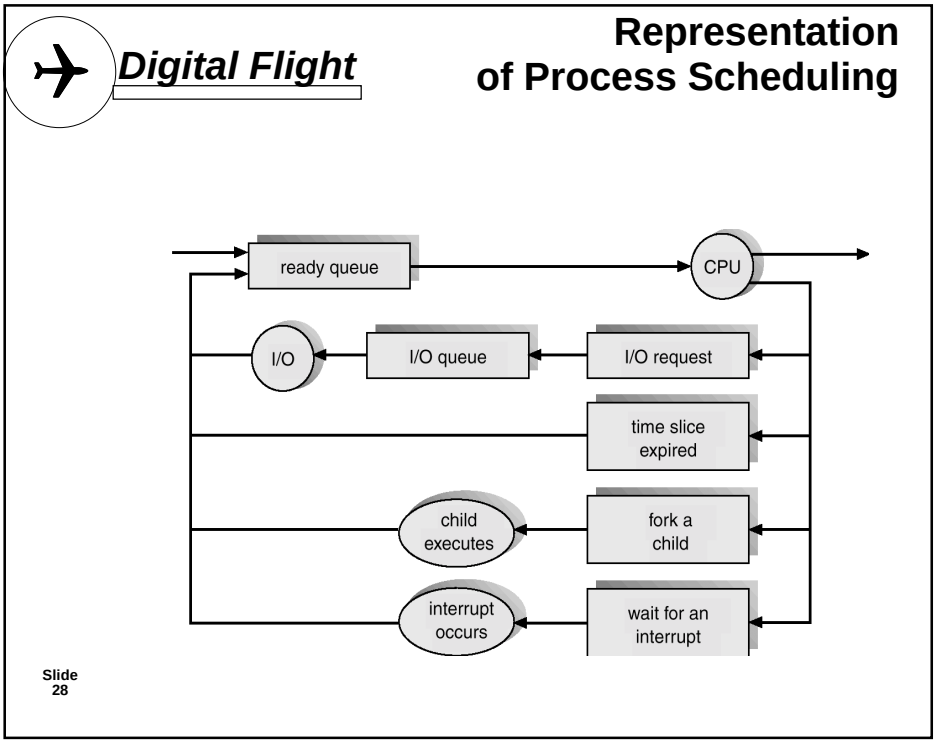
Digital Flight

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.**
- **Process migration between the various queues.**

Slide 26


Digital Flight
Ready Queue And Various
I/O Device Queues
queue header
PCB7
PCB2
ready
queue
head
tail
registers
registers
mag
tape
unit 0
head
tail
mag
tape
unit 1
head
tail
PCB3
PCB14
PCB6
disk
unit 0
head
tail
PCB5
terminal
unit 0
head
tail
Slide
27


Digital Flight
Representation
of Process Scheduling
ready queue
CPU
I/O
I/O queue
I/O request
time slice
expired
child
executes
fork a
child
interrupt
occurs
wait for an
interrupt
Slide
28

Digital Flight

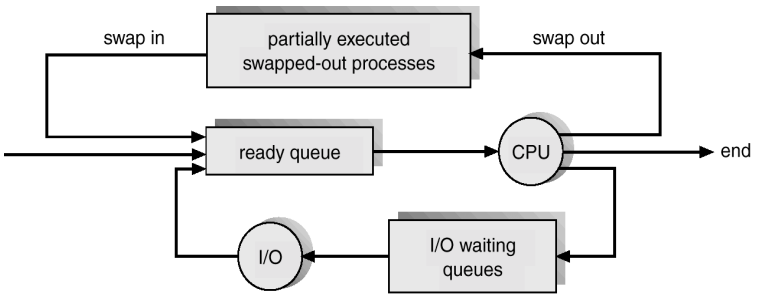
Schedulers

- 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.

Slide 29

Digital Flight

Addition of Medium Term Scheduling



Slide 30

Digital Flight

Schedulers (Cont.)

- Short-term scheduler is invoked very frequently (milliseconds)⇒ (must be fast).
- Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slow).
- The long-term scheduler 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.

Slide 31

Digital Flight

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.

Slide 32

Digital Flight

Process Termination

- **Process executes last statement and asks the operating system to terminate it (**exit**).**
  - **Process' resources are de-allocated by operating system.**

Slide 33

Digital Flight

Cooperating Processes

- ***Independent* process cannot affect or be affected by the execution of another process.**
- ***Cooperating* process can affect or be affected by the execution of another process**
- **Advantages of process cooperation**
  - **Information sharing**
  - **Computation speed-up**
  - **Modularity**
  - **Convenience**

Slide 34

Digital Flight

Producer-Consumer Problem

- Paradigm for cooperating processes, *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.

Slide 35

Digital Flight

Topics

Definitions

Overview

Common OS Components

- -System Initialization
- -Process Management
- -Memory Management
- -I/O System Management
- -Protection

System Services and Functions

Processes

Inter-process Communication

Process Synchronization

Memory Management

I/O

Safety-critical Specifics (Green Hills Software Inc.)*

Slide 36

Digital Flight

Interprocess Communication (IPC)

- **Mechanism for processes to communicate and 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***)

Slide 37

Digital Flight

IPC (Cont'd)

- **If *P* and *Q* wish to communicate 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)**

Slide 38

Digital Flight

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?

Slide 39

Digital Flight

Direct Communication

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

Slide 40

Digital Flight

Direct Communication (Cont'd)

- 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.

Slide 41

Digital Flight

Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports).
  - Each mailbox has a unique id.
  - Processes communicate only if they share a mailbox.
- Properties of communication link
  - Link established 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.

Slide 42

Digital Flight

Indirect Communication (Cont'd)

- **Operations**
  - **create a new mailbox**
  - **send and receive messages through mailbox**
  - **destroy a mailbox**

Slide 43

Digital Flight

Indirect Communication (Cont'd)

- **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.**

Slide 44

Digital Flight

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.

Slide 45

Digital Flight

Exception Conditions – Error Recovery

- Process terminates
- Lost messages
- Scrambled Messages

Slide 46

Digital Flight

Topics

Definitions

Overview

Common OS Components

- -System Initialization
- -Process Management
- -Memory Management
- -I/O System Management
- -Protection

System Services and Functions

Processes

Inter-process Communication

Process Synchronization

Memory Management

I/O

Safety-critical Specifics (Green Hills Software Inc.)*

Slide 47

Digital Flight

Process Synchronization

- Background
- The Critical-Section Problem
- Synchronization Hardware
- Semaphores
- Classical Problems of Synchronization
- Critical Regions
- Monitors

Slide 48

Digital Flight

Background

- Concurrent access to shared data may result in data inconsistency.
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes.
- Shared-memory solution to bounded-butter problem allows at most *n* – 1 items in buffer at the same time. A solution, where all *N* buffers are used is not simple.

Slide 49

Digital Flight

The Critical-Section Problem

- n processes all competing to use some shared data
- Each process has a code segment, called *critical section*, in which the shared data is accessed.
- Problem – ensure that when one process is executing in its critical section, no other process is allowed to execute in its critical section.

Slide 50

Digital Flight

Solution to Critical-Section Problem

1. Mutual Exclusion. **If process *Pi* is executing in its critical section, no other processes can be executing in their critical sections.**

2. Progress. **If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.**

Slide 51

Digital Flight

Solution to Critical-Section Problem

3. Bounded Waiting. **A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.**

   Assume that each process executes at a nonzero speed

   No assumption concerning relative speed of the *n* processes.

Slide 52

Digital Flight

Semaphore

- **Synchronization tool that does not require busy waiting.**
- **Semaphore *S* – integer variable**
- **can only be accessed via two indivisible (atomic) operations**

***wait* (*S*):** while ***S*≤ 0** do ***no-op*;**
$S := S - 1$;

***signal* (*S*): $S := S + 1$;**

Slide 53

Digital Flight

Semaphore Implementation

- **Assume two simple operations:**
  - **block suspends the process that invokes it.**
  - **wakeup(*P*) resumes the execution of a blocked process P.**

Slide 54

Digital Flight

Deadlock

- Deadlock – two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes.

- Let *S* and *Q* be two semaphores initialized to 1

P_0	P_1
wait(*S*);	*wait*(*Q*);
wait(*Q*);	*wait*(*S*);
⋮	⋮
signal(*S*);	*signal*(*Q*);
signal(*Q*)	*signal*(*S*);

Slide 55

Digital Flight

Starvation

- Starvation – indefinite blocking.

Slide 56

Digital Flight

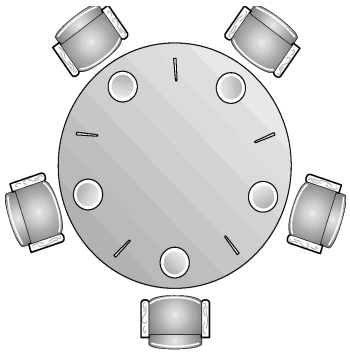
Classical Problems of Synchronization

- Bounded-Buffer Problem
- Readers and Writers Problem
- Dining-Philosophers Problem

Slide 57

Digital Flight

Dining-Philosophers Problem



Slide 58

Digital Flight

Critical Regions

- **High-level synchronization construct**
- **A shared variable *v* of type *T*, is declared as:**

  var ***v*:** shared ***T***
- **Variable *v* accessed only inside statement**

  region ***v*** when ***B*** do ***S***

where *B* is a Boolean expression.
While statement *S* is being executed, no other process can access variable *v*.

Slide 59

Digital Flight

Critical Regions (Cont.)

- **Regions referring to the same shared variable exclude each other in time.**
- **When a process tries to execute the region statement, the Boolean expression *B* is evaluated. If *B* is true, statement *S* is executed. If it is false, the process is delayed until *B* becomes true and no other process is in the region associated with *v*.**

Slide 60

Digital Flight

Monitors

- **High-level synchronization construct that allows the safe sharing of an abstract data type among concurrent processes.**

Slide 61

Digital Flight

Monitors (Cont'd)

- **To allow a process to wait within the monitor, a *condition* variable must be declared, as**

  var ***x, y: condition***

- **Condition variable can only be used with the operations *wait* and *signal*.**
  - **The operation**

    x.wait;

    means that the process invoking this operation is suspended until another process invokes

    ***x.signal*;**
  - **The *x.signal* operation resumes exactly one suspended process. If no process is suspended, then the signal operation has no effect.**

Slide 62

Digital Flight

Deadlocks

- System Model
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock
- Combined Approach to Deadlock Handling

Slide 63

Digital Flight

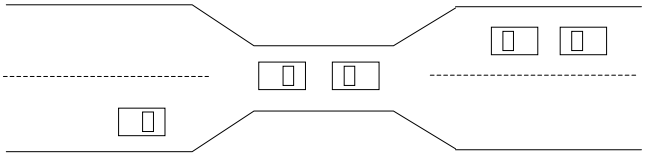
The Deadlock Problem

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.
- Example
  - System has 2 tape drives.
  - P_1 and P_2 each hold one tape drive and each needs another one.
- Example
  - semaphores *A* and *B*, initialized to 1

Slide 64

Digital Flight

Bridge Crossing Example



- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible.

Slide 65

Digital Flight

System Model

- Resource types $R_1, R_2, \ldots, R_m$

  CPU cycles, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
  - request
  - use
  - release

Slide 66

Digital Flight

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.:

- Mutual exclusion: **only one process at a time can use a resource.**
- Hold and wait: **a process holding at least one resource is waiting to acquire additional resources held by other processes.**
- No preemption: **a resource can be released only voluntarily by the process holding it, after that process has completed its task.**

Slide 67

Digital Flight

Deadlock Characterization (Cont'd)

- Circular wait:
  - **there exists a set {P_0, P_1, …, P_0} of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.**

Slide 68

Digital Flight

Resource-Allocation Graph

A set of vertices *V* and a set of edges *E*.

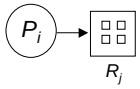
- **V is partitioned into two types:**
  - ***P* = {P_1, P_2, …, P_n}, the set consisting of all the processes in the system.**
  - ***R* = {R_1, R_2, …, R_m}, the set consisting of all resource types in the system.**
- **request edge – directed edge $P_1 \rightarrow R_j$**
- **assignment edge – directed edge $R_j \rightarrow P_i$**

Slide 69

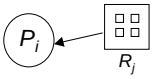
Digital Flight

Resource-Allocation Graph (Cont.)

- **Process**
- **Resource Type with 4 instances**
- **P_i requests instance of R_j**



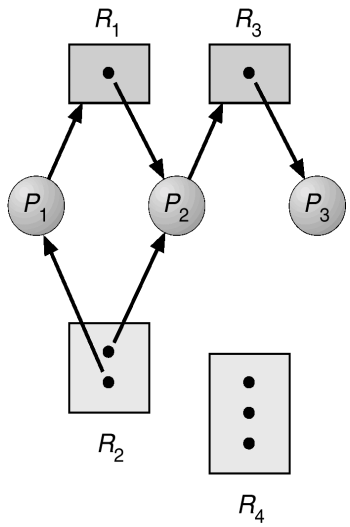
- **P_i is holding an instance of R_j**



Slide 70

Digital Flight

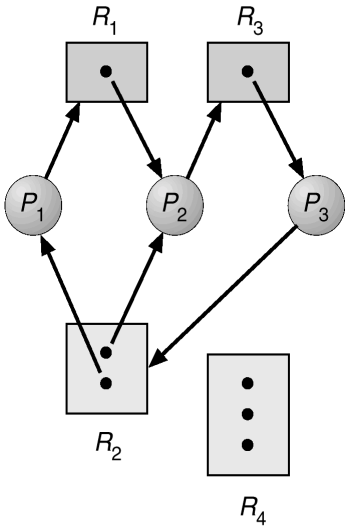
Example of a Resource Allocation Graph



Slide 71

Digital Flight

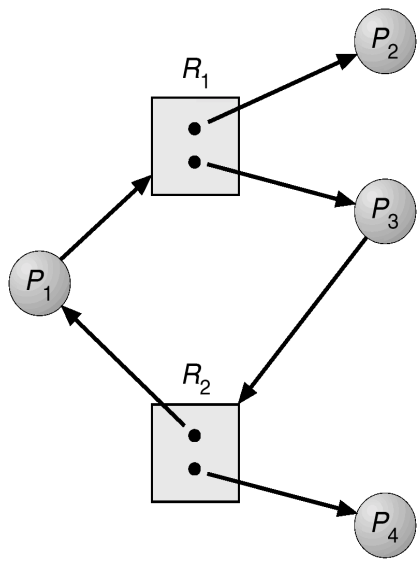
Resource Allocation Graph With A Deadlock



Slide 72

Digital Flight

Resource Allocation Graph With A Cycle But No Deadlock



Slide 73

Digital Flight

Basic Facts

- If graph contains no cycles ⇒ no deadlock.
- If graph contains a cycle ⇒
  - if only one instance per resource type, then deadlock.
  - if several instances per resource type, possibility of deadlock.

Slide 74

Digital Flight

Methods for Handling Deadlocks

- Ensure that the system will *never* enter a deadlock state.
- Allow the system to enter a deadlock state and then recover.
- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems.

Slide 75

Digital Flight

Deadlock Prevention

Restrain the ways request can be made.

- Mutual Exclusion – not required for sharable resources; must hold for nonsharable resources.
- Hold and Wait – must guarantee whenever a process requests a resource, it does not hold any other resources.
  - Require process to request and get all its resources before execution
  - Low resource utilization; starvation possible.

Slide 76

Digital Flight

Deadlock Prevention (Cont.)

- No Preemption –
  - If a process holding some resources requests another resource that can't be immediately allocated to it, then process releases all resources currently held .
  - Process restarted only when it can regain its old resources, as well as the new ones it requests.
- Circular Wait – impose a total ordering of all resource types, requires each process requests resources in increasing order of enumeration.

Slide 77

Digital Flight

Deadlock Avoidance

- Simplest and most useful model requires each process declare *maximum number* of resources of each type that it may need.
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure no circular-wait condition.
- Resource-allocation *state* is number of available and allocated resources, and maximum demands of processes.

Slide 78

Digital Flight

Safe State

- **When process requests available resource, system determines if immediate allocation leaves the system in a safe state.**
- **System is in safe state if there exists a safe sequence of all processes.**

Slide 79

Digital Flight

Safe State

- **Sequence <P_1, P_2, …, P_n> is safe if for each P_i, the resources that *Pi* can still request satisfied by currently available resources + resources held by all the P_j, with $j<i$.**
  - **If P_i resource needs not immediately available, then P_i can wait until all P_j have finished.**
  - **When P_j finishes, P_i obtains needed resources, executes, returns allocated resources, and terminates.**
  - **When P_i terminates, P_{i+1} can obtain its needed resources, and so on.**

Slide 80

Digital Flight

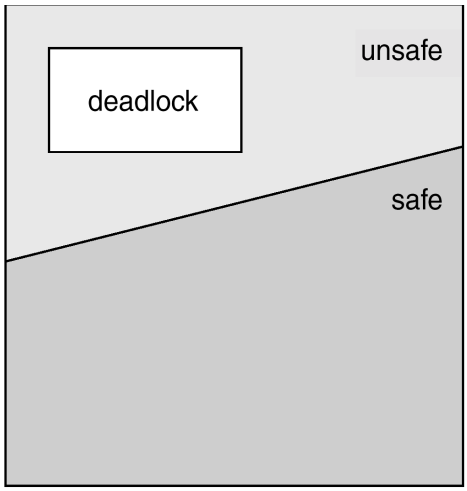
Basic Facts

- If a system is in safe state ⇒ no deadlocks.
- If a system is in unsafe state ⇒ possibility of deadlock.
- Avoidance ⇒ ensure that a system will never enter an unsafe state.

Slide 81

Digital Flight

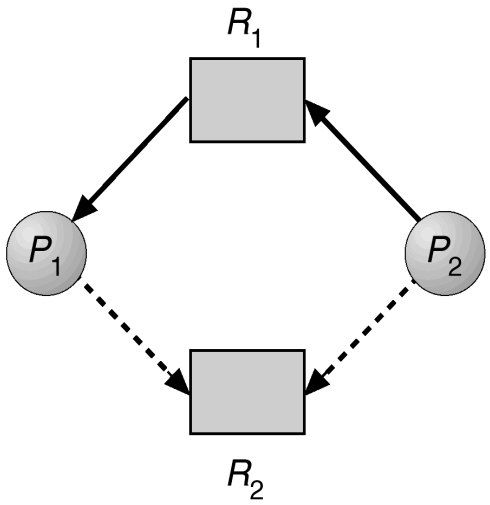
Safe, Unsafe , and Deadlock State Space



Slide 82

Digital Flight

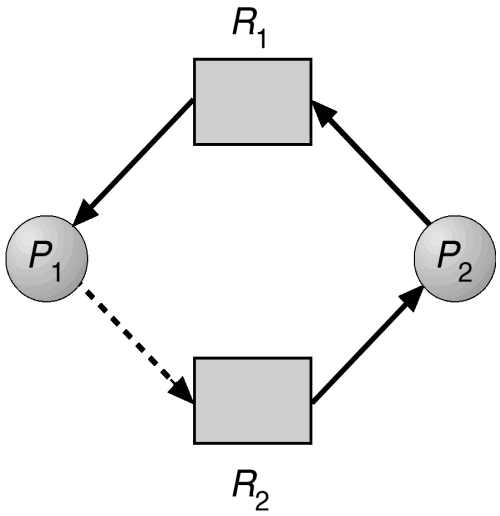
Resource-Allocation Graph For Deadlock Avoidance



Slide 83

Digital Flight

Unsafe State In A Resource Allocation Graph



Slide 84

Digital Flight

Deadlock Detection

- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme

Slide 85

Digital Flight

Detection-Algorithm Usage

- When, and how often, to invoke depends on:
  - How often a deadlock is likely to occur?
  - How many processes will need to be rolled back?
    - one for each disjoint cycle
- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes "caused" the deadlock.

Slide 86

Digital Flight

Recovery from Deadlock: Process Termination

- Abort all deadlocked processes.
- Abort one process at a time until the deadlock cycle is eliminated.
- In which order should we choose to abort?
  - Priority of the process.
  - How long process has computed, and how much longer to completion.
  - Resources the process has used.
  - Resources process needs to complete.
  - How many processes will need to be terminated.
  - Is process interactive?

Slide 87

Digital Flight

Recovery from Deadlock: Resource Preemption

- Selecting a victim – minimize cost.
- Rollback – return to some safe state, restart process from that state.
- Starvation – same process may always be picked as victim, include number of rollback in cost factor.

Slide 88

Digital Flight

Combined Approach to Deadlock Handling

- Combine the three basic approaches
  - prevention
  - avoidance
  - detection

  allowing the use of the optimal approach for each of resources in the system.
- Partition resources into hierarchically ordered classes.
- Use most appropriate technique for handling deadlocks within each class.

Slide 89

Digital Flight

Topics

Definitions

Overview

Common OS Components

-System Initialization

-Process Management

-Memory Management

-I/O System Management

-Protection

System Services and Functions

Processes

Inter-process Communication

Process Synchronization

Memory Management

I/O

Safety-critical Specifics (Green Hills Software Inc.)*

Slide 90

Digital Flight

Memory Management

- Background
- Logical versus Physical Address Space
- Swapping
- Contiguous Allocation
- Paging
- Segmentation
- Segmentation with Paging

Slide 91

Digital Flight

Background

- Program must be brought into memory and placed within a process for it to be executed.
- *Input queue* – collection of processes on the disk that are waiting to be brought into memory for execution.
- User programs go through several steps before being executed.

Slide 92

Digital Flight

Binding of Instructions and Data to Memory

- Compile time: **If memory location known a priori, absolute code can be generated; must recompile code if starting location changes.**
- Load time: **Must generate *relocatable* code if memory location is not known at compile time.**
- Execution time: **Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., *base* and *limit registers*).**

Slide 93

Digital Flight

Dynamic Linking

- **Linking postponed until execution time.**
- **Small piece of code, *stub*, used to locate the appropriate memory-resident library routine.**
- **Stub replaces itself with the address of the routine, and executes the routine.**
- **Operating system needed to check if routine is in processes' memory address.**

Slide 94

Digital Flight

Logical vs. Physical Address Space

- The concept of a logical *address space* that is bound to a separate *physical address space* is central to proper memory management.
  - *Logical address* – generated by the CPU; also referred to as *virtual address*.
  - *Physical address* – address seen by the memory unit.
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme.

Slide 95

Digital Flight

Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address.
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.
- The user program deals with *logical* addresses; it never sees the *real* physical addresses.

Slide 96

Digital Flight

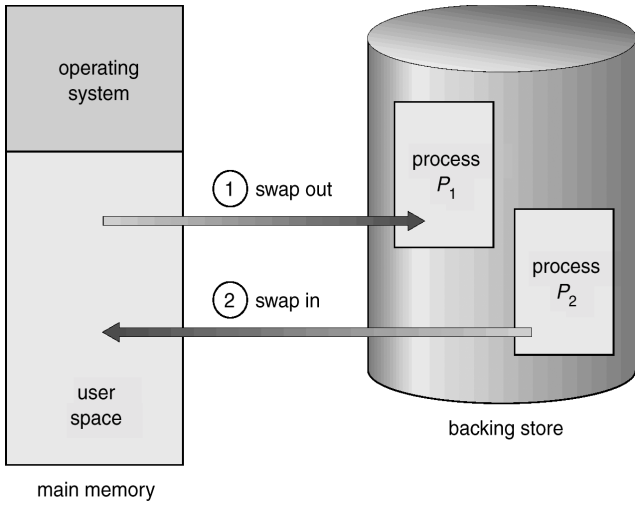
Swapping

- A process can be *swapped* temporarily out of memory to a *backing store*, and then brought back into memory for continued execution.
- *Roll out, roll in* – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed.
- Major part of swap time is transfer time; total transfer time is directly proportional to the *amount* of memory swapped.

Slide 97

Digital Flight

Schematic View of Swapping



Slide 98

Digital Flight

Contiguous Allocation

- Main memory usually into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector.
  - User processes then held in high memory.
- Single-partition allocation
  - Relocation-register scheme protects user processes from each other, and from changing operating-system code and data.
  - Relocation register contains value of smallest physical address; limit register contains range of logical addresses – each logical address must be less than the limit register.

Slide 99

Digital Flight

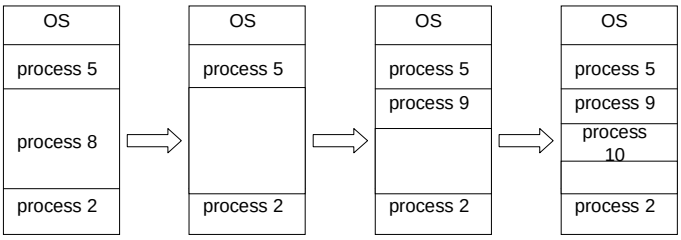
Contiguous Allocation (Cont.)

- Multiple-partition allocation
  - *Hole* – block of available memory; holes of various size are scattered throughout memory.
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it.
  - Operating system maintains information about:
    a) allocated partitions b) free partitions (hole)

Slide 100

Digital Flight

Example



Slide 101

Digital Flight

Dynamic Storage Allocation Problem

- First-fit**: Allocate *first* hole big enough.**
- Best-fit**: Allocate *smallest* hole big enough; must search entire list, unless ordered by size.**
- Worst-fit**: Allocate the *largest* hole; must also search entire list.**

First-fit and best-fit better than worst-fit in terms of speed and storage utilization.

Slide 102

Digital Flight

Fragmentation

- External fragmentation – enough memory space exists to satisfy a request, but not contiguous.
- Internal fragmentation – allocated memory slightly larger than requested memory; size difference is memory internal to partition, but not being used.

Slide 103

Digital Flight

Fragmentation

Reduce external fragmentation by compaction

- Shuffle memory contents to place all free memory together in one large block.
- Compaction is possible *only* if relocation is dynamic, and is done at execution time.
- I/O problem
  - Latch job in memory while it is involved in I/O.
  - Do I/O only into OS buffers.

Slide 104

Digital Flight

Paging

- Logical address space of a process can be noncontiguous
- Divide physical memory into fixed-sized blocks called frames (size is power of 2, between 512 bytes and 8192 bytes).
- Divide logical memory into blocks of same size called pages.
- Keep track of all free frames.
- To run a program of size n pages, need to find *n* free frames and load program.
- Set up a page table to translate logical to physical addresses.
- Internal fragmentation.

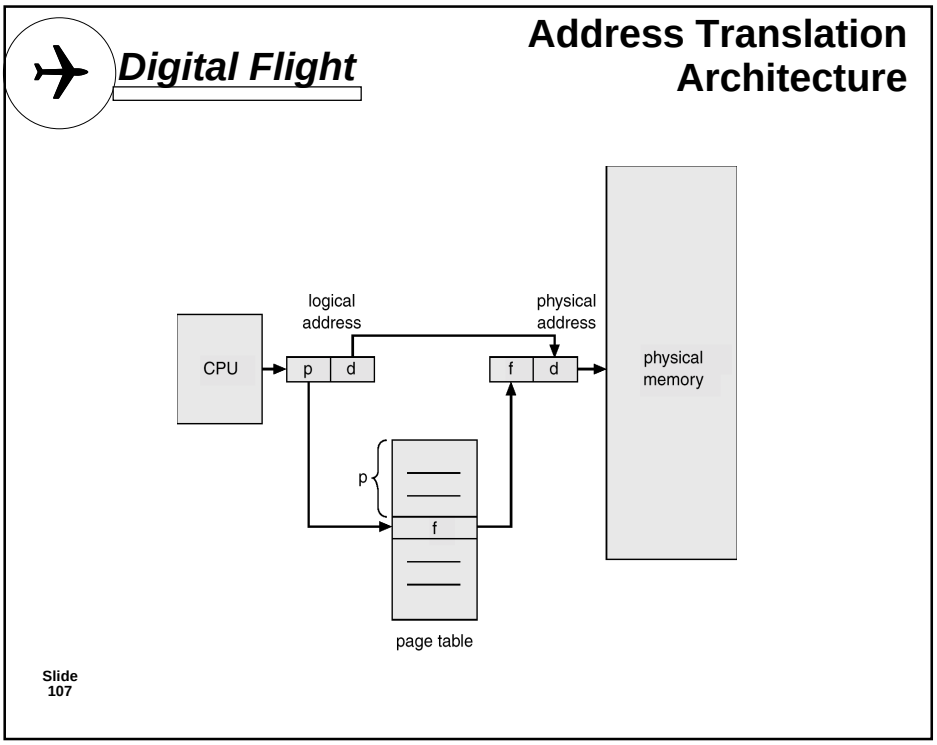
Slide 105

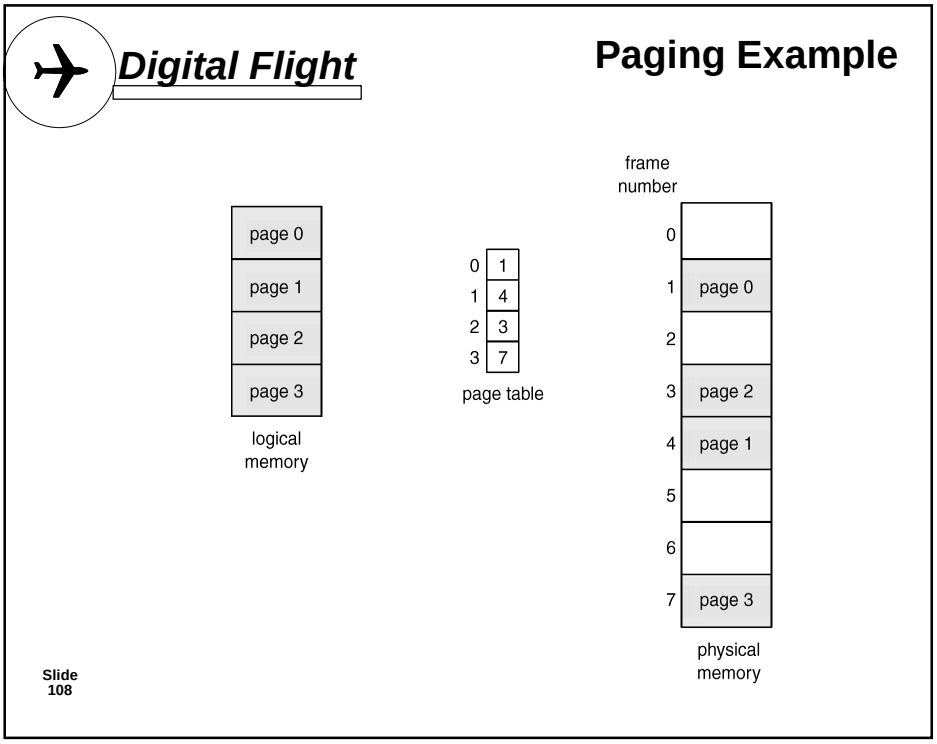
Digital Flight

Address Translation Scheme

- Address generated by CPU is divided into:
  - *Page number (p)* – used as an index into a *page table* which contains base address of each page in physical memory.
  - *Page offset (d)* – combined with base address to define the physical memory address that is sent to the memory unit.

Slide 106


Digital Flight
Address Translation Architecture
CPU
logical address
p
d
physical address
f
d
physical memory
p
f
page table
Slide 107


Digital Flight
Paging Example
page 0
page 1
page 2
page 3
logical memory
0 1
1 4
2 3
3 7
page table
frame number
0
1 page 0
2
3 page 2
4 page 1
5
6
7 page 3
physical memory
Slide 108

Digital Flight

Implementation of Page Table

- Page table kept in main memory.
- *Page-table base register (*PTBR) points to the page table.
- *Page-table length register* (PRLR) indicates size of the page table.
- Every data/instruction access requires two memory accesses. One for page table - one for data/instruction.
- Two memory access problem solved by use of fast-lookup hardware cache called *associative registers*

Slide 109

Digital Flight

Associative Register

- Associative registers – parallel search

Page #	Frame #

- Address translation (A´, A´´)
  - If A´ is in associative register, get frame # out.
  - Otherwise get frame # from page table in memory

Slide 110

Digital Flight

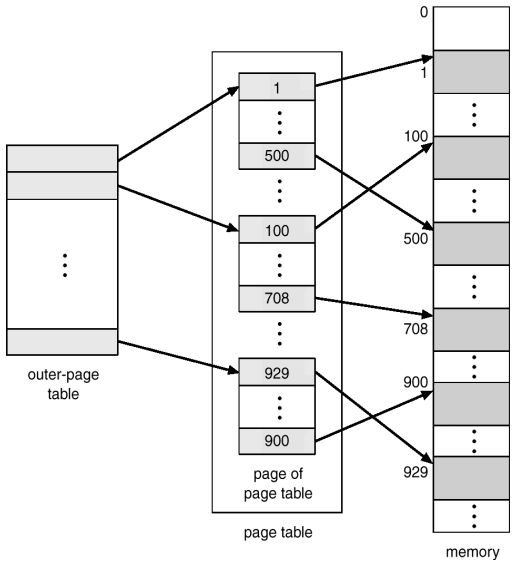
Memory Protection

- Memory protection implemented by associating protection bit with each frame.
- *Valid-invalid* bit attached to each entry in the page table:
  - "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page.
  - "invalid" indicates that the page is not in the process' logical address space.

Slide 111

Digital Flight

Two-Level Page-Table Scheme

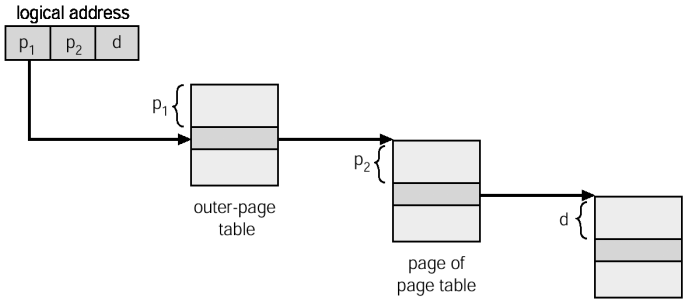


Slide 112

Digital Flight

Address Translation Scheme

- Address-translation scheme for a two-level 32-bit paging architecture



Slide 113

Digital Flight

Multilevel Paging and Performance

- Each level stored as separate table in memory, covering a logical address to a physical one may take four memory accesses.
- Though time needed for one memory access is quintupled, caching permits performance to remain reasonable.

Slide 114

Digital Flight

Comparing Memory Management Strategies

- Hardware support
- Performance
- Fragmentation
- Relocation
- Swapping
- Sharing
- Protection

Slide 115

Digital Flight

Virtual Memory

- Background
- Demand Paging
- Performance of Demand Paging
- Page Replacement
- Page-Replacement Algorithms
- Allocation of Frames
- Thrashing
- Other Considerations
- Demand Segmenation

Slide 116

Digital Flight

Background

- Virtual memory – separation of user logical memory from physical memory.
  - Only part of the program needs to be in memory for execution.
  - Logical address space can therefore be much larger than physical address space.
  - Need to allow pages to be swapped in and out.
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Slide 117

Digital Flight

Demand Paging

- Bring a page into memory only when it is needed.
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory

Slide 118

Digital Flight

Page Fault

- If reference to a page, first reference will trap OS ⇒ page fault
- OS looks at another table to decide:
  - Invalid reference ⇒ abort.
  - Just not in memory.
- Get empty frame.
- Swap page into frame, reset tables, validation bit = 1.
- Restart instruction: Least Recently Used
  - block move
  - auto increment/decrement location

Slide 119

Digital Flight

What happens if there is no free frame?

- Page replacement – find some page in memory, but not really in use, swap it out.
  - algorithm
  - performance – want an algorithm which will result in minimum number of page faults.
- Same page may be brought into memory several times.

Slide 120

Digital Flight

Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.
- Use *modify* (*dirty*) *bit* to reduce overhead of page transfers – only modified pages are written to disk.
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.

Slide 121

Digital Flight

Page-Replacement Algorithms

- Want lowest page-fault rate.
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.

Slide 122

Digital Flight

Page-Replacement Algorithms (Cont'd)

- First-In-First-Out (FIFO) replaces the oldest page
- Least Recently Used (LRU) replaces the page that was not referenced for the longest period of time
- Least Frequently Used (LFU) replaces page that was referenced the least

Slide 123

Digital Flight

Topics

Definitions

Overview

Common OS Components

-System Initialization

-Process Management

-Memory Management

-I/O System Management

-Protection

System Services and Functions

Processes

Inter-process Communication

Process Synchronization

Memory Management

I/O Systems

Safety-critical Specifics (Green Hills Software Inc.)*

Slide 124

Digital Flight

I/O Systems

- I/O hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- Performance

Slide 125

Digital Flight

I/O Hardware

- Incredible variety of I/O devices
- Common concepts
  - Port
  - Bus (daisy chain or shared direct access)
  - Controller (host adapter)
- I/O instructions control devices
- Devices have addresses, used by
  - Direct I/O instructions
  - Memory-mapped I/O

Slide 126

Digital Flight

Polling

- Determines state of device
  - `command-ready`
  - `busy`
  - `error`
- Busy-wait cycle to wait for I/O from device

Slide 127

Digital Flight

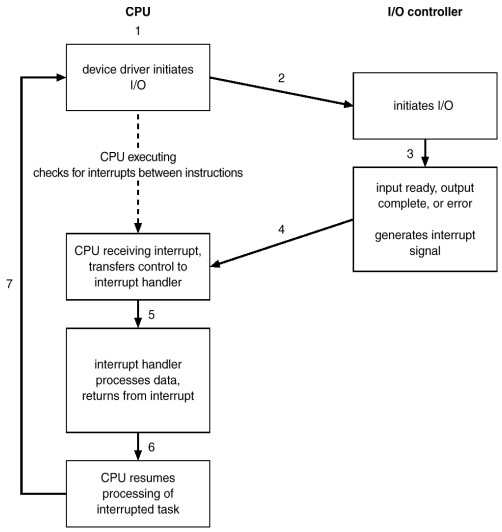
Interrupts

- CPU Interrupt request line triggered by I/O device
- Interrupt handler receives interrupts
- Maskable to ignore or delay some interrupts
- Interrupt vector to dispatch interrupt to correct handler
  - Based on priority
  - Some unmaskable
- Interrupt mechanism also used for exceptions

Slide 128

Digital Flight

Interrupt-drive I/O Cycle



Slide 129

Digital Flight

Direct Memory Access

- Used to avoid programmed I/O for large data movement
- Requires DMA controller
- Bypasses CPU to transfer data directly between I/O device and memory

Slide 130

Digital Flight

Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes
- Device-driver layer hides differences among I/O controllers from kernel
- Devices vary in many dimensions
  - Character-stream or block
  - Sequential or random-access
  - Sharable or dedicated
  - Speed of operation
  - read-write, read only, or write only

Slide 131

Digital Flight

Clocks and Timers

- Provide current time, elapsed time, timer
- if programmable interval time used for timings, periodic interrupts

Slide 132

Digital Flight

Blocking and Non-blocking I/O

- Blocking - process suspended until I/O completed
  - Easy to use and understand
  - Insufficient for some needs
- Nonblocking - I/O call returns as much as available
  - User interface, data copy (buffered I/O)
  - Returns quickly with count of bytes read or written

Slide 133

Digital Flight

Blocking and Non-blocking I/O (Cont'd)

- Asynchronous - process runs while I/O executes
  - Difficult to use
  - I/O subsystem signals process when I/O completed

Slide 134

Digital Flight

Kernel I/O Subsystem

- Scheduling
  - Some I/O request ordering via per-device queue
  - Some OSs try fairness
- Buffering - store data in memory while transferring between devices
  - To cope with device speed mismatch
  - To cope with device transfer size mismatch
  - To maintain “copy semantics”

Slide 135

Digital Flight

Kernel I/O Subsystem

- Caching - fast memory holding copy of data
  - Always just a copy
  - Key to performance
- Device reservation - provides exclusive access to a device
  - System calls for allocation and deallocation
  - Watch out for deadlock

Slide 136

Digital Flight

Error Handling

- OS can recover from read, device unavailable, transient write failures
- Most return an error number or code when I/O request fails
- Log system errors

Slide 137

Digital Flight

Kernel Data Structures

- Kernel keeps state info for I/O components, including open file tables, network connections, character device state
- Many, many complex data structures to track buffers, memory allocation, “dirty” blocks
- Some use object-oriented methods and message passing to implement I/O

Slide 138

Digital Flight

I/O Requests to Hardware Operations

- Consider reading a file from disk for a process
  - Determine device holding file
  - Translate name to device representation
  - Physically read data from disk into buffer
  - Make data available to requesting process
  - Return control to process

Slide 139

Digital Flight

Performance

- I/O a major factor in system performance
  - Demands CPU to execute device driver, kernel I/O code
  - Context switches due to interrupts
  - Data copying
  - Network traffic especially stressful

Slide 140

Digital Flight

Improving Performance

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling
- Use DMA
- Balance CPU, memory, bus, and I/O performance for highest throughput

Slide 141

Digital Flight

Topics

Definitions

Overview

Common OS Components

-System Initialization

-Process Management

-Memory Management

-I/O System Management

-Protection

System Services and Functions

Processes

Inter-process Communication

Process Synchronization

Memory Management

I/O

Safety-critical Specifics (Green Hills Software Inc.)*

Slide 142

Digital Flight

Safety Critical Specifics

Safety-critical Architectures

Why protection?

Guaranteed Resource Availability

- time
- space

Kernal Determinism & interrupt Latency

High Availability

Access Control

Schedulabilty

Protected Device Driver

Slide 143

Digital Flight

Safety-Critical Architectures

- Cyclic Executive
  - Periodic task timeline
  - Periods – harmonically related
  - Manual Multi-Tasking
- Interrupt-Driven Architecture
  - Tasks mapped to HW interrupts
  - Tasks become ISR's and HW interrupt priorities enforce SW Priority Scheduling

Slide 144

Digital Flight

Safety-Critical Architectures

- "No Tasking" run-time requirements
  - Start-up/Initialization code
  - Exception handling support
  - Math library support
  - Compiler support routines
    - Block moves, etc.
  - Deterministic execution time for all services
  - Multi-Language support
  - Subset of a "general purpose" run-time solution

Slide 145

Digital Flight

Safety-Critical Architectures

- Single Application / Multi-Tasking
  - Tasks scheduled using Priority-Based Preemptive Scheduler
  - Rate Monotonic Analysis (RMA) used to select priorities
    - Shorter period => Higher Priority
  - Task Communication which prohibits Priority Inversion
  - Emulate "event-driven" tasks with periodic "Sporadic Servers"

Slide 146

Digital Flight

Safety-Critical Architectures

Multi-Tasking run-time requirements

- **Priority-Based Preemptive Scheduler**
- **Highest Locker's or Priority Ceiling Protocol Semaphore**
  - **Protection against Priority Inversion**
  - **Minimum bounds on blocking time**
  - **Enables task communication**
- **Deterministic execution time for all services**
  - **Context Switch time**
  - **Semaphore Give/Take**
  - **No dynamic memory allocation**
- **Multi-Language support**
- **Subset of a "general purpose" run-time solution**

Slide 147

Digital Flight

Safety-Critical Architectures

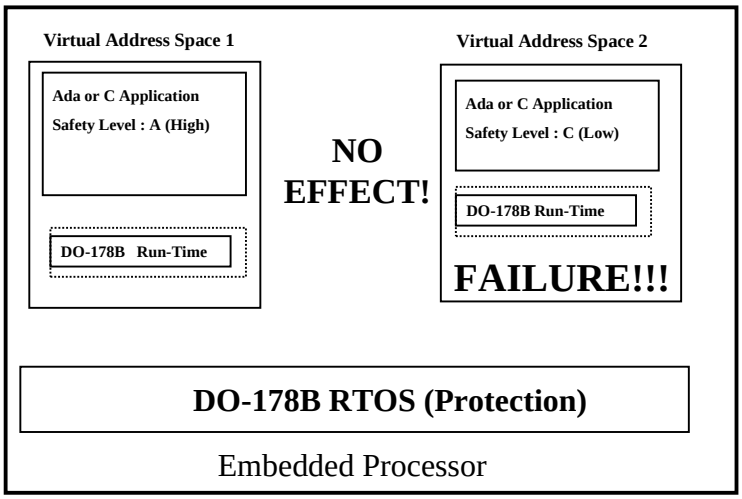
Multiple Multi-Tasking Applications with different Criticality Levels

- **Partitions developed/tested to appropriate levels**
- **Failures in one Partition effect no other**
- **Multi-Tasking Architecture reqts apply within a partition**
  - **Priority-based Preemptive Scheduler**
  - **Protection against Priority Inversion**
- **Partition scheduler enables time protection**
- **Partition cannot access another's memory or Kernel Objects (e.g. semaphores, tasks, etc.)**

Slide 148

Digital Flight

Safety-Critical Architectures



Slide 149

Digital Flight

Safety-Critical Architectures

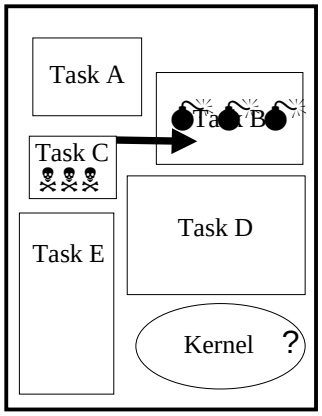
- Protected RTOS requirements
  - Provide "protected" partitions in time and space
  - Utilize HW MMU to protect memory accesses
  - Schedule tasks on a partition-basis as well as a priority basis
  - Multi-Tasking Architecture reqts apply within a partition
  - RTOS needs to protect itself from invalid accesses
  - Deterministic execution time for all services
  - Multi-Language Support
  - Subset of a "general purpose" run-time solution

Slide 150

Digital Flight

Why Protection?

Single Address Space



Tasks running in a single address space are free to:

- Corrupt the memory of other tasks

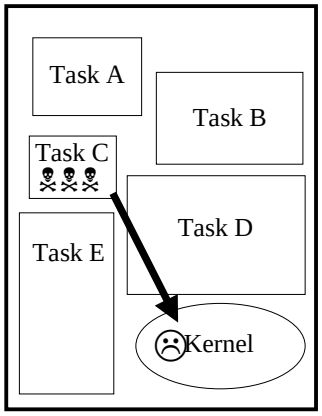
The Kernel is unaware!

Slide 151

Digital Flight

Why Protection?

Single Address Space



Tasks running in a single address space are free to:

- Corrupt the memory of other tasks
- Corrupt the kernel

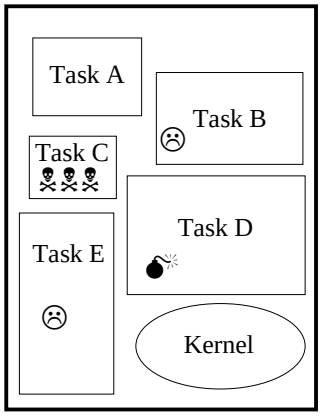
The Kernel can't prevent it!

Slide 152

Digital Flight

Why Protection?

Single Address Space



Tasks running in a single address space are free to:

- Corrupt the memory of other tasks
- Corrupt the kernel
- Create intermittent bugs that are difficult to reproduce and correct

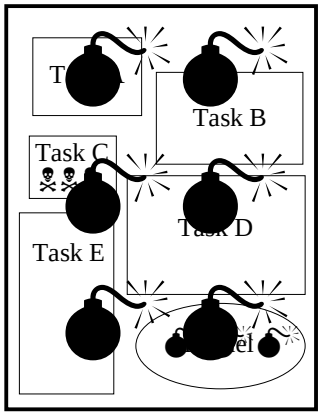
This can happen at any time

Slide 153

Digital Flight

Why Protection?

Single Address Space



Tasks running in a single address space are free to:

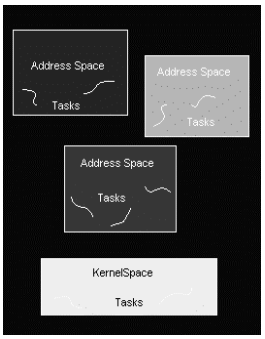
- Corrupt the memory of other tasks
- Corrupt the kernel
- Create intermittent bugs that are difficult to reproduce and correct
- Cause complete system failure

No Malicious coding needed...

Slide 154

Digital Flight

Guaranteed Resource Availability - Space Domain

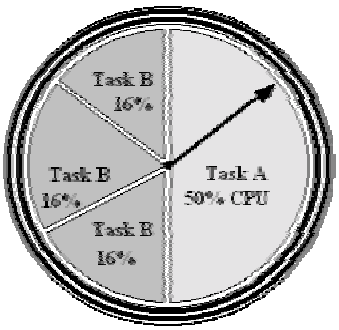


- System divided into virtual address spaces
- Each address space is provided a quota of physical memory
- Quota is hard currency that can't be forged, inflated, or stolen
- No central memory store in kernel!

Slide 155

Digital Flight

Guaranteed Resource Availability - Time Domain



- Tasks can be assigned weight within priority level
- Tasks cannot steal additional CPU time by spawning confederate tasks
  - Tasks spawned by a task must give up a portion of their CPU time allocation

Slide 156

Digital Flight

Guaranteed Resource Availability Time Domain*

Real-life denial of service failures:

- **The data entry technician and the X-ray exposure modulator**
- **Mars Pathfinder**

Slide 157

Digital Flight

Guaranteed Resource Availability - Time Domain

- Task schedulers have no concept of CPU resource protection at the address space level
- RTOS should provide an optional address space scheduler following the ARINC 653 partition scheduling model
- System designer defines execution windows for each partition
- When a partition is active, its CPU resource is absolutely guaranteed
- Provision for background partition

Slide 158

Digital Flight

Kernel Determinism and Interrupt Latency

Conventional

```
void
conventional_kernel_call(void
)
{
 ...
 disable_interrupts();
 <modify_kernel_data_structs>
 enable_interrupts();
 ...
}
```

- Interrupt disabling sequences strewn about in the kernel cause high priority interrupts to be postponed
- Mutexes may be used in kernel, potentially causing priority inversion unbeknownst to programmers
- Use of heuristics can fail in pathological cases

Slide 159

Digital Flight

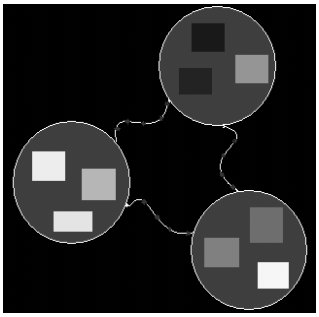
Kernel Determinism and Interrupt Latency

- Kernel should never disable interrupts in kernel calls - highest priority interrupts serviced with absolute minimum latency; scheduler timer interrupt handling informs kernel to get back into scheduler upon exit from service call.
- All service calls must have short, bounded computation times and are preemptible/restartable for operations which take longer.
- No mutexes used in the kernel
- No heuristics

Slide 160

Digital Flight

High Availability



- Task, Address Space restartability
- Software Watchdog, health monitor
- Connection Manager
  - Multiprocessor Distributed Processing
  - Built-in Heartbeat
  - Operated over LAN, backplane
  - Enables a variety of high availability models, redundant/standby nodes, hot fail-over

Slide 161

Digital Flight

Access Control

```
AddressSpace foo
    Object 10
          Task   HealthMonitor
          Entrypoint    HealthFunc
          Priority   200
    EndObject
    Object 11
         Link  bar
         OtherObjectNumber    10
    EndObject
EndAddressSpace

AddressSpace bar
    Object 10
          Task  EtherTask
          Entrypoint   EtherStart
    EndObject
    Object 11
        IODevice  etherdev
    EndObject
EndAddressSpace
```

- Kernel Objects (Task, Connection, etc): Static Boot table defines ownership for Address Spaces and thus mandatory access control of objects (enforced by kernel)

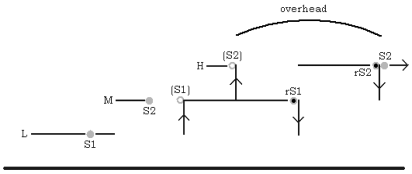
- Data (Memory): MMU protection enables mandatory access control
- Discretionary access control (e.g. Unix file) is not sufficient – no guarantee that improper access can't occur

Slide 162

Digital Flight

Schedulability

- Need to account for all execution time: not only code execution but also message transfers
- Typical solution for priority inversion, priority inheritance mutex, can cause chained blocking: very difficult to perform RMA and leads to an inefficient system:



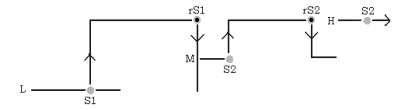
Chained Blocking Caused By Priority Inheritance

Slide 163

Digital Flight

Schedulability

- Highest Locker Semaphore prevents priority inversion and chained blocking
- HL Semaphore assigned a fixed priority – priority of the highest priority task that contends for the resource
- When task obtains HL semaphore, task priority is immediately elevated to HL priority; task never blocks on HL semaphore
- Results in a more efficient system:



Highest Locker

Slide 164

Digital Flight

Schedulability

RMA Example Comparing Highest Locker and Priority Inheritance Semaphore:

```
C represents computation time.
P represents period.

T1 : C1 = 60; P1 = 100
T2 : C2 = 40; P2 = 1000
T3 : C3 = 60; P3 = 1000

Compute Actual Utilization:
    U(1) = 60/100 = .6
    U(2) = .6 + 40/1000 = .64
    U(3) = .64 + 60/1000 = .70
Compute Maximum Utilization (RMA bound),
where RMA bound U(n) = n(2**(1/n)-1).
    U(1) = 1.0
    U(2) = .828
    U(3) = .779
Since .6 < 1.0, .64 < .828, .70 < .779, this
system is SCHEDULABLE.
```

Slide 165

Digital Flight

Schedulability

```
Now, let's assume there are 2 semaphores
shared between the tasks in the following manner:
    T1 and T2 access S1 and hold it for 25.
    T1 and T3 access S2 and hold it for 25.
RMA forces you to add the maximum blocking factor!

Actual Utilization with PI:
        For T1, each Semaphore take may cause a block:
U(1) = .6 + .5 = 1.1 > 1.0! NOT SCHEDULABLE!!

Actual Utilization with HL:
        Only one of S1 or S2 could be held when T1 is ready:
U(1) = 0.6 + 0.25 = 0.85 < 1.0, so SCHEDULABLE!!
        T2's blocking factor is also 25 since T3 may be
running at elevated priority when T2 becomes runnable:
U(2) = .6 + .04 + 25/1000 = 0.665 < 0.828, so SCHEDULABLE!!
        T3 has no blocking factor:
U(3) = .6 + .04 + 0.06 + 0 = 0.70 < 0.779, so SCHEDULABLE!!
```

Slide 166

Digital Flight

Protected Device Drivers

- User Mode Device Drivers
  - User Mode Task
    - Runs in its own virtual address space
    - Does nearly all of the work
  - Memory mapped I/O accessed via virtual memory mapping
  - Additional protection for more complicated devices
    - Devices that use DMA
    - Errant DMA programming could crash Kernel
    - Controls DMA so it can only read/write Memory that the User Mode Task owns
  - Interrupt Service Routine
    - Short
    - Services the physical device interrupt

Slide 167

Digital Flight

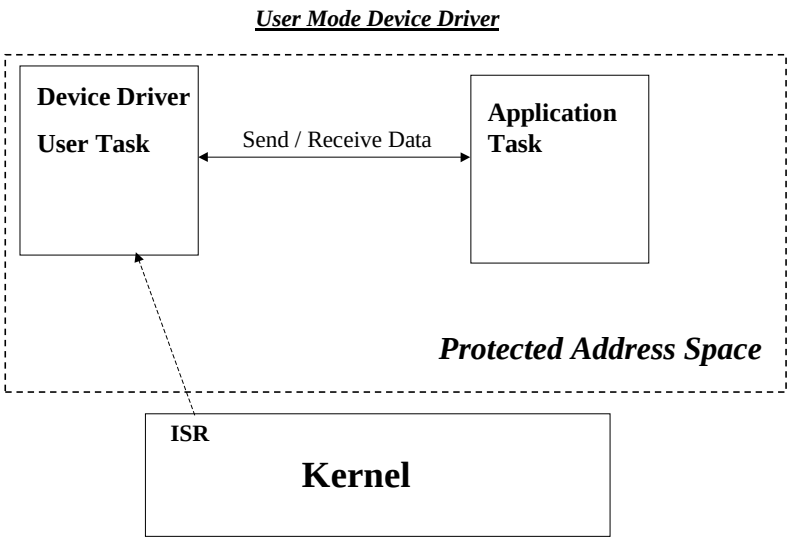
Protected Device Drivers

- User Mode Device Drivers
  - Completely secure
    - Can't corrupt OS or any other Address Space or device
    - Can't be corrupted by any other Address Space or device
  - Written in a high level language
    - Easier to program
    - Easier to debug
    - Runtime error detection (e.g. stack overflow)
    - Performance analysis
  - Fully preemptible: creates no interrupt shadow

Slide 168

Digital Flight

Protected Device Drivers



Slide 169

Digital Flight

Common Defects

- **Protection in the Time Domain**
  - Many RTOS vendors don't understand RMA
  - Task utilization support
  - schedulability analysis
  - Partition scheduler support (ARINC-653)
  - Bounded time kernel calls force smaller "subsets"
  - Hidden execution time (message passing execution time gets attributed to sender? receiver? both?)
  - Long Kernel calls (Close/Kill partition) pass work off to System Tasks
  - Non-deterministic resource reclamation
  - Cache? - Instruction Cache?

Slide 170

Digital Flight

Common Defects

- **Protection in the Space Domain**
  - Dynamic manipulation of the MMU complicates verification efforts
  - Using kernel memory pool for resource allocation allows other partitions to effect resource availability
  - Single protected segment for all “Code”, requires analysis of errant jumps
  - Preemptible kernels use task stacks for system calls
  - Dynamic installation of device drivers requires privileged user code

Slide 171