THE UNIVERSITY OF NEWCASTLE AUSTRALIA

FACULTY OF ENGINEERING AND BUILT ENVIRONMENT

www.newcastle.edu.au

OPERATING SYSTEMS

Week 2

Much of the material on these slides comes from the recommended textbook by William Stallings

Detailed content

Weekly program

- ✓ Week 1 – Operating System Overview
- ❑ **Week 2 – Processes and Threads**
- ❑ Week 3 – Scheduling
- ❑ Week 4 – Real-time System Scheduling and Multiprocessor Scheduling
- ❑ Week 5 – Concurrency: Mutual Exclusion and Synchornisation
- ❑ Week 6 – Concurrency: Deadlock and Starvation
- ❑ Week 7 – Memory Management I
- ❑ Week 8 – Memory Management II
- ❑ Week 9 – Disk and I/O Scheduling
- ❑ Week 10 – File Management
- ❑ Week 11 – Security and Protection
- ❑ Week 12 – Revision of the course
- ❑ Week 13 – Extra revision (if needed)

Key Concepts from last week

- Instruction execution phases
- Purpose of interrupt and interrupt processing mechanism
- Why do we need a memory hierarchy
- Principle of locality
- How cache memory works
- Different I/O techniques
- Multiprocessor and multicore architecture
- Batch processing, multiprogramming and time sharing OS
- Fault tolerance

Week 02 Lecture Outline

Processes and Threads

- ❑ What is a process?
- ❑ Process Control Block
- ❑ Process Models
  - ❑ Two-State Model
  - ❑ Five-State Model
  - ❑ Seven-State Model
- ❑ Process Description
- ❑ Process Control
- ❑ Threads
- ❑ Multi-threading
- ❑ Types of Threads
- ❑ Multicore and Multithreading

Videos to watch before lecture

OS Managements of application execution

- The processor is switched among multiple applications (process)
  - Maximize processor utilization
  - Provide reasonable response time
- Resources are made available to multiple applications (process)
  - Should be used efficiently
  - Avoid starvation/deadlock
- OS may be required to support inter-process communication

What is a process?

- A program in execution
- An instance of a program running on a computer
- The entity that can be assigned to and executed on a processor
- A unit of activity characterized by the execution of a sequence of instructions, a current state, and an associated set of system resources

Process Elements

- Two essential elements of a process are:
  - Program code
    - which may be shared with other processes that are executing the same program
  - A set of data associated with that code

- When the processor get prepared to execute the program code, we refer to this executable entity as a ***process***

Process Elements

- While the program is executing, this process can be uniquely characterized by a number of elements, including:

identifier

state
priority
program counter

memory pointers
context data
I/O status information
accounting information

Process Control Block

- Contains the process elements
- It is possible to interrupt a running process and later resume execution as if the interruption had not occurred
- Created and managed by the operating system
- Key tool that allows support for multiple processes

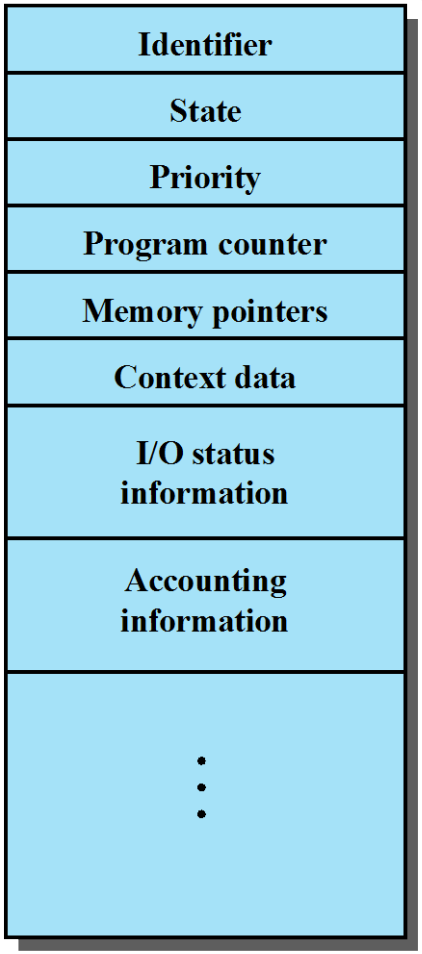


Figure 3.1 Simplified Process Control Block

Process States

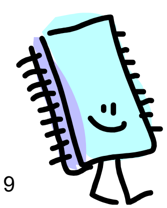
Trace

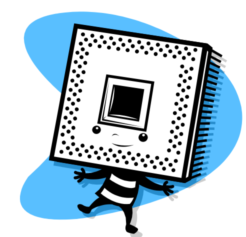
Characterize the behavior of an individual **process** by listing the sequence of instructions that execute for that process

the behavior of the **processor** can be characterized by showing how the traces of the various processes are interleaved

Dispatcher

small program that switches the processor from one process to another





Process Execution

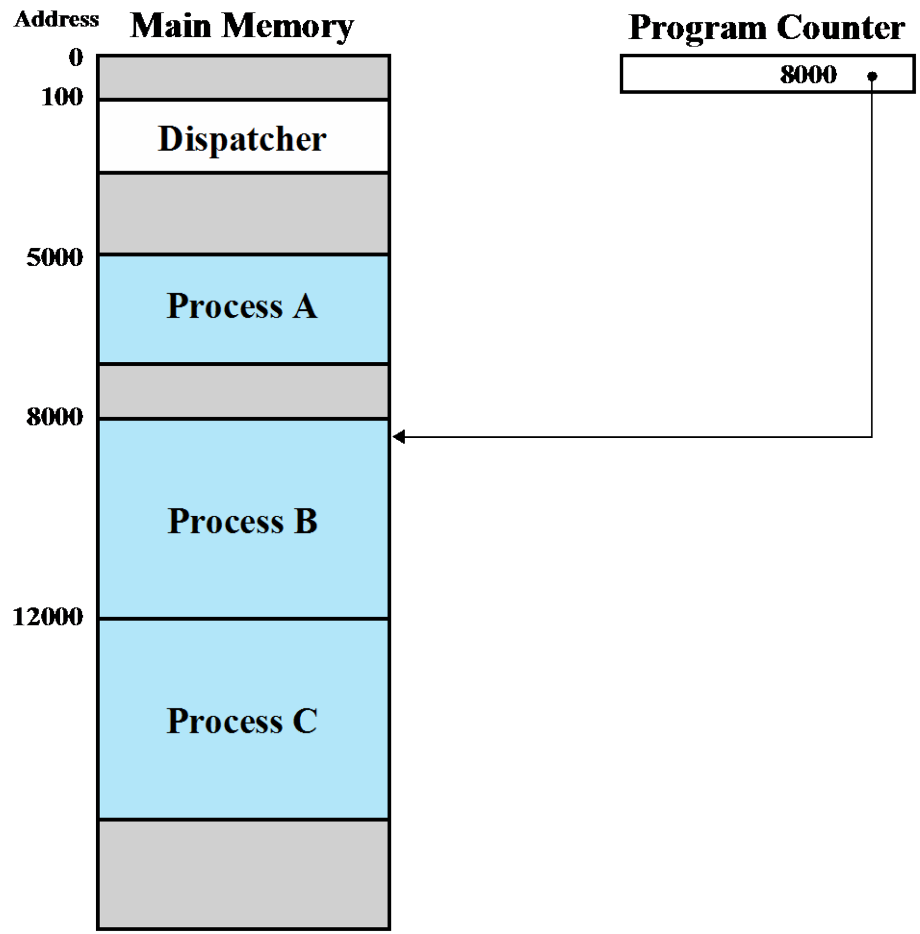
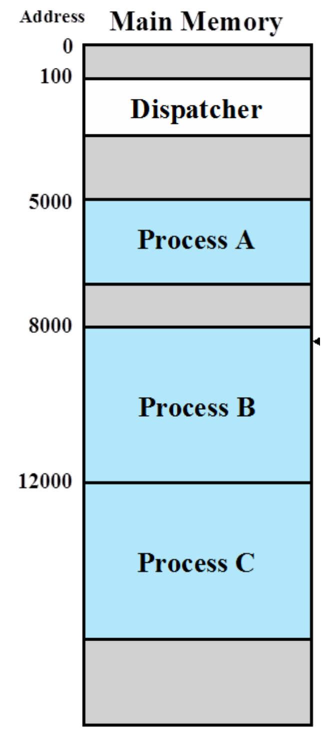


Figure 3.2 Snapshot of Example Execution (Figure 3.4) at Instruction Cycle 13

5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

(a) Trace of Process A **(b) Trace of Process B** **(c) Trace of Process C**



5000 = Starting address of program of Process A
8000 = Starting address of program of Process B
12000 = Starting address of program of Process C

Figure 3.3 Traces of Processes of Figure 3.2

Combined Execution Trace

5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011
(a) Trace of Process A	(b) Trace of Process B	(c) Trace of Process C

5000 = Starting address of program of Process A
8000 = Starting address of program of Process B
12000 = Starting address of program of Process C

1	5000	
2	5001	
3	5002	
4	5003	
5	5004	
6	5005	
-------------------		Timeout
7	100	
8	101	
9	102	
10	103	
11	104	
12	105	
13	8000	
14	8001	
15	8002	
16	8003	
---------------		I/O Request
17	100	
18	101	
19	102	
20	103	
21	104	
22	105	
23	12000	
24	12001	
25	12002	
26	12003	
27	12004	
28	12005	
-------------------		Timeout
29	100	
30	101	
31	102	
32	103	
33	104	
34	105	
35	5006	
36	5007	
37	5008	
38	5009	
39	5010	
40	5011	
-------------------		Timeout
41	100	
42	101	
43	102	
44	103	
45	104	
46	105	
47	12006	
48	12007	
49	12008	
50	12009	
51	12010	
52	12011	
-------------------		Timeout

100 = Starting address of dispatcher program

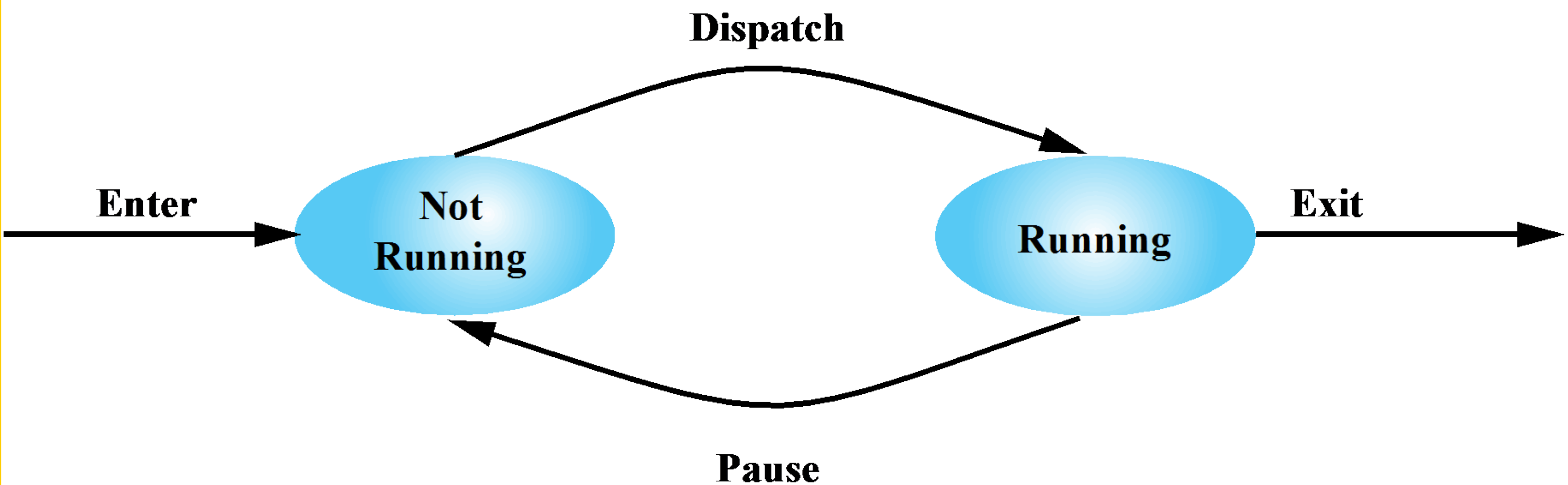
Shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

Process States

- Not all processes are running at the same time.
  - Consider single (uni)-processor:
    - Only one process is “running” at any instant.

- Process in two “states”
  - Running:
    - May have any/all registers modified by CPU.
    - Process is actually doing something.
  - Not-running
    - “idle” - freeze and store for later
    - Program code is not executed

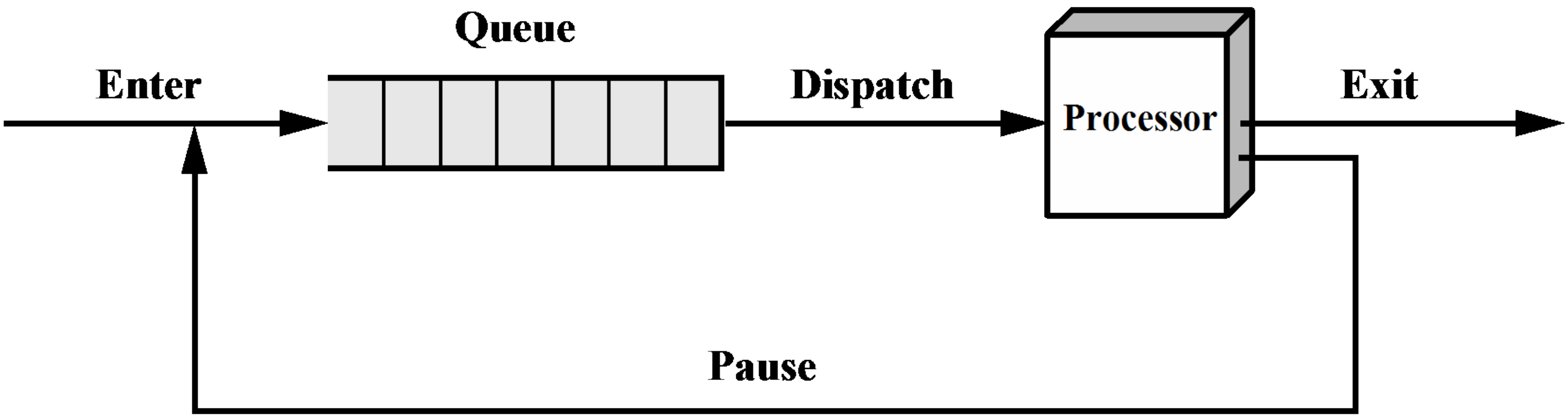
Two-State Process Model



(a) State transition diagram

Process States and State Transitions

- When a user runs a program, processes are created and inserted into the READY list
- A process moves towards the head of the list as other processes complete their turns using a processor
- A process submit to CPU and is said to make a *state transition* from READY to RUNNING state
- The act of assigning a processor to the first process on the ready list is called *dispatching* and is performed by a system entity called the **dispatcher**.
- The operating system set a hardware *interrupting clock* (interval timer) to allow a process to run for a specific time interval or *quantum*



(b) Queuing diagram

Figure 3.5 Two-State Process Model

Creating a Process

- **When?**
  - Something new, which can be self-contained.
    - Batch control
    - User logon
    - OS wants a special “service” (eg. Printer call)
    - Spawned by other process (some “independent and parallel” work)

- **What happens?**
  - OS controls everything
  - Status data structures created
  - Creation state is special
    - Process not in memory yet
    - Process code not being executed yet

Process Creation

- ***Process spawning***
  - when the OS creates a process at the explicit request of another process
- ***Parent process***
  - is the original, creating, process
- ***Child process***
  - is the new process

Process Termination

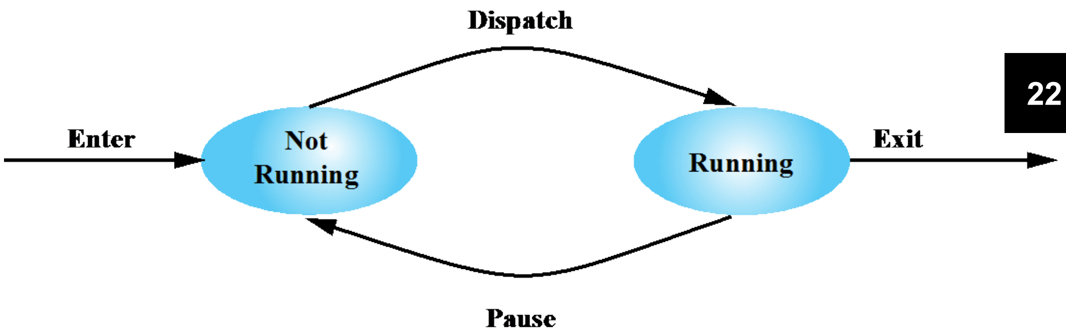
- There must be a means for a process to indicate its completion
- A batch job should include a HALT instruction or an explicit OS service call for termination
- For an interactive application, the action of the user will indicate when the process is completed (e.g. log off, quitting an application)

Terminating a Process

- **When?**
  - **Application finished**
    - User logs out, or quits application
    - Time-limit exceeded
      - processes can only live so long. Why?
  - **Process errors**
    - Memory unavailable or Illegal memory accesses
    - Protection error (Illegal access / usage of a file)
    - Prohibited computation or resulting error
    - I/O errors (eg. file not found)
  - **System reasons**
    - Time overrun
    - Invalid Instruction / privileged instruction
    - OS needs to remove process
    - Parent of process has stopped
    - Parent request

- **What happens?**
- **Is this really necessary?**
  - The process has completed, why put it in any state at all?

One More State?



(a) State transition diagram

- Two states inefficient with I/O
  - Effectively a polling solution.
  - Want to completely ignore processes which are waiting on I/O until I/O is ready.

- Use an extra “blocked” state.
  - Process can’t execute (and won’t be looked at) until some (given) event occurs.

Five-State Process Model

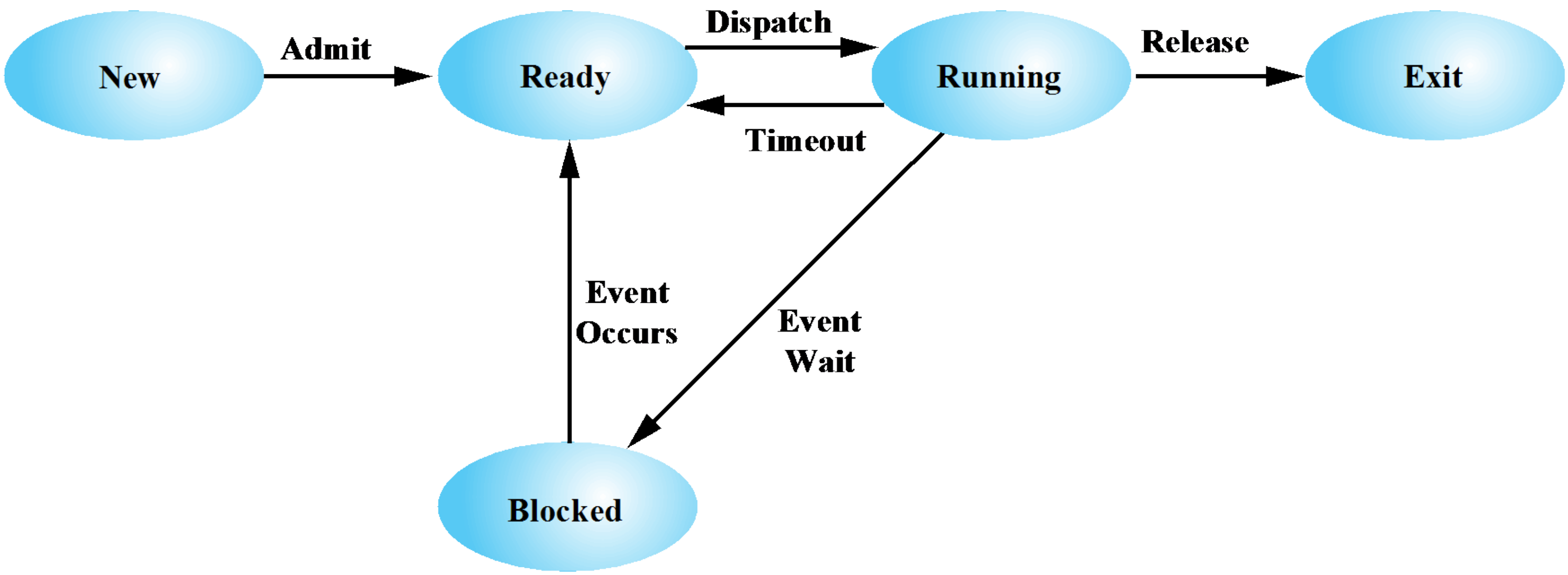


Figure 3.6 Five-State Process Model

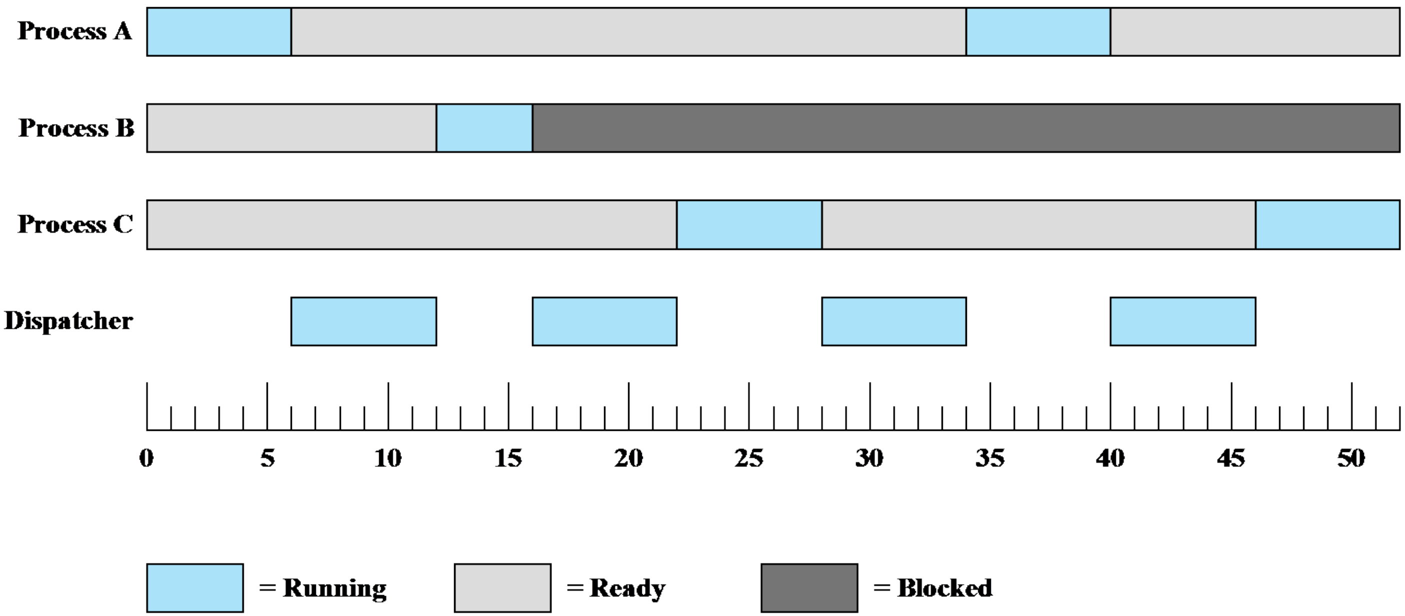


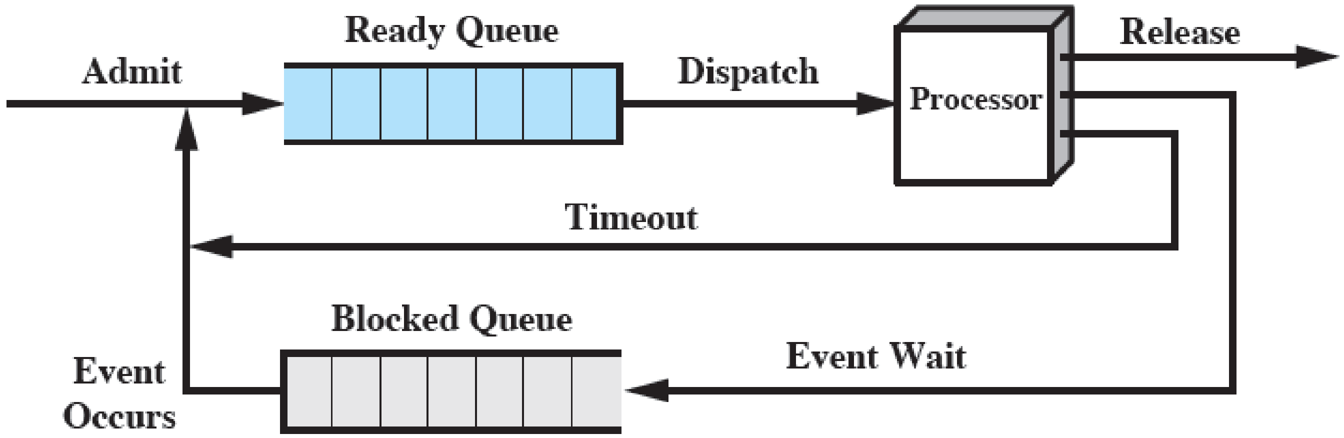
Figure 3.7 Process States for Trace of Figure 3.4

1	5000	27	12004
2	5001	28	12005
3	5002		------------------ Timeout
4	5003	29	100
5	5004	30	101
6	5005	31	102
	------------------ Timeout	32	103
7	100	33	104
8	101	34	105
9	102	35	5006
10	103	36	5007
11	104	37	5008
12	105	38	5009
13	8000	39	5010
14	8001	40	5011
15	8002		------------------ Timeout
16	8003	41	100
	--------------- I/O Request	42	101
17	100	43	102
18	101	44	103
19	102	45	104
20	103	46	105
21	104	47	12006
22	105	48	12007
23	12000	49	12008
24	12001	50	12009
25	12002	51	12010
26	12003	52	12011
			------------------ Timeout

100 = Starting address of dispatcher program

Shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

Physical Implementation

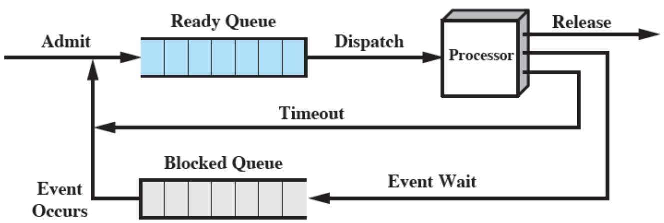


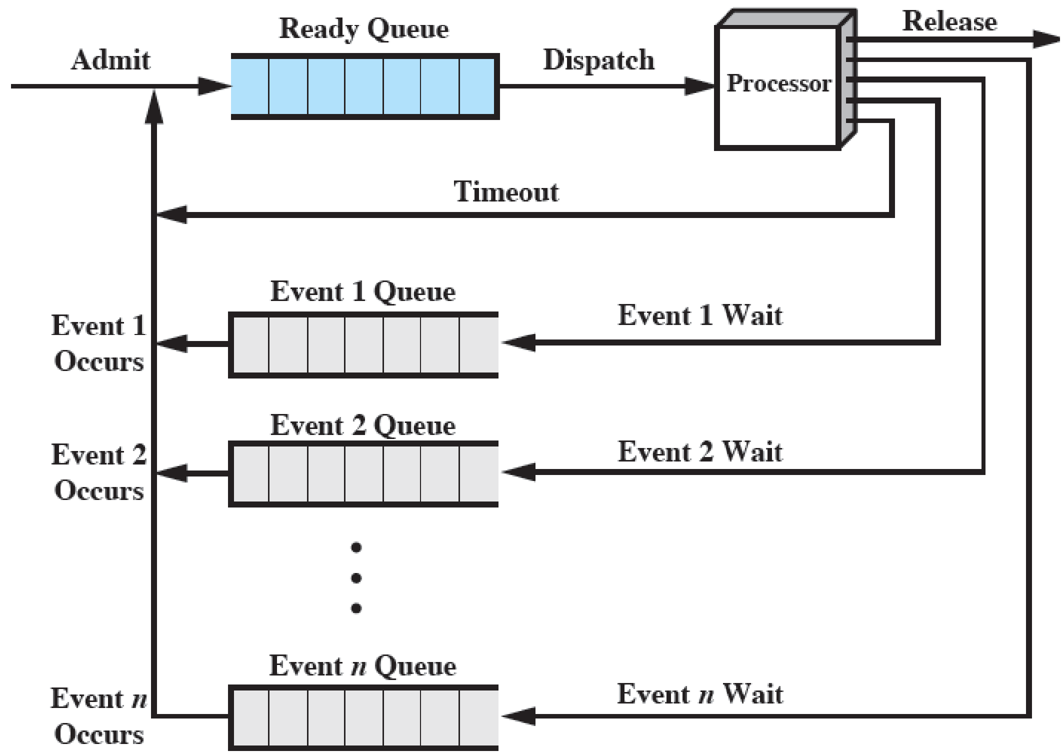
Blocked queue:

- Any "blocking" event puts process on blocked queue.
- **Any problems with this?**
- **Can we improve it?**

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

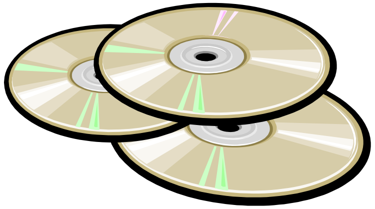
Physical Implementation





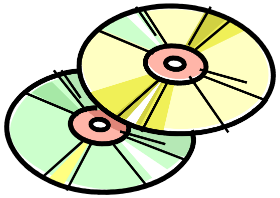
Well...

Consider the following:
- 10 processes in memory (filling it)
- Each process is waiting on I/O
  - Why might this happen?
- What is processor running?



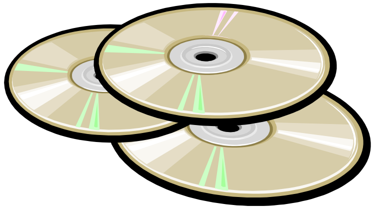
Suggestion:
- Create another state "suspend"
- Suspended processes are removed from memory - save to disk
- Some other process
  - A previously suspended process (from the disk) taken in and run.
  - Or honor a new-process request
  - **Which one to do?**
- If this is so good, **why not suspend all blocked processes?**

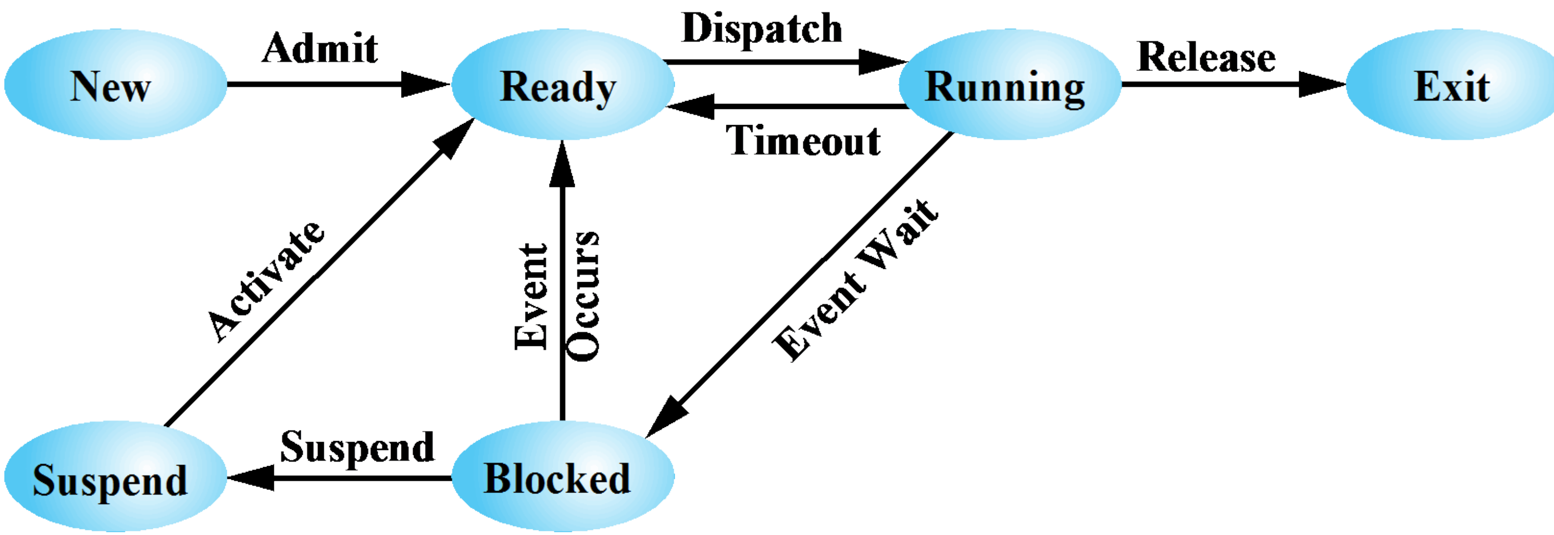
Suspended Processes



– Swapping

- involves moving part or all of a process from main memory to disk
- when none of the processes in main memory is in the Ready state, the OS swaps one of the blocked processes out on to disk into a suspend queue

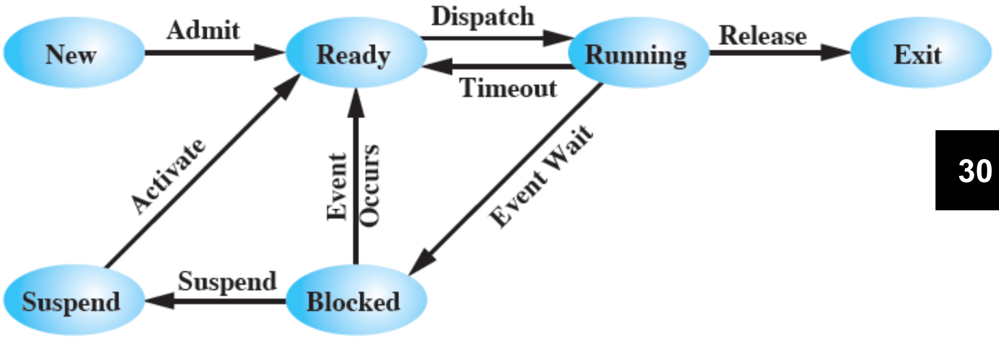




(a) With One Suspend State

Figure 3.9 Process State Transition Diagram with Suspend States

Suspend ideas:



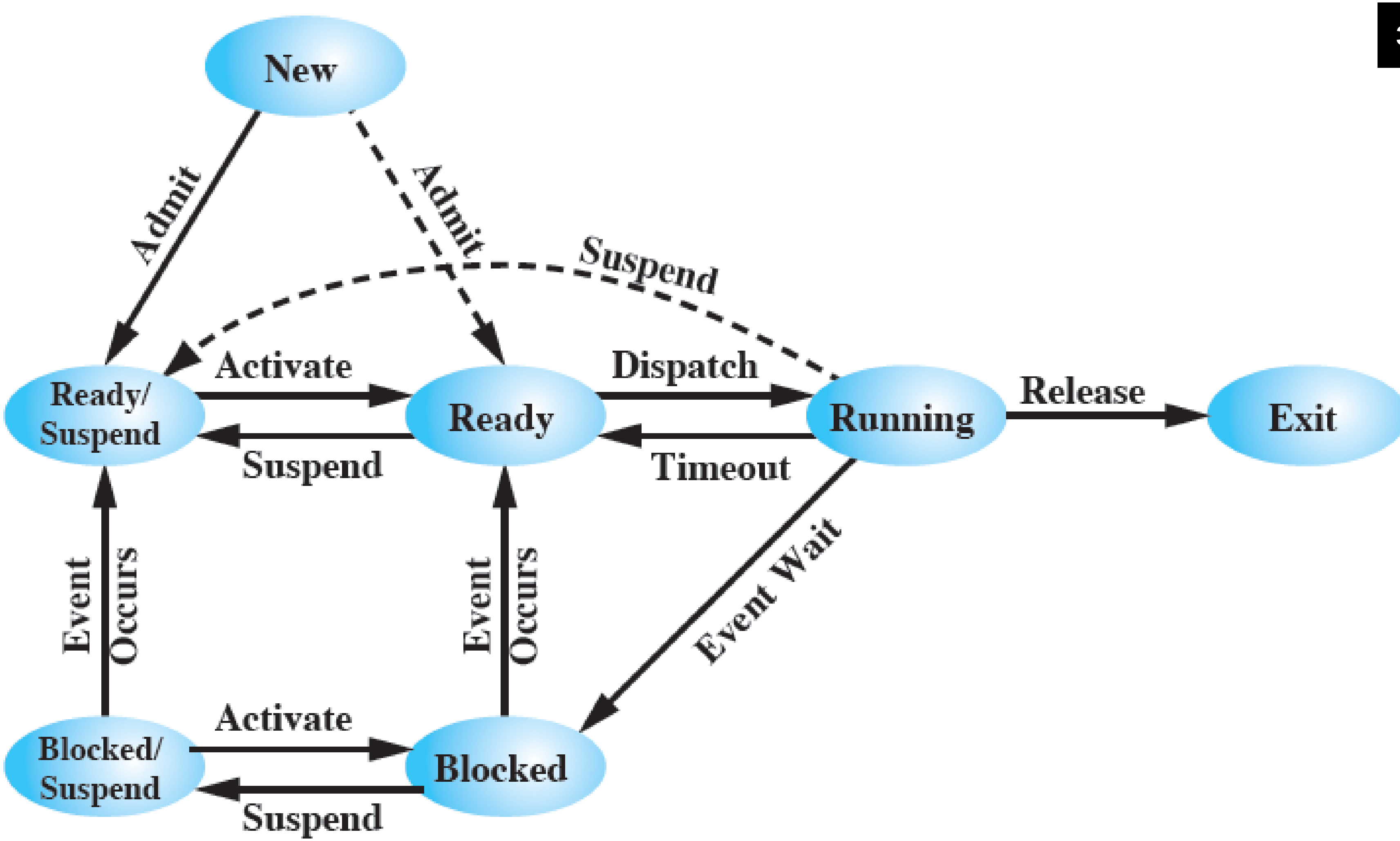
- Get I/O bound processes out of the way
  - Lots of processes can run (briefly) while a slow print job is going

But

- Don't want to activate a process which is still waiting for an event
  - **Why not?**

- Want to know we can activate a process as soon as the event it's waiting for occurs
  - **Why?**

- **What do we do about the suspend state?**


New
Admit
Admit
Suspend
Ready/
Suspend
Activate
Ready
Dispatch
Running
Release
Exit
Suspend
Timeout
Event
Occurs
Event
Occurs
Event Wait
Blocked/
Suspend
Activate
Blocked
Suspend

Reasons for process suspension

Swapping	The OS needs to release sufficient main memory to bring in a process that is ready to execute.
Other OS reason	The OS may suspend a background or utility process or a process that is suspected of causing a problem.
Interactive user request	A user may wish to suspend execution of a program for purposes of debugging or in connection with the use of a resource.
Timing	A process may be executed periodically (e.g., an accounting or system monitoring process) and may be suspended while waiting for the next time interval.
Parent process request	A parent process may wish to suspend execution of a descendent to examine or modify the suspended process, or to coordinate the activity of various descendants.

Characteristics of a Suspended Process

- The process is not immediately available for execution
- The process may or may not be waiting on an event
- The process was placed in a suspended state by an agent: either itself, a parent process, or the OS, for the purpose of preventing its execution
- The process may not be removed from this state until the agent explicitly orders the removal

Process Description

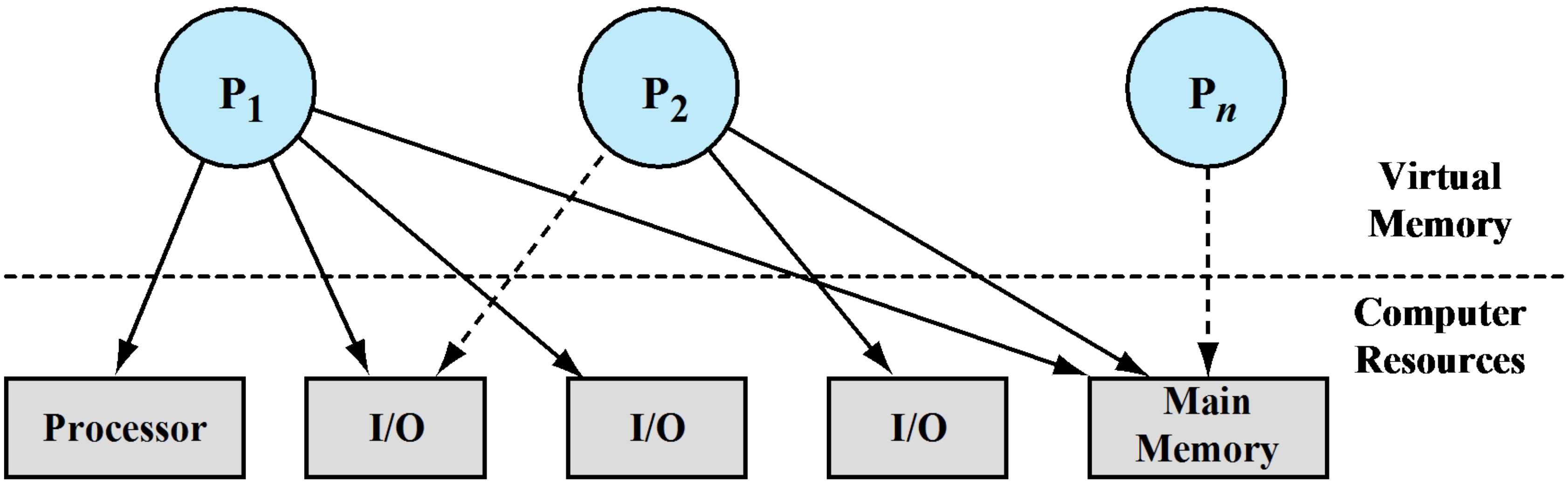


Figure 3.10 Processes and Resources (resource allocation at one snapshot in time)

OS Control Structures

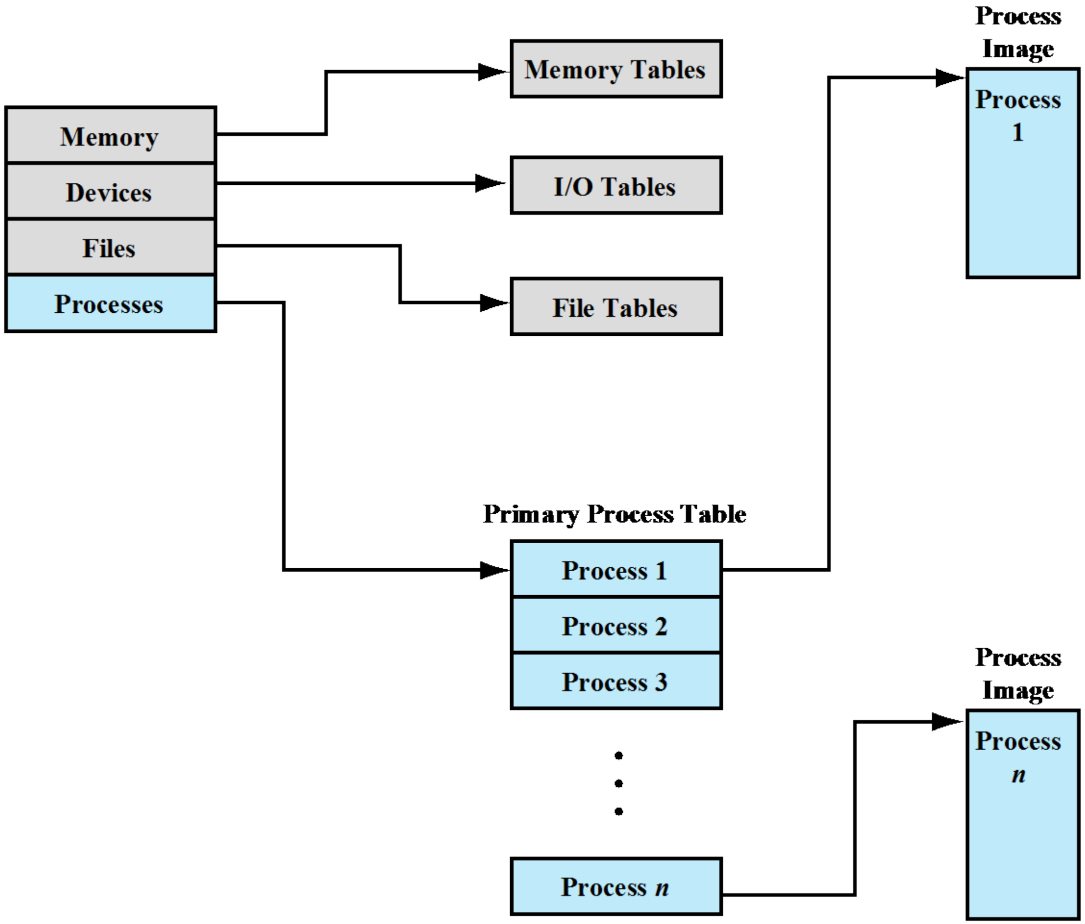


Figure 3.11 General Structure of Operating System Control Tables

OS Control Structures

- **Memory Tables**
  - Used to keep track of both main (real) and secondary (virtual) memory
  - Must include:
    - allocation of main memory to processes
    - allocation of secondary memory to processes
    - protection attributes of blocks of main or virtual memory
    - information needed to manage virtual memory
- **IO Tables**
  - Used by the OS to manage the I/O devices and channels of the computer system
  - At any given time, an I/O device may be available or assigned to a particular process
  - If an I/O operation is in progress, the OS needs to know:
    - the status of the I/O operation
    - the location in main memory being used as the source or destination of the I/O transfer

OS Control Structures

File Table

- These tables provide information about:
  - existence of files
  - location on secondary memory
  - current status
  - other attributes

- Information may be maintained and used by a file management system
  - in which case the OS has little or no knowledge of files

- In other operating systems, much of the detail of file management is managed by the OS itself

OS Control Structures

Process Tables

- Must be maintained to manage processes
- There must be some reference to memory, I/O, and files, directly or indirectly
- The tables themselves must be accessible by the OS and therefore are subject to memory management

Process Control Structures

OS must know two things to manage or control a process:
Process location and process attributes.

- **Process Location**
  - A process must include a program or set of programs to be executed
  - A set of data locations for local and global variables and constants
  - The execution of a program typically involves a stack that is used to keep track of procedure calls and parameter passing between procedures
  - Each process has associated with it a number of attributes that are used by the OS for process control
  - The collection of program, data, stack, and attributes is referred to as the **process image**
  - Process image location will depend on the memory management scheme being used

Typical Elements of a Process Image

User Data

The modifiable part of the user space. May include program data, a user stack area, and programs that may be modified.

User Program

The program to be executed.

Stack

Each process has one or more last-in-first-out (LIFO) stacks associated with it. A stack is used to store parameters and calling addresses for procedure and system calls.

Process Control Block

Data needed by the OS to control the process (see Table 3.5).

Typical Elements of a Process Control Block

Process Identification

Identifiers

Numeric identifiers that may be stored with the process control block include

- Identifier of this process
- Identifier of the process that created this process (parent process)
- User identifier

Processor State Information

User-Visible Registers

A user-visible register is one that may be referenced by means of the machine language that the processor executes while in user mode. Typically, there are from 8 to 32 of these registers, although some RISC implementations have over 100.

Control and Status Registers

These are a variety of processor registers that are employed to control the operation of the processor. These include

- **Program counter:** Contains the address of the next instruction to be fetched
- **Condition codes:** Result of the most recent arithmetic or logical operation (e.g., sign, zero, carry, equal, overflow)
- **Status information:** Includes interrupt enabled/disabled flags, execution mode

Stack Pointers

Each process has one or more last-in-first-out (LIFO) system stacks associated with it. A stack is used to store parameters and calling addresses for procedure and system calls. The stack pointer points to the top of the stack.

(Table is located on page 129 in the textbook)

Typical Elements of a Process Control Block (Cnt..)

Process Control Information

Scheduling and State Information

This is information that is needed by the operating system to perform its scheduling function. Typical items of information:

- **Process state**: Defines the readiness of the process to be scheduled for execution (e.g., running, ready, waiting, halted).
- **Priority:** One or more fields may be used to describe the scheduling priority of the process. In some systems, several values are required (e.g., default, current, highest-allowable)
- **Scheduling-related information:** This will depend on the scheduling algorithm used. Examples are the amount of time that the process has been waiting and the amount of time that the process executed the last time it was running.
- **Event:** Identity of event the process is awaiting before it can be resumed.

Data Structuring

A process may be linked to other process in a queue, ring, or some other structure. For example, all processes in a waiting state for a particular priority level may be linked in a queue. A process may exhibit a parent-child (creator-created) relationship with another process. The process control block may contain pointers to other processes to support these structures.

Interprocess Communication

Various flags, signals, and messages may be associated with communication between two independent processes. Some or all of this information may be maintained in the process control block.

Process Privileges

Processes are granted privileges in terms of the memory that may be accessed and the types of instructions that may be executed. In addition, privileges may apply to the use of system utilities and services.

Memory Management

This section may include pointers to segment and/or page tables that describe the virtual memory assigned to this process.

Resource Ownership and Utilization

Resources controlled by the process may be indicated, such as opened files. A history of utilization of the processor or other resources may also be included; this information may be needed by the scheduler.

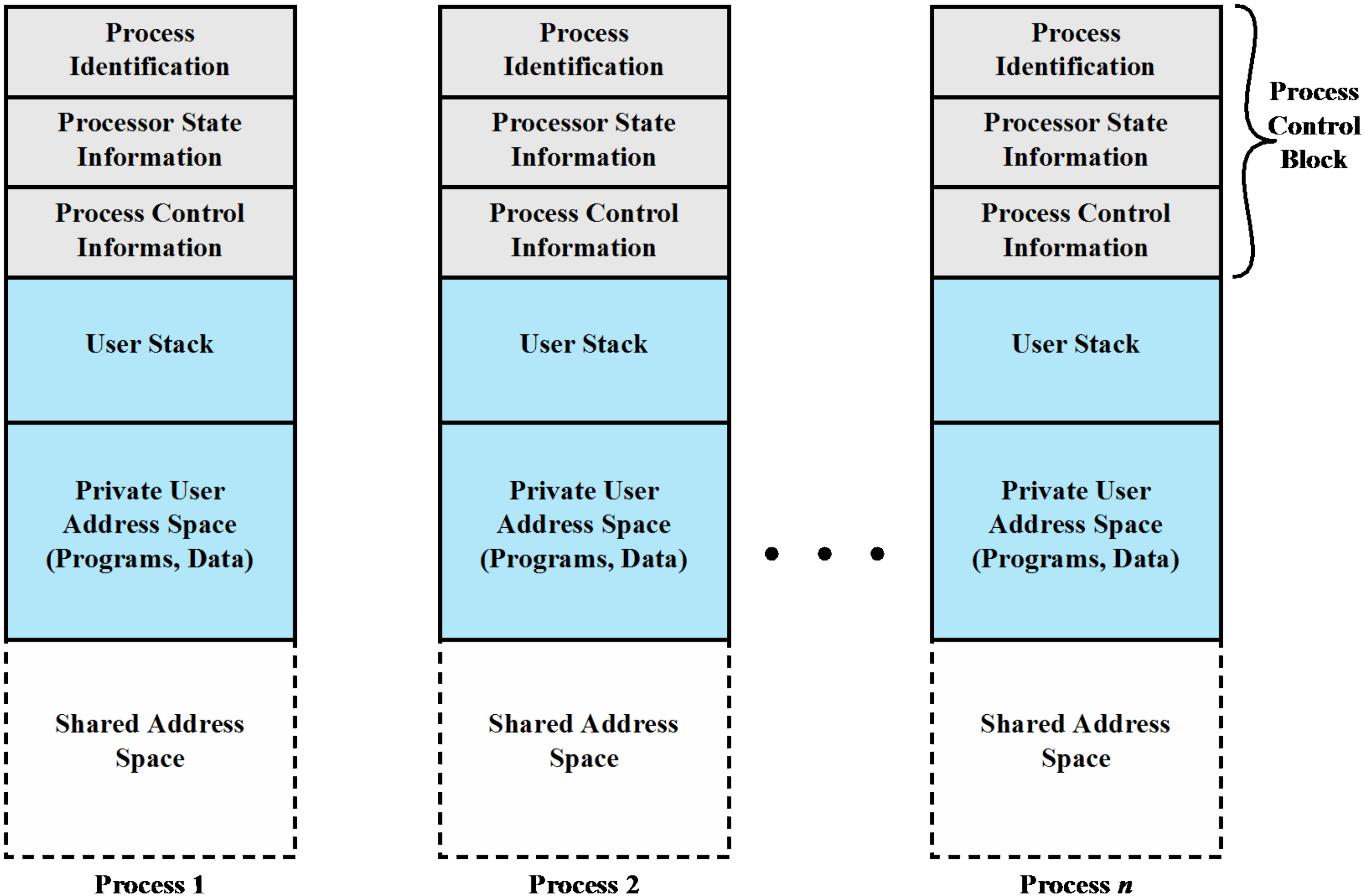
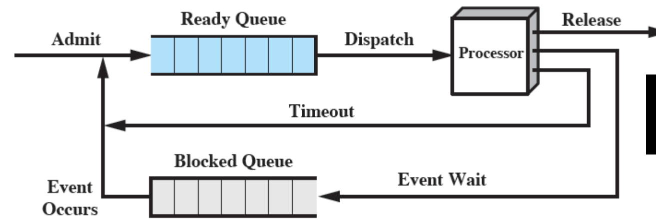


Figure 3.13 User Processes in Virtual Memory



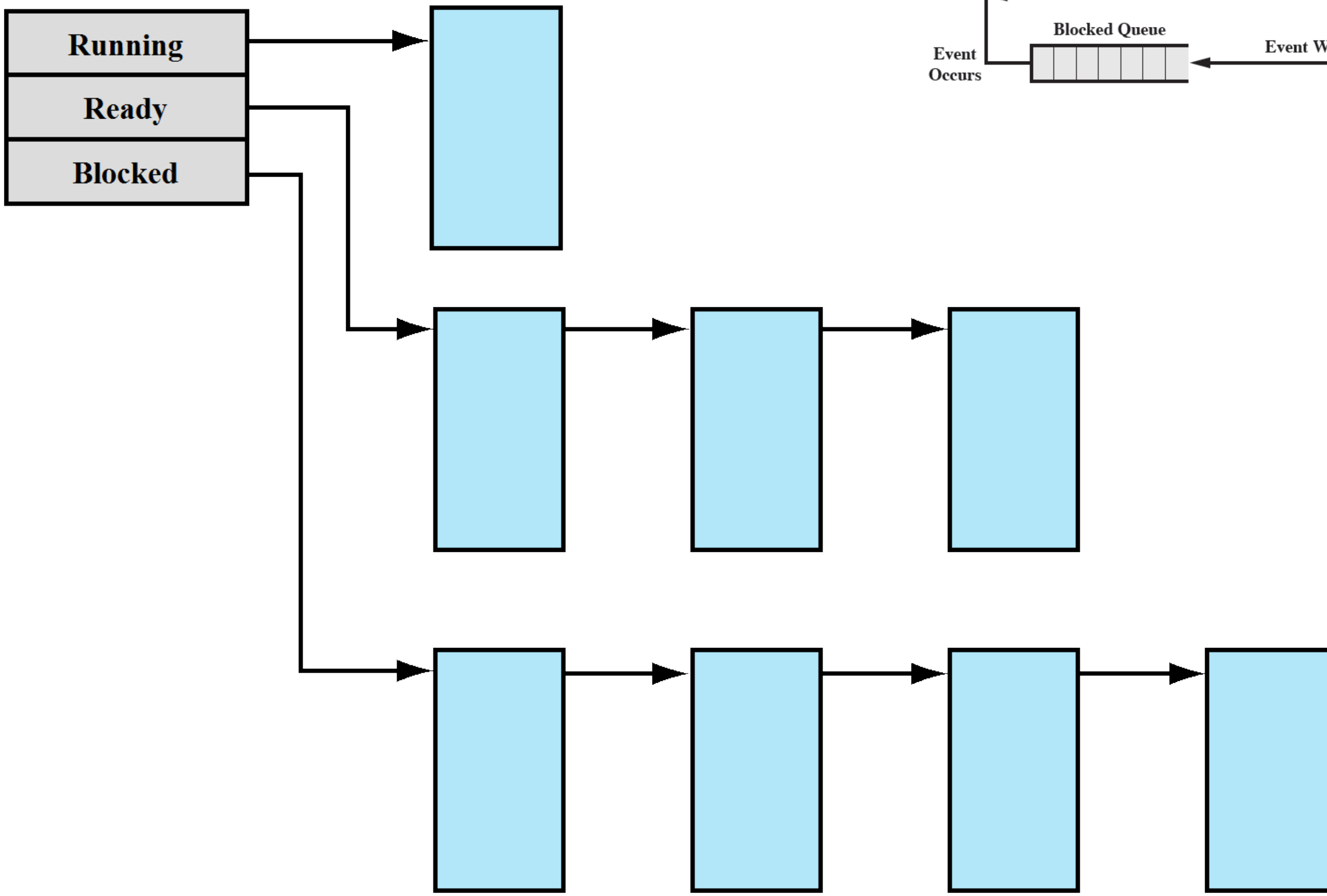


Figure 3.14 Process List Structures

Role of the Process Control Block

- The most important data structure in an OS
  - contains all of the information about a process that is needed by the OS
  - blocks are read and/or modified by virtually every module in the OS
  - defines the state of the OS
- Difficulty is not access, but protection
  - a bug in a single routine could damage process control blocks, which could destroy the system's ability to manage the affected processes
  - a design change in the structure or semantics of the process control block could affect a number of modules in the OS
  - **Handler routine**: to protect PCB, only handler is allowed to read/write PCB

Modes of Execution

User Mode

- less-privileged mode
- user programs typically execute in this mode

System Mode

- more-privileged mode
- also referred to as control mode or kernel mode
- kernel of the operating system

BUT how does the processor know which mode?

Typical Functions of an OS Kernel

Process Management

- Process creation and termination
- Process scheduling and dispatching
- Process switching
- Process synchronization and support for interprocess communication
- Management of process control blocks

Memory Management

- Allocation of address space to processes
- Swapping
- Page and segment management

I/O Management

- Buffer management
- Allocation of I/O channels and devices to processes

Support Functions

- Interrupt handling
- Accounting
- Monitoring

Process Creation

- Once the OS decides to create a new process it:
  - assigns a unique process identifier to the new process
  - allocates space for the process
  - initializes the process control block
  - sets the appropriate linkages
  - creates or expands other data structures

Process Switching

Mechanisms for Interrupting the Execution of a Process

Mechanism	Cause	Use
Interrupt	External to the execution of the current instruction	Reaction to an asynchronous external event
Trap	Associated with the execution of the current instruction	Handling of an error or an exception condition
Supervisor call	Explicit request	Call to an operating system function

System Interrupts

- **Interrupt**
  - Due to some sort of event that is external to and independent of the currently running process
    - » clock interrupt
    - » I/O interrupt
    - » memory fault
- **Trap**
  - An error or exception condition generated within the currently running process
  - OS determines if the condition is fatal
    - » moved to the Exit state and a process switch occurs
    - » action will depend on the nature of the error
- **Supervisor Call**
  - Call from a program being executed
  - An I/O request to open a file
    - » Transfer to a routine that is part of OS

Mode Switching

- **If no interrupts are pending the processor:**
  - proceeds to the fetch stage and fetches the next instruction of the current program in the current process
- **If an interrupt is pending the processor:**
  - sets the program counter to the starting address of an interrupt handler program
  - switches from user mode to kernel mode so that the interrupt processing code may include privileged instructions

Change of Process State

- The steps in a full process switch are:

1. save the context of the processor
2. update the process control block of the process currently in the Running state
3. move the process control block of this process to the appropriate queue
4. select another process for execution
5. update the process control block of the process selected
6. update memory management data structures
7. restore the context of the processor to that which existed at the time the selected process was last switched out

If the currently running process is to be moved to another state (Ready, Blocked, etc.), then the OS must make substantial changes in its environment

Execution of the Operating System

Non-process Kernel

Execution within User Processes

Process based OS

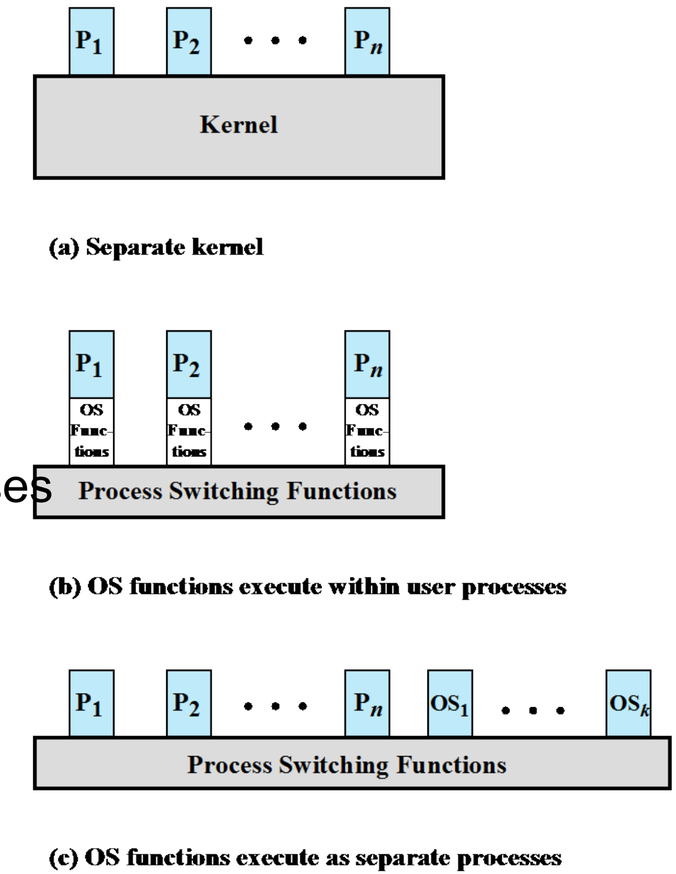
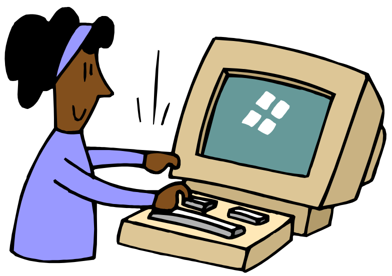


Figure 3.15 Relationship Between Operating System and User Processes



Execution Within User Processes

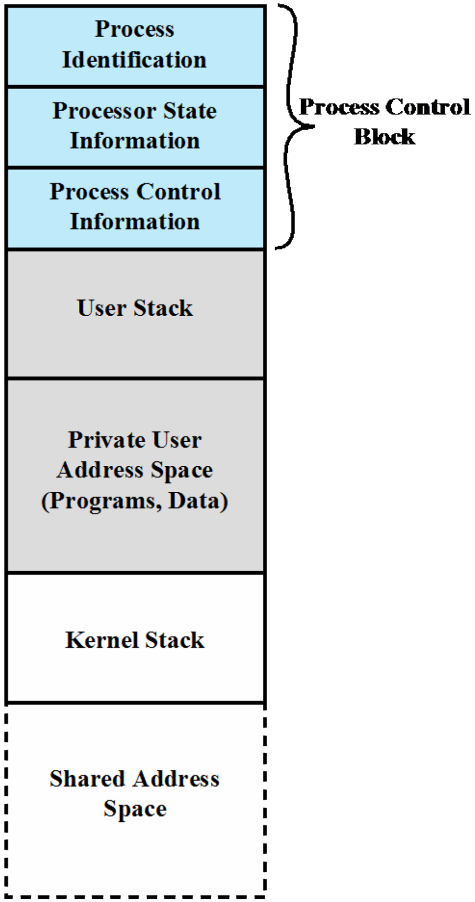
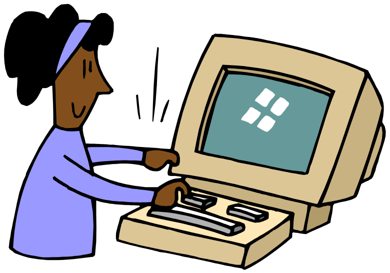


Figure 3.16 Process Image: Operating System Executes Within User Space



Processes and Threads

Resource Ownership

- Process includes a virtual address space to hold the process image
- Time to time allocated ownership of resources
  - the OS performs a protection function to prevent unwanted interference between processes with respect to resources

Scheduling/Execution

- Follows an execution path that may be interleaved with other processes
  - a process has an execution state (Running, Ready, etc.) and a dispatching priority and is scheduled and dispatched by the OS

Processes and Threads

- The unit of dispatching is referred to as a ***thread*** or ***lightweight process***
- The unit of resource ownership is referred to as a ***process*** or ***task***
- ***Multithreading -*** The ability of an OS to support <u>multiple</u>, <u>concurrent paths of execution</u> within a single process

Single VS Multi-Threaded Approaches

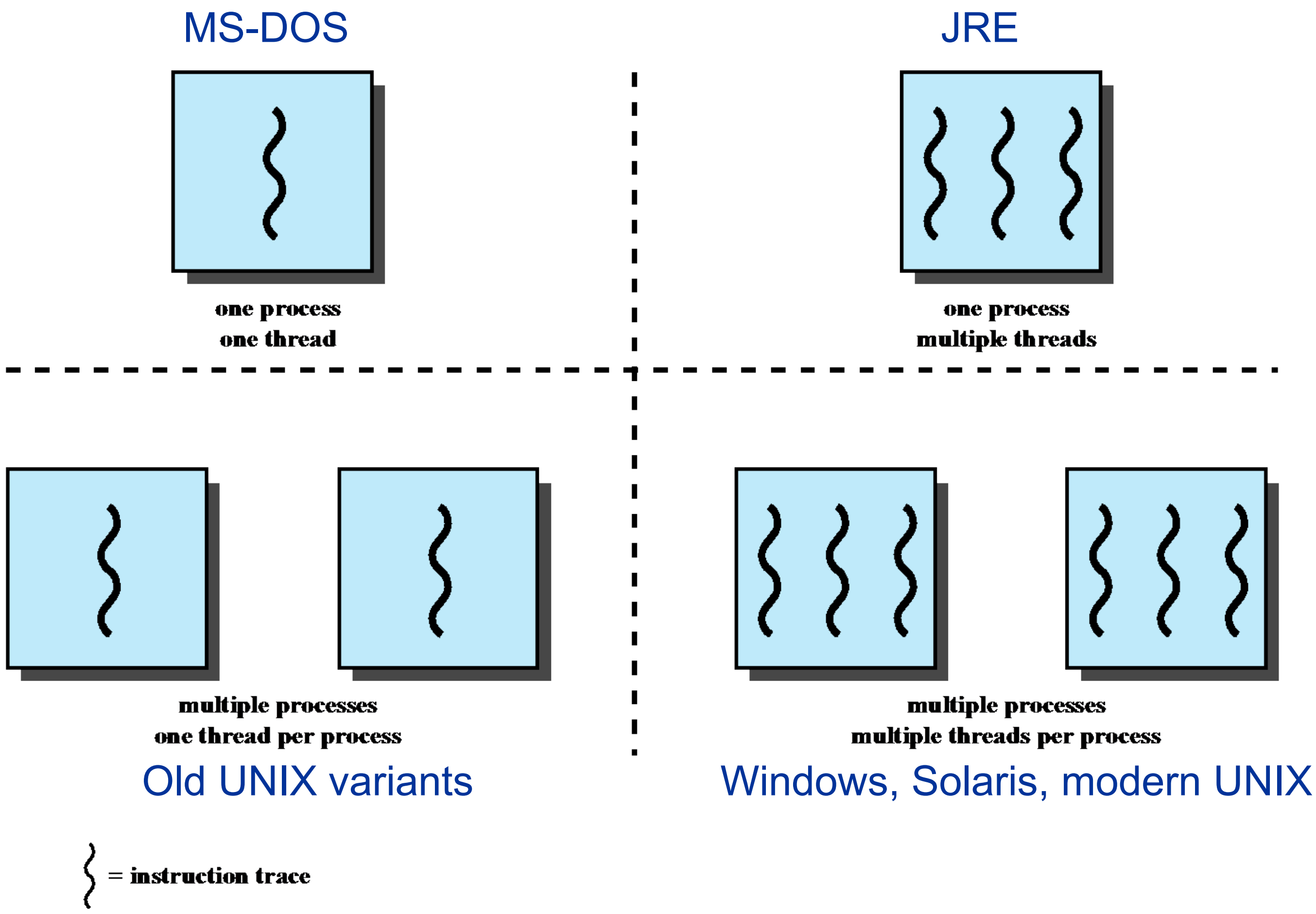


Figure 4.1 Threads and Processes

Multi-threaded environment: Process and Threads

Processes

- The unit or resource allocation and a unit of protection
- A virtual address space that holds the process image
- Protected access to:
  - processors, files, I/O and other processes

Threads

- One or more threads in a process
- Each thread has:
  - an execution state (Running, Ready, etc.)
  - saved thread context when not running
  - an execution stack
  - some per-thread static storage for local variables
  - access to the memory and resources of its process (all threads of a process share this)

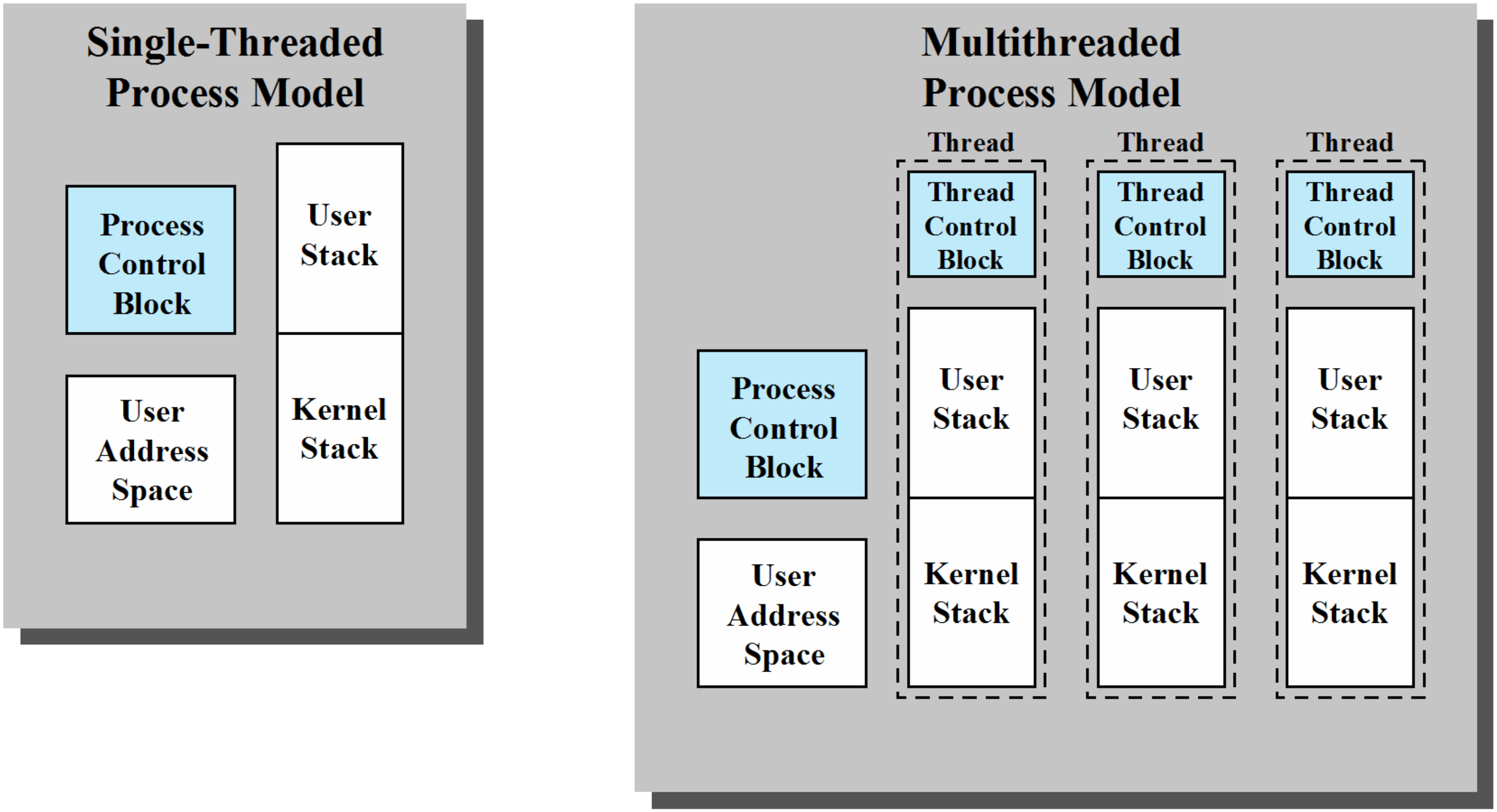


Figure 4.2 Single Threaded and Multithreaded Process Models

Key Benefits of Threads

- Takes less time to create a new thread than a process
  - In some systems 10 times faster
- Less time to terminate a thread than a process
- Switching between two threads (within the same process) takes less time than switching between processes
- Threads enhance efficiency in communication between programs

Thread Use in a Single-Processor System

- Foreground and background work
- Asynchronous processing
- Speed of execution
- Modular program structure

Threads VS Process Management

- In an OS that supports threads, scheduling and dispatching is done on a thread basis
- Most of the state information dealing with execution is maintained in thread-level data structures

- Process level action affect all of the threads in a process
  - Suspending a process involves suspending all threads of the process
  - Termination of a process terminates all threads within the process

Thread Execution States

- The key states for a thread are:
  - Running
  - Ready
  - Blocked
- Thread operations associated with a change in thread state are:
  - Spawn
  - Block
  - Unblock
  - Finish
- Does blocking of a thread block a process?

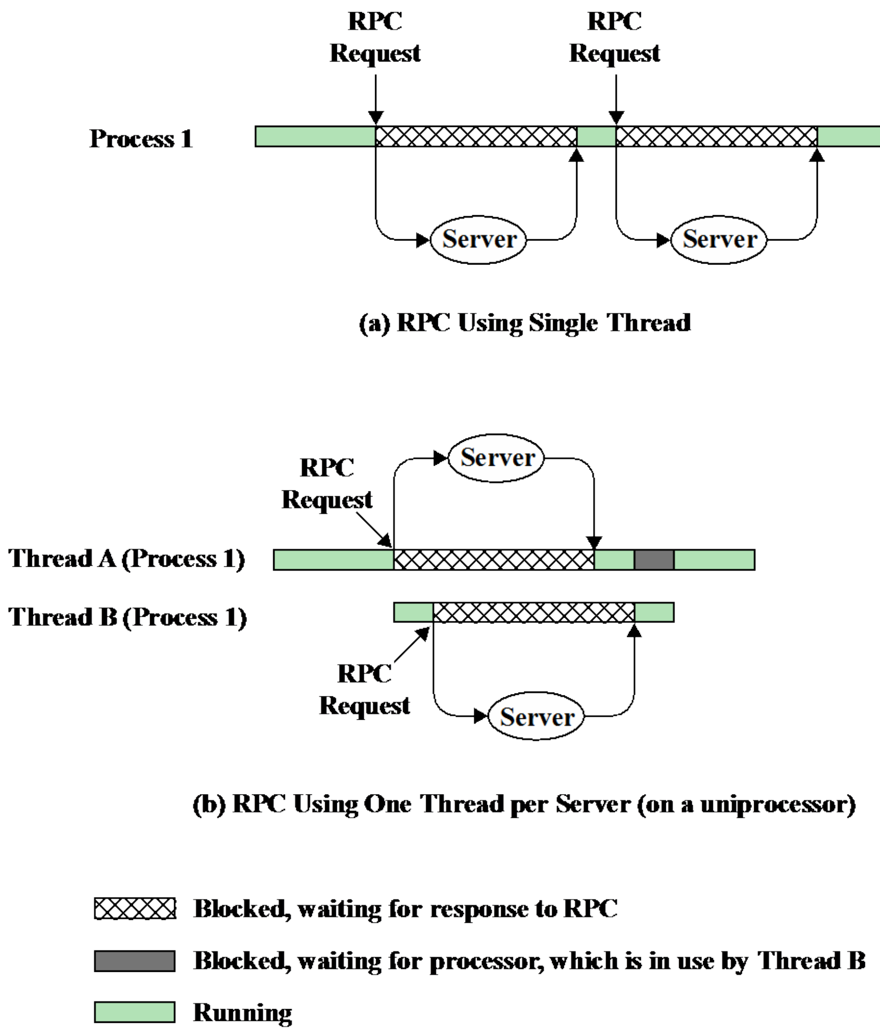


Figure 4.3 Remote Procedure Call (RPC) Using Threads

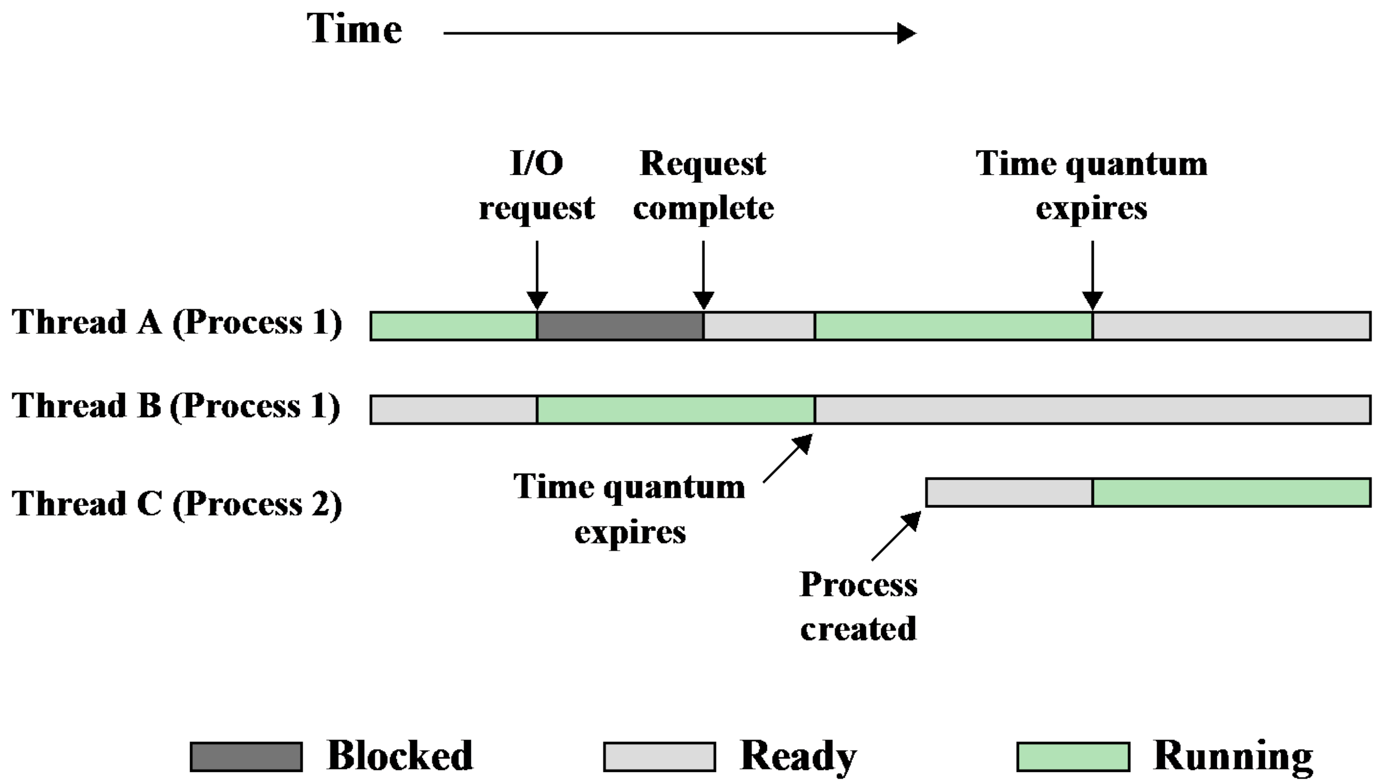


Figure 4.4 Multithreading Example on a Uniprocessor

Thread Synchronization

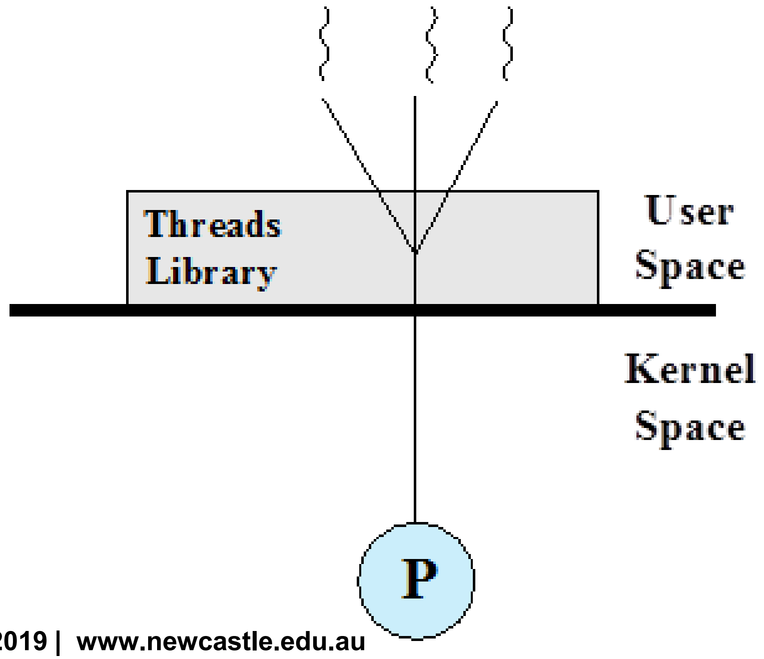
- It is necessary to synchronize the activities of the various threads
  - all threads of a process share the same address space and other resources
  - any alteration of a resource by one thread affects the other threads in the same process

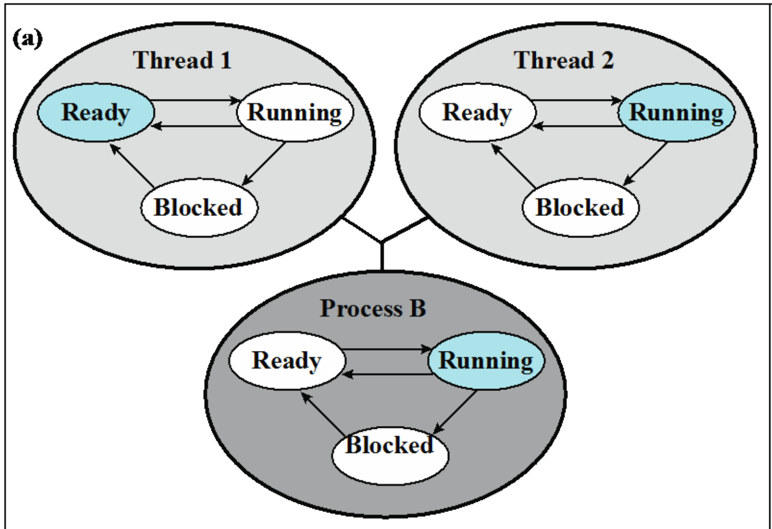
Types of Threads

- User Level Thread (ULT)
- Kernel level Thread (KLT)
- Combined Approach

User Level Threads (ULT)

- All thread management is done by the application
- The kernel is not aware of the existence of threads





Thread 2 issues a system call

Process B expires its time slice

Thread 2 is blocked in for cooperation from Thread 1

Colored state
is current state

Figure 4.6 Examples of the Relationships Between User-Level Thread States and Process States

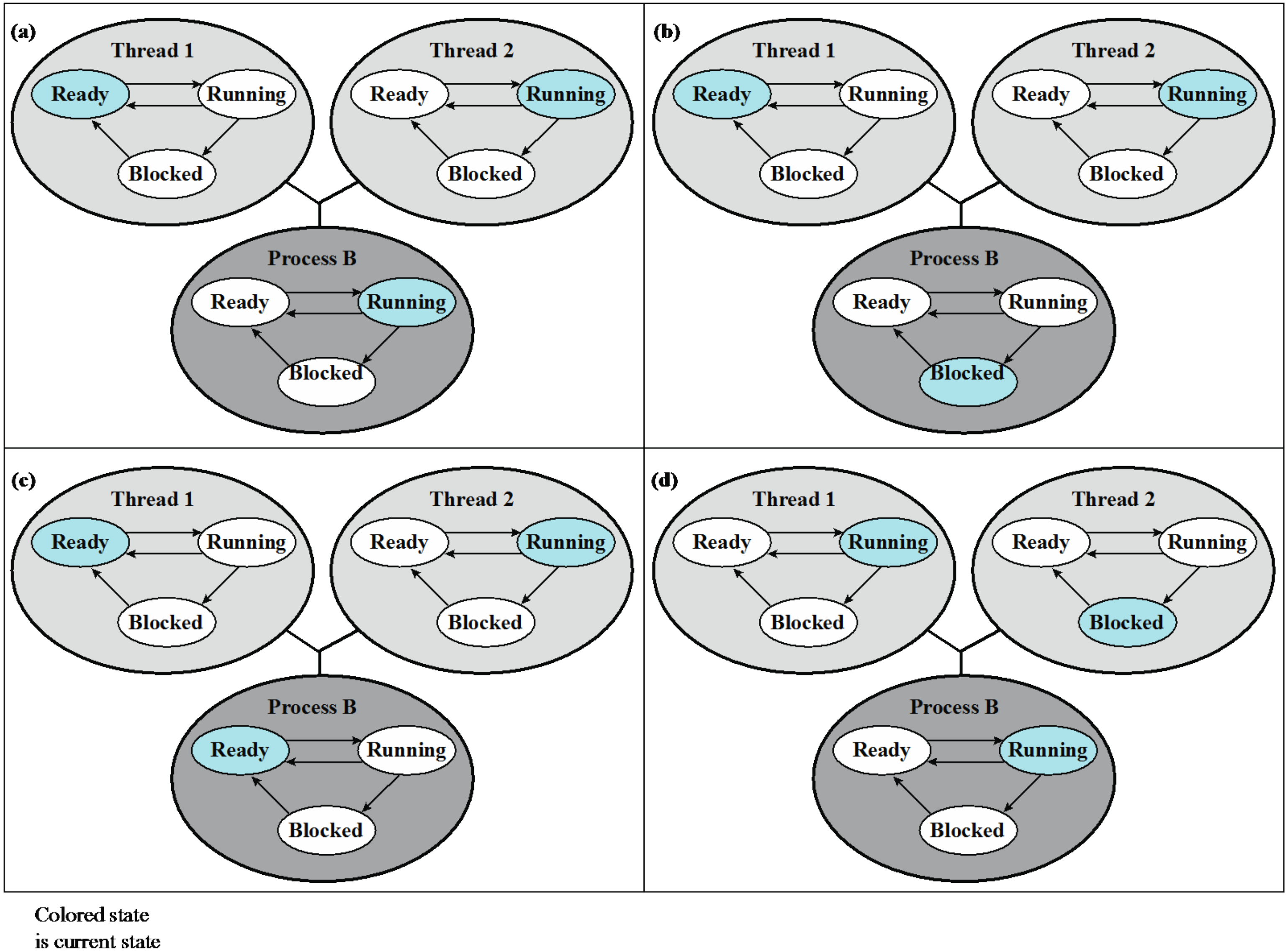


Figure 4.6 Examples of the Relationships Between User-Level Thread States and Process States

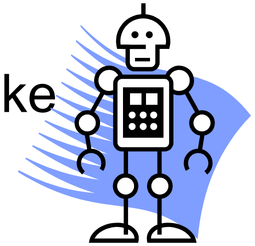
Pros and Cons of ULTs

Advantages

- Thread switching does not require kernel mode privileges
- Scheduling can be application specific
- ULTs can run on any OS

Disadvantages

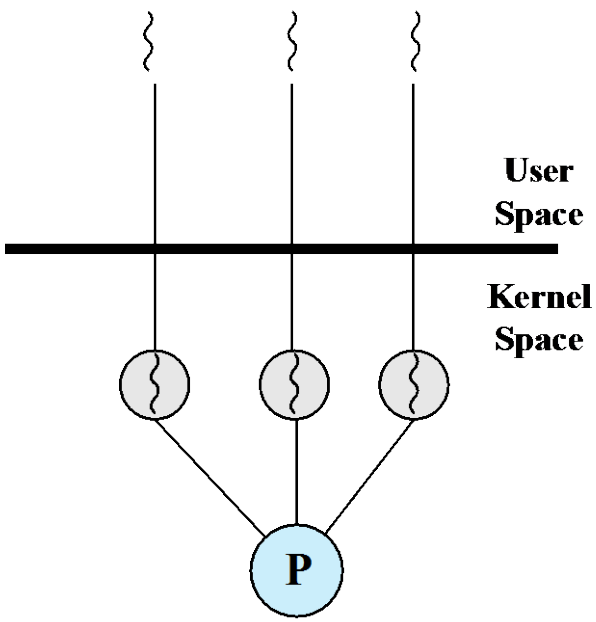
- In a typical OS many system calls are blocking
    - as a result, when a ULT executes a system call, not only is that thread blocked, but all of the threads within the process are blocked
- In a pure ULT strategy, a multithreaded application cannot take advantage of multiprocessing



THE UNIVERSITY OF NEWCASTLE AUSTRALIA

Kernel-Level Threads (KLTs)

- Thread management is done by the kernel
  - no thread management is done by the application
    - Windows is an example of this approach



(b) Pure kernel-level

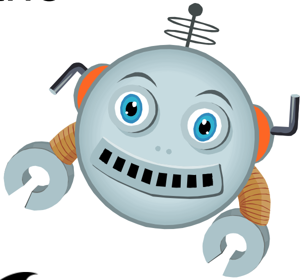
Pros and Cons of KLTs

Advantages

- The kernel can simultaneously schedule multiple threads from the same process on multiple processors
- If one thread in a process is blocked, the kernel can schedule another thread of the same process
- Kernel routines can be multithreaded

Disadvantages

- The transfer of control from one thread to another within the same process requires a mode switch to the kernel



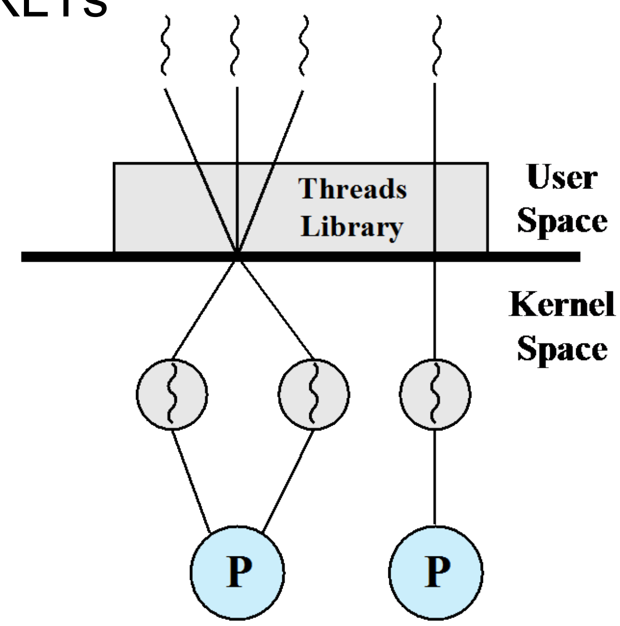
ULT vs KLTs

Operation	User-Level Threads	Kernel-Level Threads	Processes
Null Fork	34	948	11,300
Signal Wait	37	441	1,840

- Might be application specific as well!

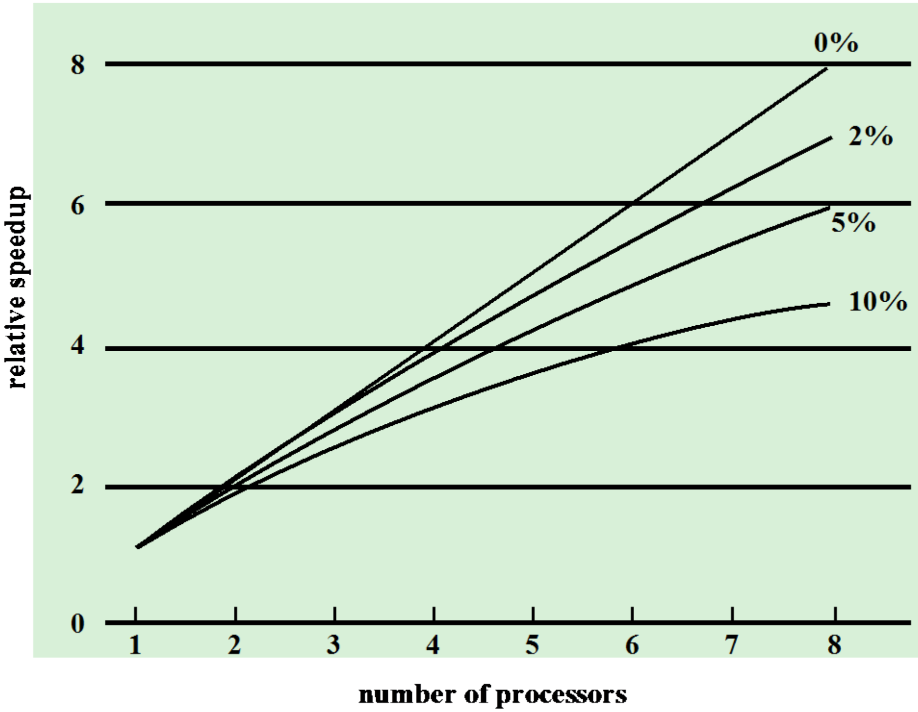
Combined Approaches (ULT/KLT)

- Thread creation is done in the user space
- Bulk of scheduling and synchronization of threads is by the application
- Multiple ULTs from an application is mapped onto some (fewer or smaller) number of KLTs
- Solaris is an example



(c) Combined

Multicore and Multithreading Performance

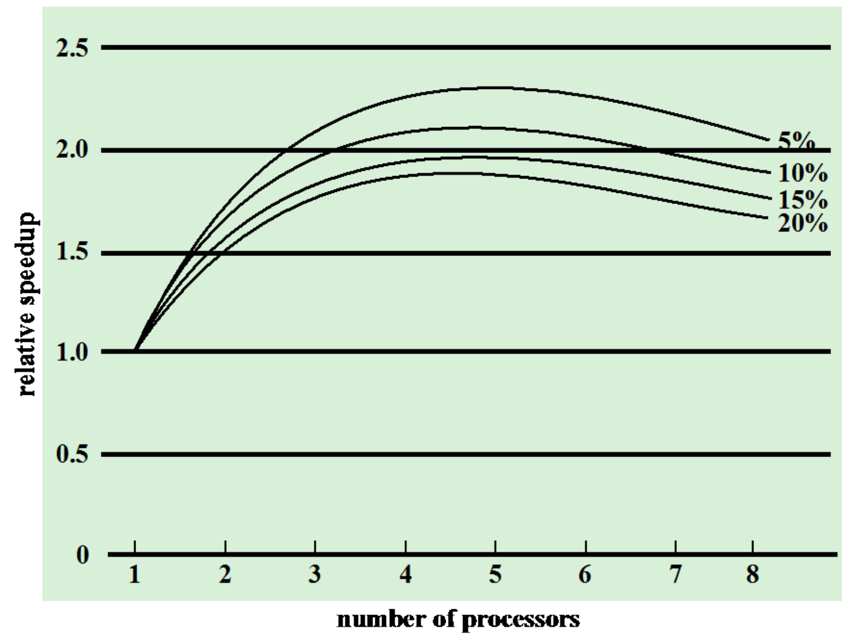


(a) Speedup with 0%, 2%, 5%, and 10% sequential portions

Amdahl's Law:

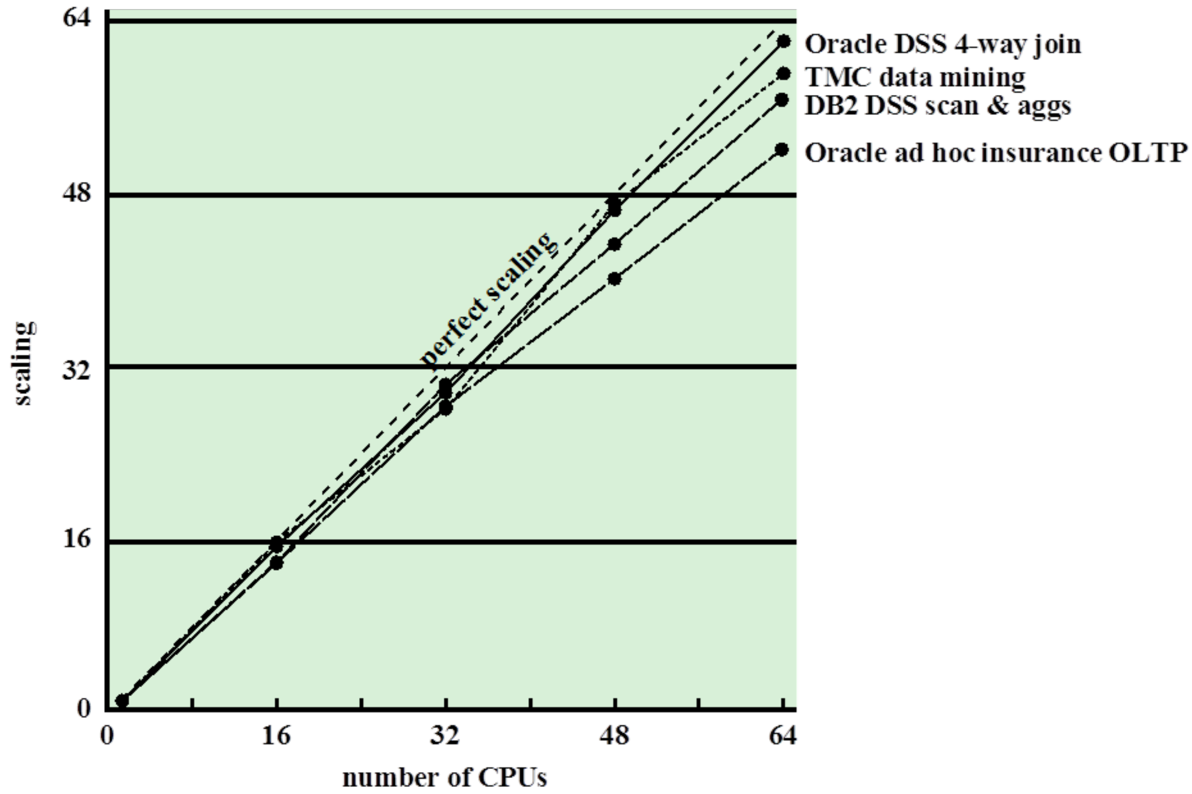
$$Speedup = \frac{1}{(1-f) + \frac{f}{N}}$$

f: fraction of code that is infinitely parallelizable
(1-f): fraction of code that is inherently serial
N: number of parallel processors



(b) Speedup with overheads

Multicore and Multithreading Performance



Summary

- A process is an entity consisting of two essential elements: program code and a set of data
- Processes Control Block is a data structure that stores all relevant information related to a process
- A process passes through different states throughout its lifetime
  - Two-state process model
  - Five-state model
  - Seven-state model
- The principle function of OS is to create, manage and terminate processes
- OS needs to keep track of various kinds of information to manage and control a process
- Process is related to resource ownership and Thread is related to program execution
- Multithreaded system allows multiple concurrent thread inside a process for efficiency
- Threading can be supported at user-level or at kernel level

References

- **Operating Systems – Internal and Design Principles**
  - By William Stallings
- Chapter 3, 4