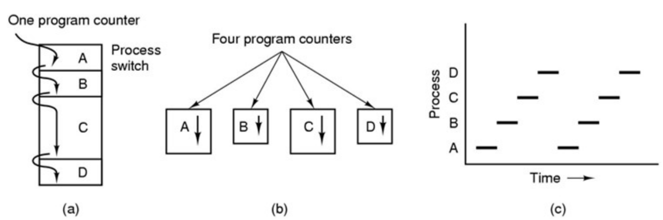
Chapter 2

Processes and Threads

2.1 Processes

Processes

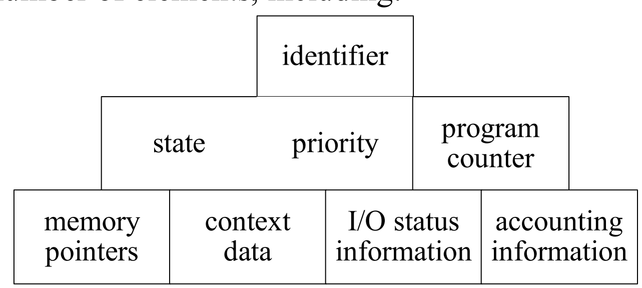
The Process Model



- Multiprogramming of four programs
- Conceptual model of 4 independent, sequential processes
- Only one program active at any instant

Process Elements

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



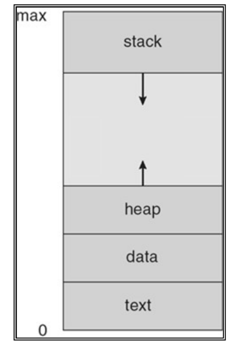
Processes
Process Concept

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
- Process – a program in execution; process execution must progress in sequential fashion
- A process resources includes:
  - Address space (text segment, data segment)
  - CPU (virtual)
    - program counter
    - registers
    - stack
  - Other resource (open files, child processes…)

5

5

Processes
Process in Memory



6

6

Processes
Process Creation

Principal events that cause process creation

1. System initialization
2. Execution of a process creation system Call
3. User request to create a new process
4. Initiation of a batch job

7

7

Process Creation

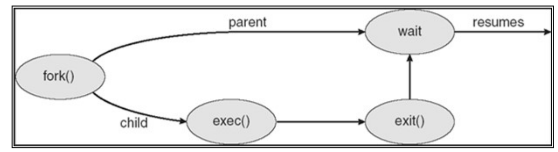
- Once the OS decides to create a new process it:
  - assigns a unique process identifier to the new process
  - allocates space for the process
  - initializes the process control block
  - sets the appropriate (thích hợp) linkages
  - creates or expands other data structures

8

Processes
Process Creation (2)

- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - **fork** system call creates new process
  - **exec** system call used after a **fork** to replace the process' memory space with a new program

Processes
Process Creation (3) : Example



Processes
Process Termination

Conditions which terminate processes

1. Normal exit (voluntary)
2. Fatal error (voluntary)
3. Error exit (involuntary)
4. Killed by another process (involuntary)

Processes
Process Hierarchies

- Parent creates a child process, child processes can create its own process
- Forms a hierarchy
  - UNIX calls this a "process group"
- Windows has no concept of process hierarchy
  - all processes are created equal

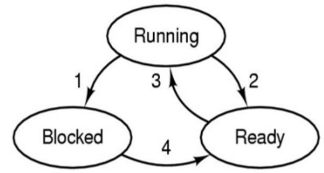
Processes
Process Hierarchies

```
duybinh@ubuntu:~$ pstree -p
systemd(1)─┬─ModemManager(740)─┬─{ModemManager}(855)
           │                   └─{ModemManager}(871)
           ├─NetworkManager(875)─┬─dhclient(1005)
           │                     ├─{NetworkManager}(924)
           │                     └─{NetworkManager}(926)
           ├─VGAuthService(583)
           ├─accounts-daemon(748)─┬─{accounts-daemon}(854)
           │                      └─{accounts-daemon}(870)
           ├─acpid(735)
           ├─agetty(991)
           ├─anacron(743)
           ├─apache2(5741)─┬─apache2(5744)
           │               ├─apache2(5745)
           │               ├─apache2(5746)
           │               ├─apache2(5747)
           │               ├─apache2(5748)
           │               └─apache2(5749)
           ├─avahi-daemon(767)───avahi-daemon(853)
           ├─boltd(3782)─┬─{boltd}(3783)
           │             └─{boltd}(3785)
           ├─colord(1744)─┬─{colord}(1745)
           │              └─{colord}(1747)
           ├─cron(717)
           ├─cups-browsed(889)─┬─{cups-browsed}(901)
           │                   └─{cups-browsed}(902)
           ├─cupsd(719)
```

13

13

Processes
Process States



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

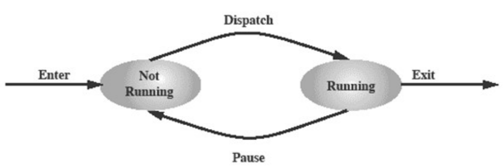
- Possible process states
  - running
  - blocked
  - ready
- Transitions between states shown

14

14

Two-State Process Model

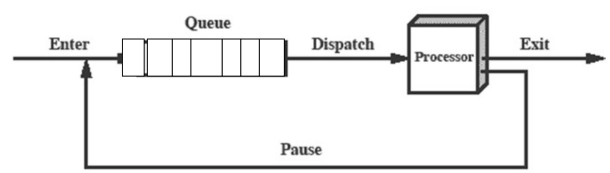
- A process may be in one of two states:
  - running
  - not-running



(a) State transition diagram

15

Queuing Diagram



(b) Queuing diagram

16

Five-State Process Model

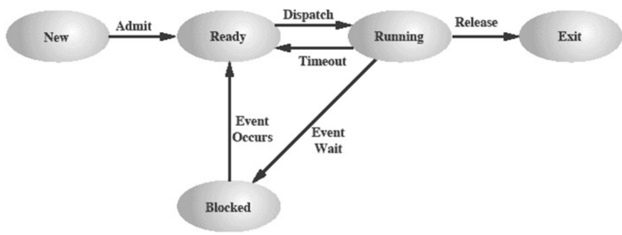


Figure 3.6 Five-State Process Model

17

Process States for Trace of Figure 3.4

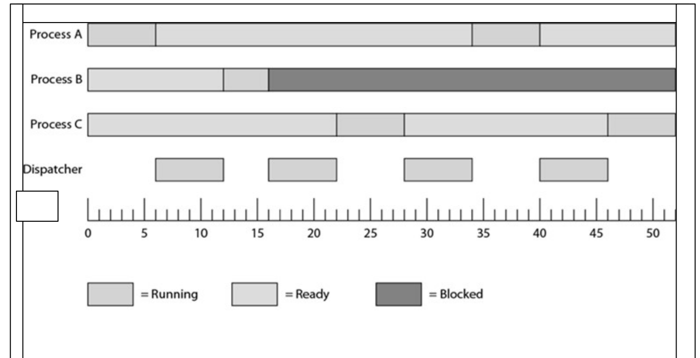
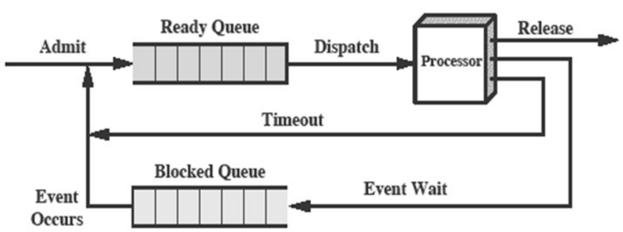


Figure 3.7 Process States for Trace of Figure 3.4

18

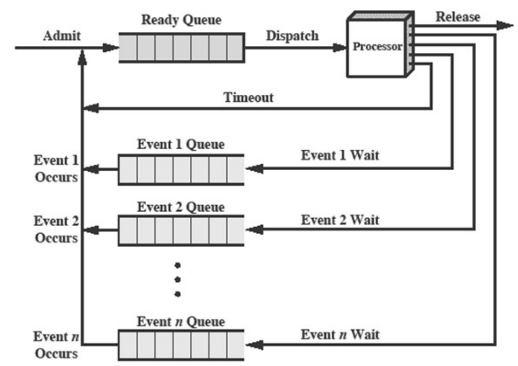
Using Two Queues



(a) Single blocked queue

19

Multiple Blocked Queues



(b) Multiple blocked queues

20

Suspended (treo/trì hoãn) Processes

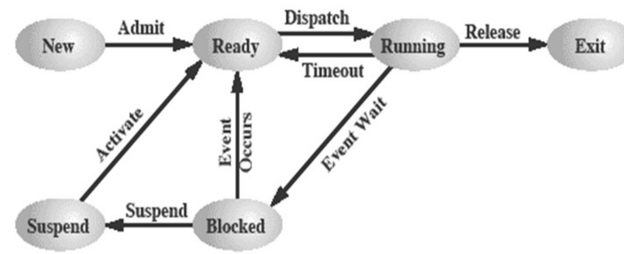
– Swapping

– Involves (bao gồm) moving part of 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

21

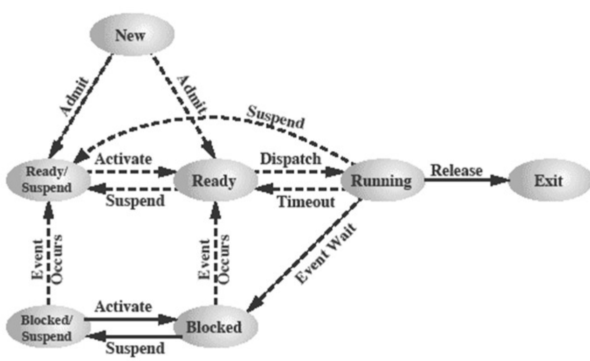
One Suspend State



(a) With One Suspend State

22

Two Suspend States

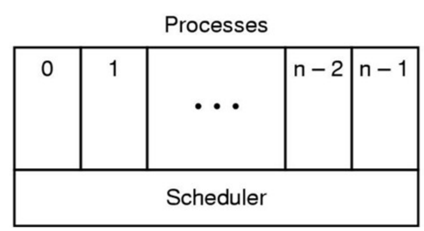


(b) With Two Suspend States

23

Processes

Process States

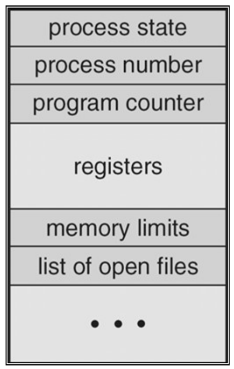


- Lowest layer of process-structured OS
  - handles interrupts, scheduling
- Above that layer are sequential processes

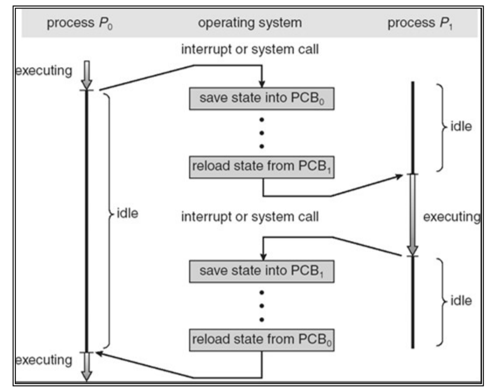
24

24

Processes
Process Control Block (PCB)



Processes
context switch



OS Control Tables

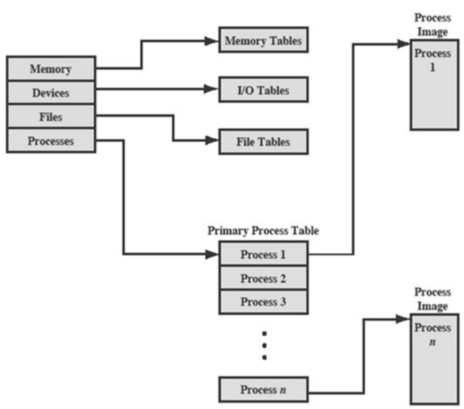


Figure 3.11 General Structure of Operating System Control Tables

Memory Tables

- Used to keep track of both main (real) and secondary (virtual) memory
- Processes are maintained (duy trì/giữ) on secondary memory using some sort of virtual memory or simple swapping mechanism

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

I/O 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~~ (đang tiến hành), 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

29

File Tables

- Information may be maintained and used by a file management system
  - in which case the OS has little or no knowledge (nhận biết) of files
- In other operating systems, much of the detail of file management is managed by the OS itself

These tables provide information about:

- existence of files
- location on secondary memory
- current status
- other attributes

30

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 (do đó/vì vậy) are subject to memory management

31

Processes
Implementation of Processes

Process management	Memory management	File management
Registers Program counter Program status word Stack pointer Process state Priority Scheduling parameters Process ID Parent process Process group Signals Time when process started CPU time used Children's CPU time Time of next alarm	Pointer to text segment Pointer to data segment Pointer to stack segment	Root directory Working directory File descriptors User ID Group ID

Fields of a process table entry

32

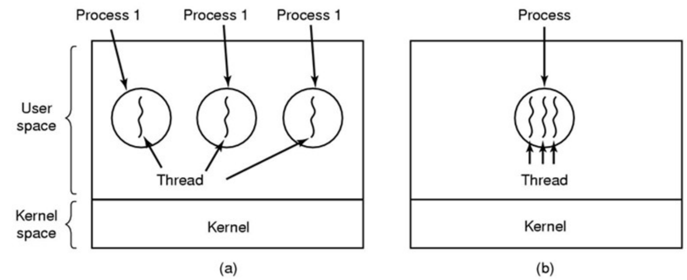
32

2.2 Threads

33

33

Threads
The Thread Model



(a) Three processes each with one thread
(b) One process with three threads

34

34

Threads
Process with single thread

- A process:
  - Address space (text section, data section)
  - Single thread of execution
    - program counter
    - registers
    - Stack
  - Other resource (open files, child processes…)

35

35

Threads
Process with multiple threads

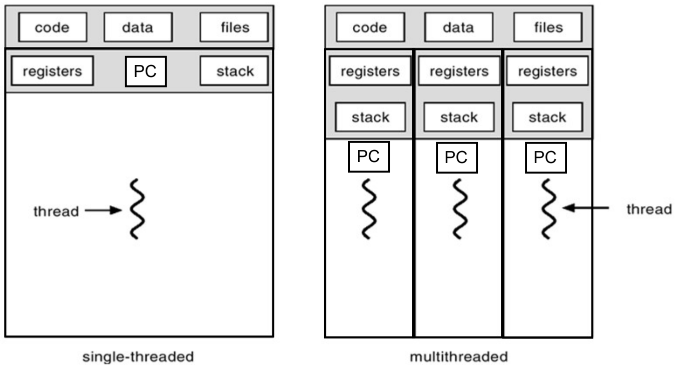
Multiple threads of execution in the same environment of process

- Address space (text section, data section)
- Multiple threads of execution, each thread has private set:
  - program counter
  - registers
  - stack
- Other resource (open files, child processes…)

36

36

Threads
Single and Multithreaded Processes



37

37

Threads
Items shared and Items private

Per process items	Per thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

- Items shared by all threads in a process
- Items private to each thread

38

38

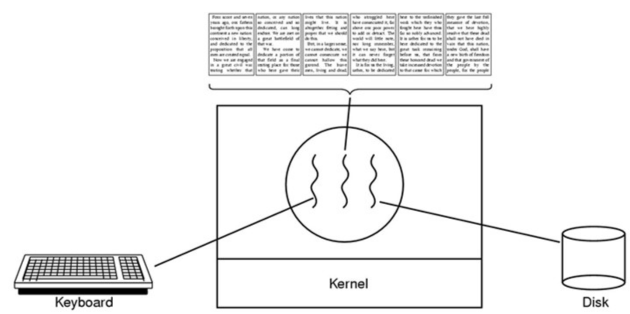
Threads
Benefits

- Responsiveness
- Resource Sharing
- Economy
- Utilization of Multiprocessor Architectures

39

39

Threads
Thread Usage (1)

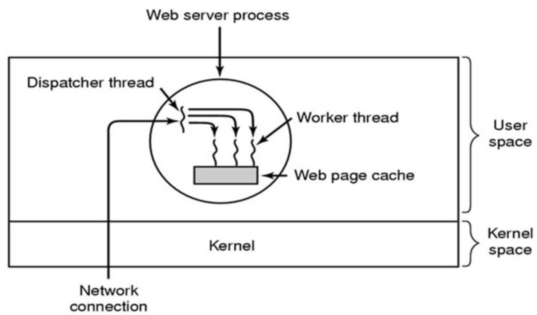


A word processor with three threads

40

40

Threads
Thread Usage (2)



A multithreaded Web server

41

41

Threads
Thread Usage (3)

```
while (TRUE) {
  get_next_request(&buf);
  handoff_work(&buf);
}
        (a)
```

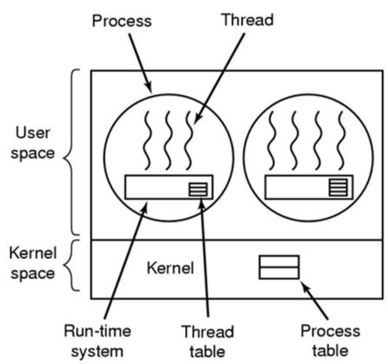
```
while (TRUE) {
  wait_for_work(&buf)
  look_for_page_in_cache(&buf, &page);
  if (page_not_in_cache(&page)
      read_page_from_disk(&buf, &page);
  return_page(&page);
}
        (b)
```

- Rough outline of code for previous slide
  - (a) Dispatcher thread
  - (b) Worker thread

42

42

Threads
Implementing Threads in User Space (1)



A user-level threads package

43

43

Threads
Implementing Threads in User Space (2)

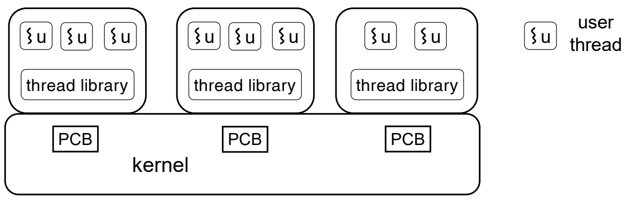
- Thread library, (run-time system) in user space
  - thread_create
  - thread_exit
  - thread_wait
  - thread_yield (chịu nhường) (to voluntarily (tự nguyện) give up the CPU)
- Thread control block (TCB) (Thread Table) stores states of user thread (program counter, registers, stack)
- Kernel does not know the present of user thread

44

44

Threads
Implementing Threads in User Space (3)

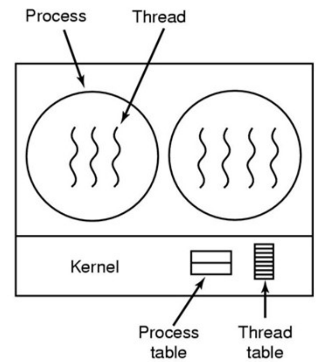
- Traditional OS provide only one "kernel thread" presented by PCB for each process.
  - *Blocking problem:* If one user thread is blocked ->the kernel thread is blocked, -> all other threads in process are blocked.



45

45

Threads
Implementing Threads in the Kernel (1)



A threads package managed by the kernel

46

46

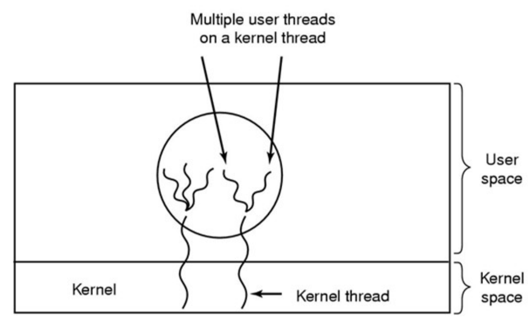
Threads
Implementing Threads in the Kernel (2)

- Multithreading is directly supported by OS:
  - Kernel manages processes and threads
  - CPU scheduling for thread is performed in kernel
- Advantage of multithreading in kernel
  - Is good for multiprocessor architecture
  - If one thread is blocked does not cause the other thread to be blocked.
- Disadvantage of Multithreading in kernel
  - Creation and management of thread is slower

47

47

Threads
Hybrid Implementations



Multiplexing user-level threads onto kernel-level threads

48

48

Relationship Between Threads and Processes

Threads:Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH
1:M	A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.	Ra (Clouds), Emerald
M:N	Combines attributes of M:1 and 1:M cases.	TRIX

Table 4.2 Relationship between Threads and Processes

49

2.3 Scheduling

50

50

Scheduling

Introduction to Scheduling (1)

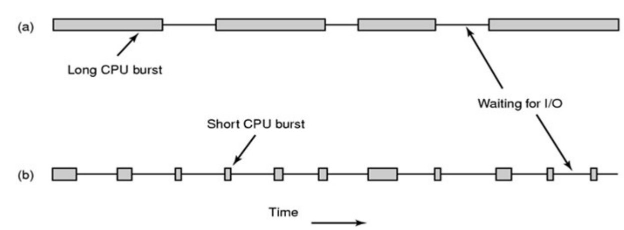
- Maximum CPU utilization obtained (thu được) with multiprogramming
- CPU–I/O Burst (mảnh/khoảng) Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait
- CPU burst distribution

51

51

Scheduling

Introduction to Scheduling (2)



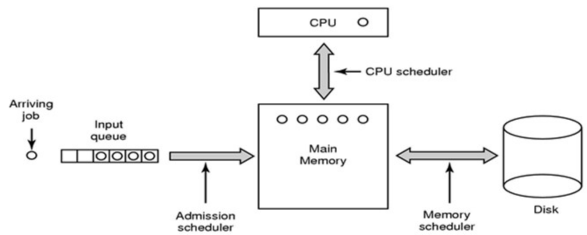
- Bursts of CPU usage alternate (đan xen/xen kẻ) with periods of I/O wait
  - a CPU-bound (giới hạn/rang buộc) process
  - an I/O bound process

52

52

Scheduling

Introduction to Scheduling (3)



Three level scheduling

53

53

Scheduling

Introduction to Scheduling (4)

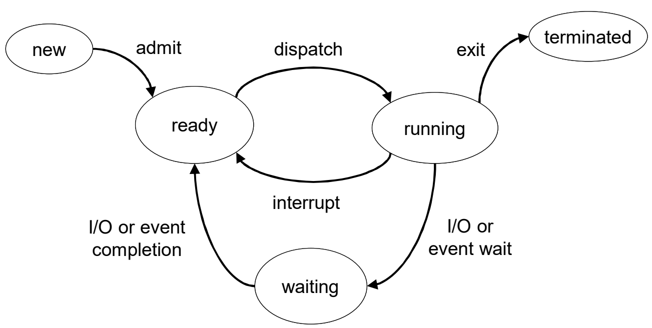
- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting or new process is created to ready
  4. Terminates
- *Nonpreemptive* scheduling algorithm picks (chọn) process and let it run until it blocks or until it voluntarily releases the CPU
- *Preemptive* scheduling algorithm picks process and let it run for a maximum of fix time

54

54

Scheduling

Introduction to Scheduling (5)



55

55

Scheduling

Introduction to Scheduling (6)

Scheduling Criteria

- CPU utilization – keep the CPU as busy as possible
- Throughput – # of processes that complete their execution per time unit
- Turnaround time – amount of time to execute a particular process
- Waiting time – amount of time a process has been waiting in the ready queue
- Response time – amount of time it takes from when a request was submitted until the first response is produced, **not** output (for time-sharing environment)

56

56

Scheduling
Introduction to Scheduling (7)

Optimization Criteria
- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

57

57

Scheduling
Introduction to Scheduling (8)

Scheduling Algorithm Goals

All systems
Fairness - giving each process a fair share of the CPU
Policy enforcement - seeing that stated policy is carried out
Balance - keeping all parts of the system busy

Batch systems
Throughput - maximize jobs per hour
Turnaround time - minimize time between submission and termination
CPU utilization - keep the CPU busy all the time

Interactive systems
Response time - respond to requests quickly
Proportionality - meet users' expectations

Real-time systems
Meeting deadlines - avoid losing data
Predictability - avoid quality degradation in multimedia systems

58

58

Processor Scheduling

- Aim (mục đích) is to assign (chỉ định) processes to be executed by the processor in a way that meets (thỏa mãn) **system (performance) objectives**, such as response time, throughput, and processor efficiency
- Broken down into three separate functions:

long term scheduling | medium term scheduling | short term scheduling

59

Table 9.1 Types of Scheduling

Long-term scheduling	The decision to add to the pool of processes to be executed
Medium-term scheduling	The decision to add to the number of processes that are partially or fully in main memory
Short-term scheduling	The decision as to which available process will be executed by the processor
I/O scheduling	The decision as to which process's pending (chưa giải quyết) I/O request shall be handled by an available I/O device

60

Scheduling and Process State Transitions

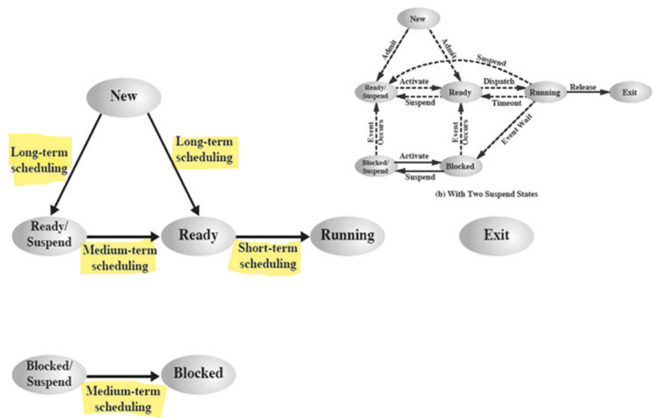


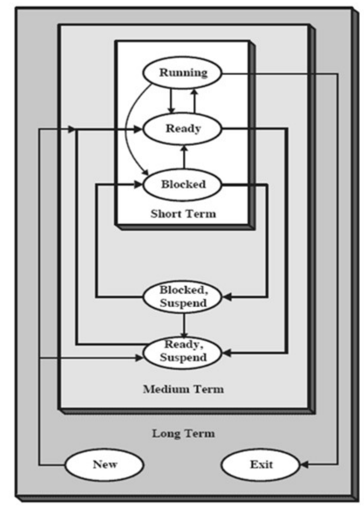
Figure 9.1 Scheduling and Process State Transitions

61

Figure 9.2

Nesting of

Scheduling Functions



62

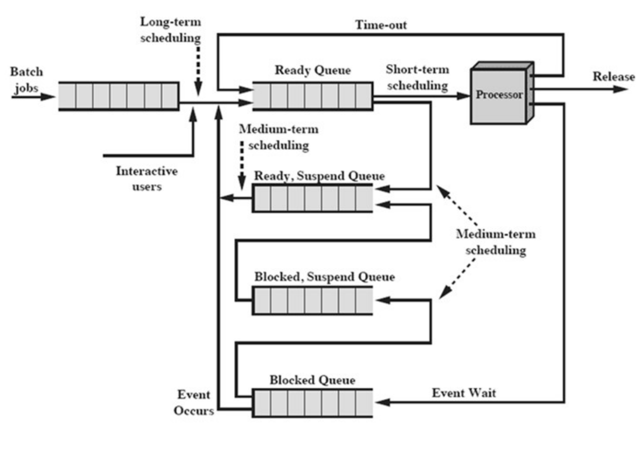
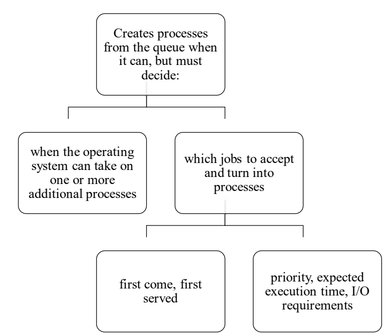


Figure 9.3 Queuing Diagram for Scheduling

63

Long-Term Scheduler

- Determines which programs are **admitted** to the system for processing
- Controls the **degree of multiprogramming**
    - the more processes that are created, the smaller the percentage of time that each process can be executed
    - may limit to provide satisfactory (làm thỏa mãn) service to the current set of processes



64

Medium-Term Scheduling

- Part of the swapping function
- Swapping-in decisions are based on the need to manage the degree of multiprogramming
  - considers (xem xét) the memory requirements of the swapped-out processes

65

Short-Term Scheduling

- Known as the **dispatcher**
- Executes most frequently (thường xuyên)
- Makes the fine-grained (làm mịn) decision of which process to execute next
- Invoked (cầu khẩn) when an event occurs that may lead to the blocking of the current process or that may provide an opportunity (cơ hội) to pre-empt (chiếm được) a currently running process in favor (sự cho phép) of another

Examples:

- **Clock interrupts**
- I/O interrupts
- **Operating system calls**
- **Signals (e.g., semaphores)**

66

Short Term Scheduling Criteria

- Main objective is to allocate processor time to optimize certain (chắc chắn) aspects (khía cạnh) of system behaviour (hành vi)
- A set of criteria is needed to evaluate (ước lượng) the scheduling policy

User-oriented criteria

- relate to the behavior of the system as perceived (hiểu) by the individual user or process (such as response time in an interactive system)
- important on virtually (hầu như) all systems

System-oriented criteria

- focus in on effective and efficient utilization of the processor (rate at which processes are completed)
- generally of minor (thứ yếu) importance on single-user systems

67

Scheduling Criteria

User Oriented, Performance Related

Turnaround time This is the interval of time between the submission of a process and its completion. Includes actual execution time plus time spent waiting for resources, including the processor. This is an appropriate measure for a batch job.

Response time For an interactive process, this is the time from the submission of a request until the response begins to be received. Often a process can begin producing some output to the user while continuing to process the request. Thus, this is a better measure than turnaround time from the user's point of view. The scheduling discipline should attempt to achieve low response time and to maximize the number of interactive users receiving acceptable response time.

Deadlines When process completion deadlines can be specified, the scheduling discipline should subordinate other goals to that of maximizing the percentage of deadlines met.

User Oriented, Other

Predictability A given job should run in about the same amount of time and at about the same cost regardless of the load on the system. A wide variation in response time or turnaround time is distracting to users. It may signal a wide swing in system workloads or the need for system tuning to cure instabilities.

System Oriented, Performance Related

Throughput The scheduling policy should attempt to maximize the number of processes completed per unit of time. This is a measure of how much work is being performed. This clearly depends on the average length of a process but is also influenced by the scheduling policy, which may affect utilization.

Processor utilization This is the percentage of time that the processor is busy. For an expensive shared system, this is a significant criterion. In single-user systems and in some other systems, such as real-time systems, this criterion is less important than some of the others.

System Oriented, Other

Fairness In the absence of guidance from the user or other system-supplied guidance, processes should be treated the same, and no process should suffer starvation.

Enforcing priorities When processes are assigned priorities, the scheduling policy should favor higher-priority processes.

68

Priority Queuing

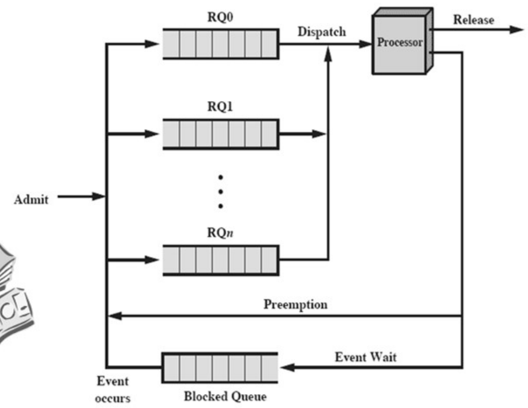


Figure 9.4 Priority Queuing

69

Selection Function

- Determines which process, among ready processes, is selected next for execution
- May be based on priority, resource requirements, or the execution characteristics of the process
- If based on execution characteristics then important quantities are:
  - w = time spent in system so far, **waiting**
  - e = time spent in **execution** so far
  - s = total service time required by the process, including e; generally, this quantity must be estimated (ước lượng) or supplied (cung cấp) by the user

70

Alternative Scheduling Policies

Table 9.3 Characteristics of Various Scheduling Policies

	FCFS	Round robin	SPN	SRT	HRRN	Feedback
Selection function	max[w]	constant	min[s]	min[s − e]	$\max\left(\frac{w+s}{s}\right)$	(see text)
Decision mode	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
Throughput	Not emphasized	May be low if quantum is too small	High	High	High	Not emphasized
Response time	May be high, especially if there is a large variance in process execution times	Provides good response time for short processes	Provides good response time for short processes	Provides good response time	Provides good response time	Not emphasized
Overhead	Minimum	Minimum	Can be high	Can be high	Can be high	Can be high
Effect on processes	Penalizes short processes; penalizes I/O bound processes	Fair treatment	Penalizes long processes	Penalizes long processes	Good balance	May favor I/O bound processes
Starvation	No	No	Possible	Possible	No	Possible

71

Decision Mode

- Specifies (chỉ định) the **instants (lập tức)** in time at which the selection function is exercised

- Two categories:
  - Nonpreemptive
  - Preemptive

72

Nonpreemptive vs Preemptive

Nonpreemptive

- once a process is in the running state, it will continue until it terminates, blocks itself for I/O, or **give up voluntarily (tự nguyện)**

Preemptive

- currently running process may be interrupted and moved to ready state by the OS
- preemption may occur when new process arrives (đi đến), on an interrupt, or periodically (định kỳ)

73

Process Scheduling Example

Table 9.4 Process Scheduling Example

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

74

Comparison of Scheduling Policies

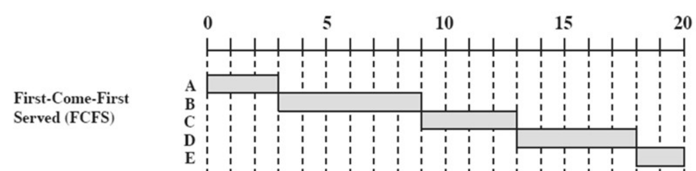
- RR: Round Robin
- SPN: Shortest Process Next
- SRT: Shortest Remaining Time
- HRRN: Highest Response Ratio Next
- FB: Feedback

Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (T_s)	3	6	4	5	2	Mean
FCFS						
Finish Time	3	9	13	18	20	
Turnaround Time (T_r)	3	7	9	12	12	8.60
T_r/T_s	1.00	1.17	2.25	2.40	6.00	2.56
RR $q = 1$						
Finish Time	4	18	17	20	15	
Turnaround Time (T_r)	4	16	13	14	7	10.80
T_r/T_s	1.33	2.67	3.25	2.80	3.50	2.71
RR $q = 4$						
Finish Time	3	17	11	20	19	
Turnaround Time (T_r)	3	15	7	14	11	10.00
T_r/T_s	1.00	2.5	1.75	2.80	5.50	2.71
SPN						
Finish Time	3	9	15	20	11	
Turnaround Time (T_r)	3	7	11	14	3	7.60
T_r/T_s	1.00	1.17	2.75	2.80	1.50	1.84
SRT						
Finish Time	3	15	8	20	10	
Turnaround Time (T_r)	3	13	4	14	2	7.20
T_r/T_s	1.00	2.17	1.00	2.80	1.00	1.59
HRRN						
Finish Time	3	9	13	20	15	
Turnaround Time (T_r)	3	7	9	14	7	8.00
T_r/T_s	1.00	1.17	2.25	2.80	3.5	2.14
FB $q = 1$						
Finish Time	4	20	16	19	11	
Turnaround Time (T_r)	4	18	12	13	3	10.00
T_r/T_s	1.33	3.00	3.00	2.60	1.5	2.29
FB $q = 2^i$						
Finish Time	4	17	18	20	14	

75

First-Come-First-Served (FCFS)

- Simplest scheduling policy
- Also known as first-in-first-out (FIFO) or a strict (chính xác) queuing scheme
- When the current process ceases (dừng) to execute, the longest process in the Ready queue is selected
- Performs much better for long processes than short ones
- Tends (tiến đến) to favor(thiên vị) processor-bound processes over (hơn) I/O-bound processes

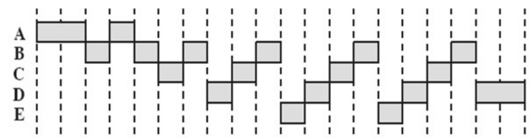


76

Round Robin

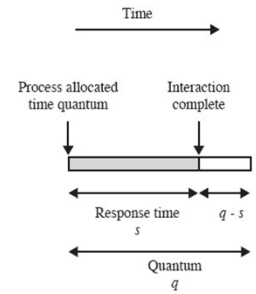
- Uses preemption based on a clock
- Also known as **time slicing** because each process is given a slice of time before being preempted
- Principal design issue is the length of the time quantum, or slice, to be used
- Particularly effective in a general-purpose time-sharing system or transaction (giao dịch) processing system
- One drawback (mặt hạn chế) is its relative treatment (sự giải quyết) of processor-bound (rang buộc) and I/O-bound processes

Round-Robin (RR), $q = 1$



77

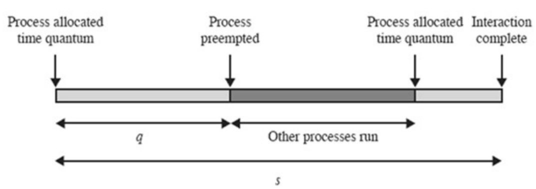
Effect of Size of Preemption Time Quantum



(a) Time quantum greater than typical interaction

78

Effect of Size of Preemption Time Quantum



(b) Time quantum less than typical interaction

79

Virtual Round Robin (VRR)

"Favor" (cho phép) I/O-bound processes to provide fairness and higher
I/O device utilization

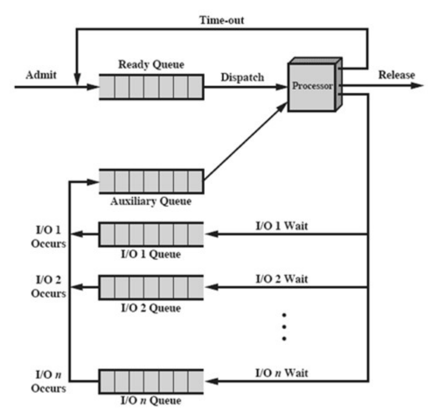


Figure 9.7 Queuing Diagram for Virtual Round-Robin Scheduler

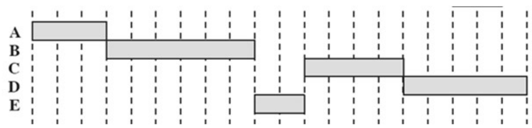
80

Shortest Process Next (SPN)

- **Nonpreemptive** policy in which the process with the shortest expected processing time is selected next
- A short process will jump to the head of the queue
- Possibility of starvation for longer processes

- One difficulty is the need to know, or at least estimate (đánh giá), the required processing time of each process
- If the programmer's estimate is substantially (về thực chất/căn bản) under the actual running time, the system may abort the job

Shortest Process Next (SPN)



81

Shortest Remaining (còn lại) Time (SRT)

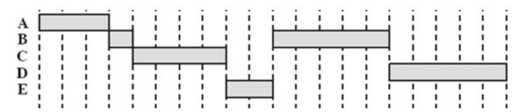
- **Preemptive** version of SPN
- Scheduler always chooses the process that has the shortest expected **remaining** processing time
- Risk (rủi ro) of starvation of longer processes

- Should give superior (nhiều hơn) turnaround (quay vòng) time performance to SPN because a short job is given **immediate (tức thì) preference (ưu tiên/thích)** to a running longer job

Table 9.4 Process Scheduling Example

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Shortest Remaining Time (SRT)



82

Highest Response Ratio Next (HRRN)

- Chooses next process with the greatest ratio (tỷ suất)
- Attractive (hấp dẫn) because it ~~accounts for~~ (giải thích cho) the **age** of the process

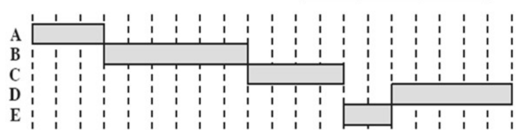
- While shorter jobs are favored (được hưởng ân huệ), **aging (sự già hóa) without service** increases the ratio so that a longer process will eventually (tính cho cùng) get past competing (đua tranh/cạnh tranh) shorter jobs

$$Ratio = \frac{time\ spent\ waiting + expected\ service\ time}{expected\ service\ time}$$

Table 9.4 Process Scheduling Example

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Highest Response Ratio Next (HRRN)



83

Feedback Scheduling

- Preemptive with time quantum
- Demoted (giáng cấp) to the next lower-priority queue
- With each queue (except(trừ) the lowest priority queue), FCFS
- Lowest-priority queue: RR

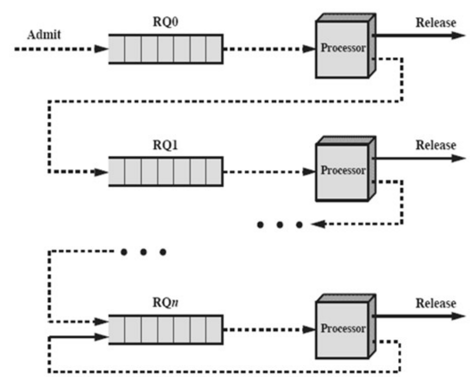
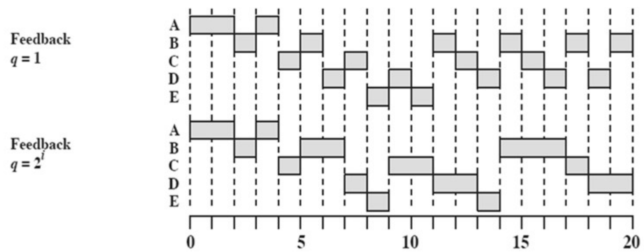


Figure 9.10 Feedback Scheduling

84

Feedback Performance



85

Scheduling
Scheduling in Interactive Systems (4)

Priority Scheduling: A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority
- Preemptive
- nonpreemptive

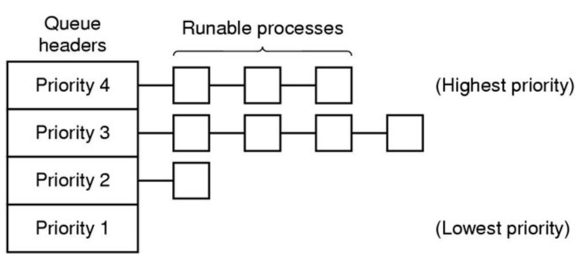
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem ≡ Starvation – low priority processes may never execute
- Solution ≡ Aging – as time progresses increase the priority of the process

86

86

Scheduling
Scheduling in Interactive Systems (5)

A scheduling algorithm with four priority classes



87

87

Priority scheduling

- Each priority has a priority number
- The highest priority can be scheduled first
- If all priorities equal, then it is FCFS

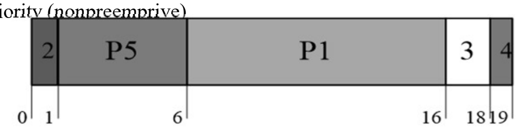
88

Example

- E.g.:

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

- Priority (nonpreemprive)



- Average waiting time
  = (6 + 0 + 16 + 18 + 1) /5 = 8.2

89

Scheduling
Scheduling in Real-Time Systems (1)

- *Hard real-time* systems – required to complete a critical task within a guaranteed (đảm bảo) amount of time
- *Soft real-time* computing – requires that critical processes receive priority over less fortunate (may mắn) ones

90

90

Scheduling
Scheduling in Real-Time Systems(2)

Schedulable real-time system

- Given
  - *m* periodic events
  - event *i* occurs within period P_i and requires C_i seconds
- Then the load can only be handled if

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

91

91

Scheduling
Policy versus Mechanism

- Separate (tách biệt) what is allowed to be done with how it is done
  - a process knows which of its children threads are important and need priority
- Scheduling algorithm parameterized
  - mechanism in the kernel
- Parameters filled in by user processes
  - policy set by user process

92

92

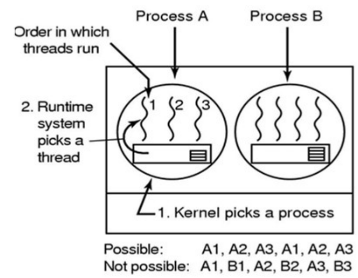
Scheduling
Thread Scheduling (1)

- Local Scheduling – How the threads library decides which thread to put onto an available
- Global Scheduling – How the kernel decides which kernel thread to run next

93

93

Scheduling
Thread Scheduling (2)

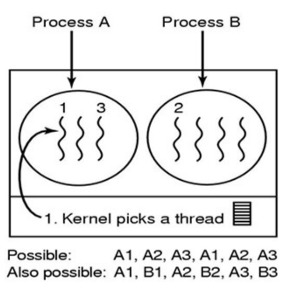


Possible scheduling of user-level threads
- 50-msec process quantum
- threads run 5 msec/CPU burst

94

94

Scheduling
Thread Scheduling (3)



Possible scheduling of kernel-level threads
- 50-msec process quantum
- threads run 5 msec/CPU burst

95

95

2.4 Interprocess Communication

96

96

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

97

97

Problem of shared data

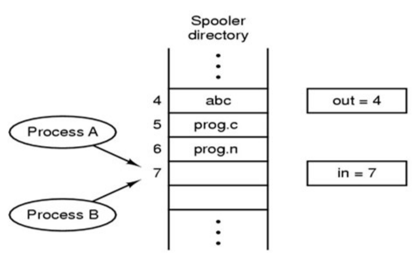
- Concurrent access to shared data may result in data inconsistency (không thống nhất)
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes
- Need of mechanism for processes to communicate and to synchronize their actions

98

98

Race Conditions

- Two processes want to access shared memory at same time and the final result depends who runs precisely, are called **race condition**
- **Mutual exclusion** is the way to prohibit (cấm) more than one process from accessing to shared data at the same time



99

99

Critical Regions (1)

The Part of the program where the shared memory is accessed is called **Critical Regions (Critical Section)**

Four conditions to provide mutual exclusion

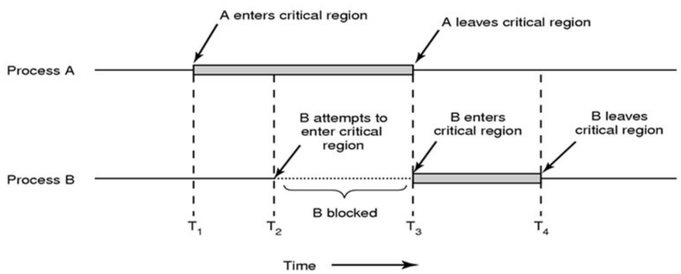
1. No two processes simultaneously in critical region
2. No assumptions made about speeds or numbers of CPUs
3. No process running outside its critical region may block another process
4. No process must wait forever to enter its critical region

100

100

Critical Regions (2)

Mutual exclusion using critical regions (Example)



101

101

Solution: Mutual exclusion with Busy waiting

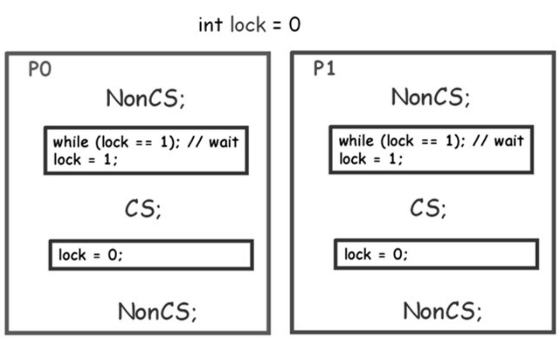
- Software proposal (đề xuất)
  - Lock Variables
  - Strict Alternation
  - Peterson's Solution
- Hardware proposal
  - Disabling Interrupts
  - The TSL Instruction

102

102

Mutual exclusion with Busy waiting

Software Proposal 1: Lock Variables

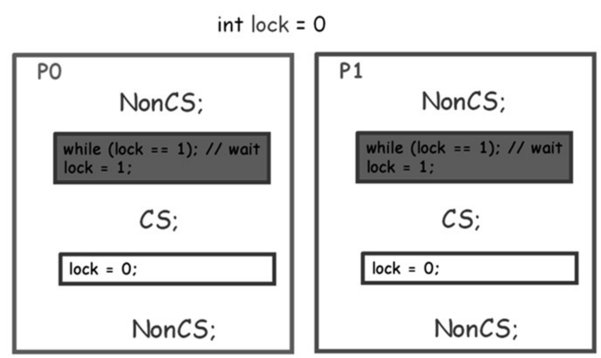


103

103

Mutual exclusion with Busy waiting

Software Proposal 1: Event



104

104

Mutual exclusion with Busy waiting
Software Proposal 2: Strict Alternation

int turn = 1

```
P0
        NonCS;
  while (turn !=0); // wait
        CS;
  turn = 1;
        NonCS;
```

```
P1
        NonCS;
  while (turn != 1); // wait
        CS;
  turn = 0;
        NonCS;
```

105

105

Mutual exclusion with Busy waiting
Software Proposal 2: Strict Alternation

- Only 2 processes
- Responsibility (trách nhiệm) Mutual Exclusion
  - One variable "*turn*“, one process “*turn*” (phiên/lượt) come in CS at the moment.

106

106

Mutual exclusion with Busy waiting
Software Proposal 3: Peterson's Solution

- int turn;
- int interest[2] = FALSE;

```
Pi              NonCS;
  j = 1 - i;
  interest[i] = TRUE;
  turn = j;
  while (turn==j && interest[j]==TRUE);
                CS;
  interest[i] = FALSE;
                NonCS;
```

107

107

Mutual exclusion with Busy waiting
Software Proposal 3: Peterson's Solution

```
Pj              NonCS;
  i = 1 - j;
  interest[j] = TRUE;
  turn = i;
  while (turn==i && interest[i]==TRUE);
                CS;
  interest[j] = FALSE;
                NonCS;
```

108

108

Mutual exclusion with Busy waiting
Comment for Software Proposal 3: Peterson's Solution

- Satisfy (thỏa mãn) 3 conditions:
  - Mutual Exclusion
    - Pi can enter CS when *interest[j] == F, or turn == i*
    - If both want to come back, because *turn* can only receive value 0 or 1, so one process enter CS
  - Progress (tiến độ/tiến triển)
    - Using 2 variables distinct *interest[i]* ==> opposing cannot lock
  - Bounded Wait: both *interest[i]* and *turn* change value
- Not extend into N processes

109

109

Mutual exclusion with Busy waiting
Comment for Busy-Waiting solutions

- Don't need system's support
- Hard to extend
- Solution 1 is better when *atomicity* is supported

110

110

Busy waiting – Hardware Proposal

- Software proposal
  - Lock Variables
  - Strict Alternation
  - Peterson's Solution
- Hardware proposal
  - Disabling Interrupts
  - The TLS Instruction

111

111

Busy waiting – Hardware Proposal 1: Disabling Interrupt

```
NonCS;
Disable Interrupt;
CS;
Enable Interrupt;
NonCS;
```

- Disable Interrupt: prohibit all interrupts, including spin interrupt
- Enable Interrupt: permit interrupt

112

112

Hardware proposal 1: Disable Interrupt

- Not be careful
  - If process is locked in CS?
    - System Halt
  - Permit process use command privileges
    - Danger!
- System with N CPUs?
  - Don't ensure Mutual Exclusion

113

Hardware proposal 1: TSL Instruction

- CPU support primitive Test and Set Lock
  - Return a variable's current value, set variable to true value
  - Cannot divide up to perform (Atomic)

```
TSL (boolean &target)
{
        TSL = target;
        target = TRUE;
}
```


114

Applied TSL

```
int lock = 0

Pi
    NonCS;
    while (TSL(lock)); // wait
    CS;
    lock = 0;
    NonCS;
```


115

Comment for hardware solutions in Busy-Waiting

- Necessary hardware mechanism's support
  - Not easy with n-CPUs system
- Easily extend to N processes

116

Comment for hardware solutions in Busy-Waiting

- Using CPU not effectively
  - Constantly (luôn luôn) test condition when wait for coming in CS
- Overcome (khắc phục)
  - Lock processes that not enough condition to come in CS, concede (nhường) CPU to other process
    - Using Scheduler
    - Wait and See...

117

117

Synchronous solution

- Sleep & Wakeup
  - Semaphore
  - Message passing

118

118

"Sleep & Wake up" solution

```
if not Sleep();
CS;
Wakeup(somebody);
```

- Give up (từ bỏ) CPU when not come in CS
- When CS is empty, will be waken up to come in CS
- Need support of OS
  - Because of changing status of process

119

119

"Sleep & Wake up" solution: Idea

- OS support 2 primitive (nguyên hàm):
  - Sleep(): System call receives blocked status
  - WakeUp(P): P process receive ready status
- Application
  - After checking condition, coming in CS or calling Sleep() depend on result of checking
  - Process that using CS before, will wake up processes blocked before

120

120

Apply Sleep() and Wakeup()

```
▪ int busy;
▪ int blocked;
if (busy) {
        blocked = blocked + 1;
        Sleep();
    }
else busy = 1;

CS;

busy = 0;
    if(blocked) {     WakeUp(P);
                      blocked = blocked - 1;
              }
```

121

121

Problem with Sleep & WakeUp

- Reason:
  - Checking condition and giving up CPU can be broken
  - Lock variable is not protected

122

122

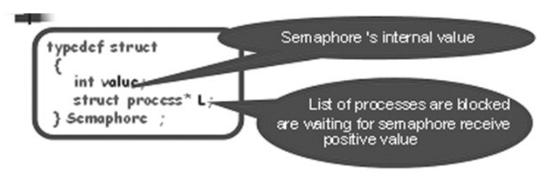
Semaphore

- Suggested by Dijkstra, 1965
- Properties: Semaphore s;
  - Unique value
  - Manipulate with 2 primitives:
    - Down(s)
    - Up(s)
  - Down and Up primitives excuted cannot divide up

123

123

Install Semaphore (Sleep & Wakeup)



Semaphore: similar to resource
- Processes "request" semaphore: call Down(s)
  - If Down(s) is not finished: resource is not allocated
    - Blocked, insert to s.L

Need OS's support
- Sleep() & Wakeup()

124

124

Install Semaphore (Sleep & Wakeup)

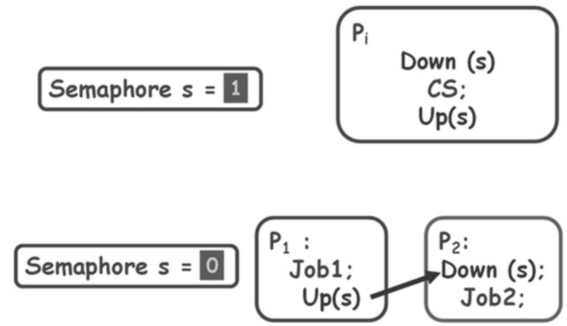
```
Down (S)
{
  S.value --;
  if S.value < 0
  {
    Add(P,S.L);
    Sleep();
  }
}
```

```
Up(S)
{
  S.value ++;
  if S.value ≤ 0
  {
    Remove(P,S.L);
    Wakeup(P);
  }
}
```

125

125

Using Semaphore



126

126

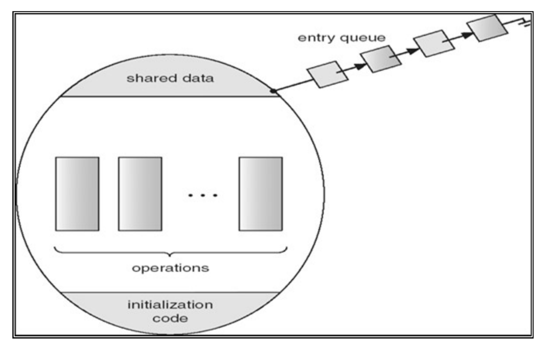
Monitor

- Hoare (1974) & Brinch (1975)
- Synchronous mechanism is provided by programming language
  - Support with functions, such as Semaphore
  - Easier for using and detecting than Semaphore
    - Ensure Mutual Exclusion automatically
    - Using condition variable to perform Synchronization

127

127

Monitor: structure



128

128

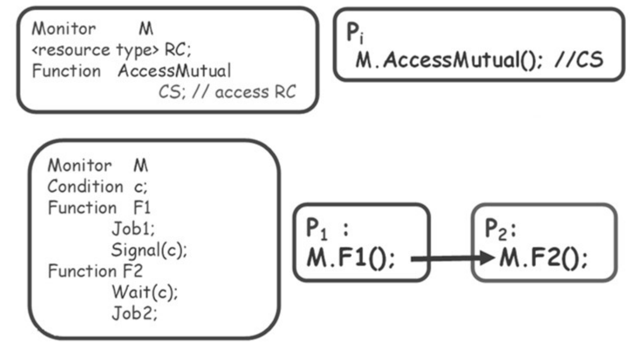
Monitor: structure

monitor ***monitor-name***
{
shared variable declarations
procedure body *P1* (...) {
. . .
}
procedure body *P2* (...) {
. . .
}
procedure body *Pn* (...) {
. . .
}
{
initialization code
}
}

129

129

Using Monitor



130

130

Message Passing

- Processes must name each other explicitly:
  - **send** (*P, message*) – send a message to process P
  - **receive**(*Q, message*) – receive a message from process Q
- Properties of communication link
  - Links are established automatically
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
  - The link may be unidirectional, but is usually bi-directional

131

131

Classical Problems of Synchronization

- Bounded-Buffer Problem (Producer-Consumer Problem)
- Readers and Writers Problem
- Dining-Philosophers Problem

132

132

Summary

- The most fundamental concept in a modern OS is the process
- The principal function of the OS is to create, manage, and terminate processes
- Process control block contains all of the information that is required for the OS to manage the process, including its current state, resources allocated to it, priority, and other relevant (thích hợp) data
- The most important states are Ready, Running and Blocked
- The running process is the one that is currently being executed by the processor
- A blocked process is waiting for the completion of some event
- A running process is interrupted either by an interrupt or by executing a supervisor call to the OS

133

Summary

- User-level threads
  - created and managed by a threads library that runs in the user space of a process
  - a mode switch is not required to switch from one thread to another
  - only a single user-level thread within a process can execute at a time
  - if one thread blocks, the entire (toàn bộ) process is blocked
- Kernel-level threads
  - threads within a process that are maintained by the kernel
  - a mode switch is required to switch from one thread to another
  - multiple threads within the same process can execute in parallel on a multiprocessor
  - blocking of a thread does not block the entire process
- Process/related to resource ownership
- Thread/related to program execution

134

Summary

- The operating system must make three types of scheduling decisions with respect (mối quan hệ) to the execution of processes:
  - Long-term – determines when new processes are admitted to the system
  - Medium-term – part of the swapping function and determines when a program is brought into main memory so that it may be executed
  - Short-term – determines which ready process will be executed next by the processor
- From a user's point of view, response time is generally the most important characteristic of a system; from a system point of view, throughput or processor utilization (sự sử dụng) is important
- Algorithms:
  - » FCFS, Round Robin, SPN, SRT, HRRN, Feedback

135