Chapter 4: Thread & Concurency

4.2 Using Amdahl's Law, calculate the speedup gain of an application that has a 60 percent parallel component for (a) two processing cores and (b) four processing cores.

Answer:

a. With two processing cores we get a speedup of 1.42 times.

b. With four processing cores, we get a speedup of 1.82 times.

4.14 Using Amdahl's Law, calculate the speedup gain for the following applications:

- 40 percent parallel with (a) eight processing cores and (b) sixteen processing cores
- 67 percent parallel with (a) two processing cores and (b) four processing cores
- 90 percent parallel with (a) four processing cores and (b) eight processing cores

*The question calculates the **speedup gain** for **varying degrees** of parallelism (40 percent, 67 percent, and 90 percent) and varying numbers of processing cores (two, four, eight, sixteen), using Amdahl's Law. The results show that as the number of processing cores and degree of parallelism increase, speedup gain increases.*

Explanation:

Amdahl's Law is used to find the maximum improvement in performance of a system task when only part of the system can be improved. It is defined as Speedup = 1 / ((1 -P) + P/N), where P is the portion of the task that can be made parallel and N is the number of processors.

a) 40 percent (0.4) parallel:

- ***With eight processing cores:*** *Speedup = 1 / ((1 - 0.4) + (0.4 / 8)) = 1.538*
- ***With sixteen processing cores:*** *Speedup = 1 / ((1 - 0.4) + (0.4 / 16)) = 1.905*

b) 67 percent (0.67) parallel:

- ***With two processing cores:*** *Speedup = 1 / ((1 - 0.67) + (0.67 / 2)) = 1.764*
- ***With four processing cores:*** *Speedup = 1 / ((1 - 0.67) + (0.67 / 4)) = 2.258*

c) 90 percent (0.9) parallel:

- ***With four processing cores:*** *Speedup = 1 / ((1 - 0.9) + (0.9 / 4)) = 2.957*
- ***With eight processing cores:*** *Speedup = 1 / ((1 - 0.9) + (0.9 / 8)) = 3.902*

Chapter 5: CPU Scheduling

The Gantt Chart for the schedule, turnaround time, waiting time): FCFS, SJF, SRTF, RR

5.3 Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed. In answering the questions, use nonpreemptive scheduling, and base all decisions on the information you have at the time the decision must be made.

Process	Arrival Time	Burst Time
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

a. What is the average turnaround time for these processes with the FCFS scheduling algorithm?

b. What is the average turnaround time for these processes with the SJF scheduling algorithm?

c. The SJF algorithm is supposed to improve performance, but notice that we chose to run process P^1 at time 0 because we did not know that two shorter processes would arrive soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes P_1 and P_2 are waiting during this idle time, so their waiting time may increase. This algorithm could be known as ***future-knowledge scheduling***.

Answer:

a/ Average turnaround for these processes: (8 + (12 - 0.4) + (13 - 1)) / 3 = 10.53

b/ Average turnaround for these processes: (8 + (9 - 1) + (13 – 0.4)) / 3 = 9.53

c/ CPU is left idle: Average turnaround for these processes: ((2 - 1) + (6 – 0.4) + (14 - 0)) / 3 = 6.87

Remember that turnaround time is finishing time minus arrival time, so you have to subtract the arrival times to compute the turnaround times. FCFS is 11 if you forget to subtract arrival time.

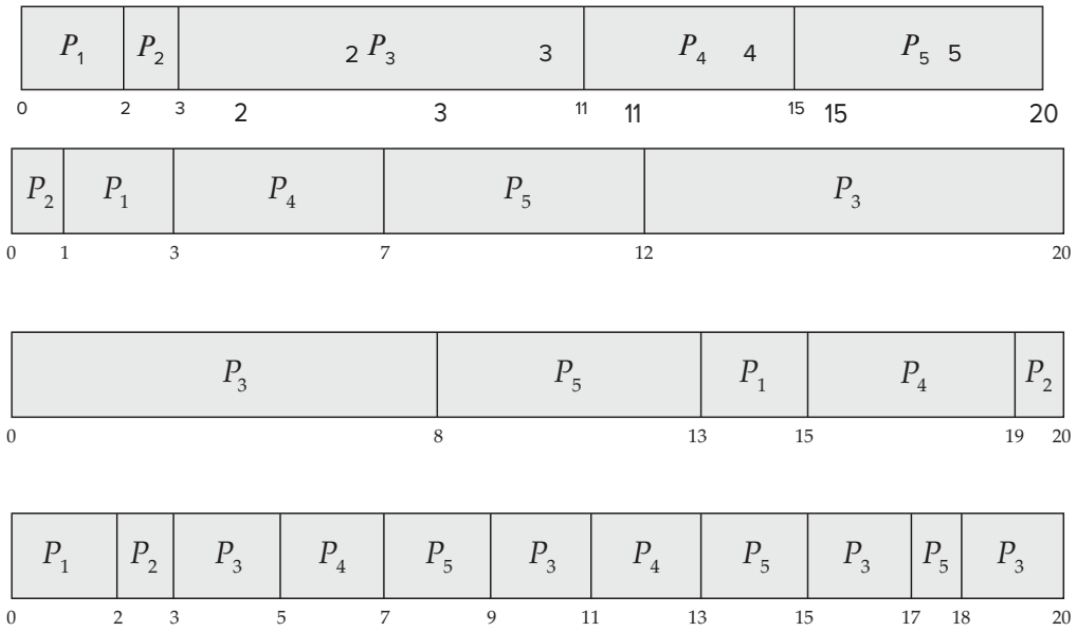
5.4 Consider the following set of processes, with the length of the CPU burst time given in milliseconds:

Process	Burst Time	Priority
P 1	2	2
P 2	1	1
P 3	8	4
P 4	4	2
P 5	5	3

The processes are assumed to have arrived in the order *P*1, *P*2, *P*3, *P*4, *P*5, all at time 0.

a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, nonpreemptive priority (a larger priority number implies a higher priority), and RR (quantum = 2).

a. The four Gantt charts:



b. What is the turnaround time of each process for each of the scheduling algorithms in part a?

	FCFS	SJF	Priority	RR
P_1	2	3	15	2
P_2	3	1	20	3
P_3	11	20	8	20
P_4	15	7	19	13
P_5	20	12	13	18

c. What is the waiting time of each process for each of these scheduling algorithms?

	FCFS	SJF	Priority	RR
P_1	0	1	13	0
P_2	2	0	19	2
P_3	3	12	0	12
P_4	11	3	15	9
P_5	15	7	8	13

d. Which of the algorithms results in the minimum average waiting time (over all processes)?

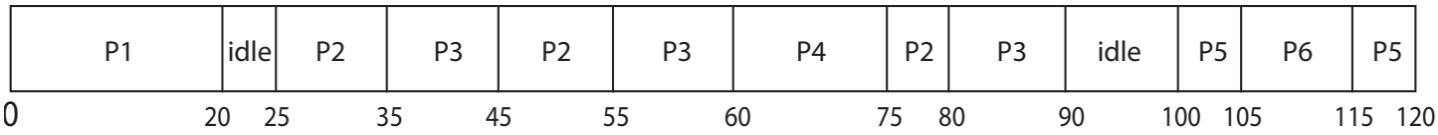
→ SJF has the shortest wait time.

5.5 The following processes are being scheduled using a preemptive, roundrobin scheduling algorithm.

Process	**Priority**	**Burst**	**Arrival**
P_1	40	20	0
P_2	30	25	25
P_3	30	25	30
P_4	35	15	60
P_5	5	10	100
P_6	10	10	105

Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes listed below, the system also has an **idle task** (which consumes no CPU resources and is identified as *Pidle*). This task has priority 0 and is scheduled whenever the system has no other available processes to run. The length of a time quantum is 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

a. Show the scheduling order of the processes using a Gantt chart.
The Gantt chart:



b. What is the turnaround time for each process?
P1: 20-0 - 20, P2: 80-25 = 55, P3: 90 - 30 = 60, P4: 75-60 = 15, P5: 120-100 = 20, P6: 115-105 = 10

c. What is the waiting time for each process?
P1: 0, P2: 40, P3: 35, P4: 0, P5: 10, P6: 0

d. What is the CPU utilization rate? 105/120 = 87.5 percent

- Assume that you have following processes, their arrival times and burst times. For FCFS scheduling algorithms, determine the waiting times for each process and the average waiting times.

Job	Arrival Time	CPU burst
J1	0	5
J2	3	6
J3	4	8
J4	10	4
J5	12	1
J6	15	3

Job	Thời gian chờ	Thời gian hoàn thành
J1	0	5 – 0 = 5
J2	5 – 3 = 2	11 – 3 = 8
J3	11 – 4 = 7	19 – 4 = 15
J4	19 – 10 = 9	23 – 10 = 13
J5	23 – 12 = 11	24 – 12 = 12
J6	24 – 15 = 9	27 – 15 = 12
Trung Bình	6.33	10.83

- Assume that you have following processes, their arrival times and burst times. For SJF non-preemptive scheduling algorithms, determine the waiting times for each process and the average waiting times. ***(dùng số liệu bài trên)***

Job	Thời gian chờ	Thời gian hoàn thành
J1	0	5 – 0 = 5
J2	5 – 3 = 2	11 – 3 = 8
J3	19 – 4 = 15	27 – 4 = 23
J4	11 – 10 = 1	15 – 10 = 5
J5	15 – 12 = 3	16 – 12 = 4
J6	16 – 15 = 1	19 – 15 = 4
Trung Bình	3.67	8.167

5.17 Consider the following set of processes, with the length of the CPU burst given in milliseconds:

Process	Burst Time	Priority
*P*1	5	4
*P*2	3	1
*P*3	1	2
*P*4	7	2
*P*5	4	3

The processes are assumed to have arrived in the order *P*1, *P*2, *P*3, *P*4, *P*5, all at time 0.

a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, nonpreemptive priority (a larger priority number implies a higher priority), and RR (quantum = 2).

b. What is the turnaround time of each process for each of the scheduling algorithms in part a?

c. What is the waiting time of each process for each of these scheduling algorithms?

d. Which of the algorithms results in the minimum average waiting time (over all processes)?

5.18 The following processes are being scheduled using a preemptive, priority-based, round-robin scheduling algorithm.

Process	Priority	Burst	Arrival
*P*1	8	15	0
*P*2	3	20	0
*P*3	4	20	20
*P*4	4	20	25
*P*5	5	5	45
*P*6	5	15	55

Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. The scheduler will execute the highestpriority process. For processes with the same priority, a round-robin scheduler will be used with a time quantum of 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

a. Show the scheduling order of the processes using a Gantt chart.

b. What is the turnaround time for each process?

c. What is the waiting time for each process?

5.10 The traditional UNIX scheduler enforces an inverse relationship between priority numbers and priorities: the higher the number, the lower the priority. The scheduler recalculates process priorities once per second using the following function:

Priority = (recent CPU usage / 2) + base

where base = 60 and *recent CPU usage* refers to a value indicating how often a process has used the CPU since priorities were last recalculated.

Assume that recent CPU usage for process *P*1 is 40, for process *P*2 is 18, and for process *P*3 is 10. What will be the new priorities for these three processes when priorities are recalculated? Based on this information, does the traditional UNIX scheduler raise or lower the relative priority of a CPU-bound process?

Answer:

The priorities assigned to the processes will be 80, 69, and 65, respectively. The scheduler lowers the relative priority of CPU-bound processes.

5.22 Consider a system running ten I/O-bound tasks and one CPU-bound task. Assume that the I/O-bound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete. Also assume that the context-switching overhead is 0.1 millisecond and that all processes are long-running tasks. Describe the CPU utilization for a round-robin scheduler when:

a. The time quantum is 1 millisecond

b. The time quantum is 10 milliseconds

Answer:

a. The time quantum is 1millisecond: Irrespective of which process is scheduled, the scheduler incurs a 0.1 millisecond context-switching cost for every context-switch. This results in a CPU utilization of 1/1.1 * 100 = 91%.

b. The time quantum is 10 milliseconds: The I/O-bound tasks incur a context switch after using up only 1 millisecond of the time quantum. The time required to cycle through all the processes is therefore 10*1.1 + 10.1 (as each I/O-bound task executes for 1 millisecond and then incur the context switch task, whereas the CPU-bound task executes for 10 milliseconds before incurring a context switch). The CPU utilization is therefore 20/21.1 * 100 = 94%.

5.35 Consider two processes, *P*1 and *P*2, where *p*1 = 50, *t*1 = 25, *p*2 = 75, and *t*2 = 30.

a. Can these two processes be scheduled using rate-monotonic scheduling? Illustrate your answer using a Gantt chart such as the ones in Figure 5.21–Figure 5.24.

b. Illustrate the scheduling of these two processes using earliestdeadline-first (EDF) scheduling.

Answer:

Consider when P_1 is assigned a higher priority than P_2 with the rate monotonic scheduler. P_1 is scheduled at $t = 0$, P_2 is scheduled at $t = 25$, P_1 is scheduled at $t = 50$, and P_2 is scheduled at $t = 75$. P_2 is not scheduled early enough to meet its deadline. When P_1 is assigned a lower priority than P_2, then P_1 does not meet its deadline since it will not be scheduled in time.

Chapter 8: Deadlocks: Banker's Algorithm

8.2 Suppose that a system is in an unsafe state. Show that it is possible for the threads to complete their execution without entering a deadlocked state.

Answer:

An unsafe state may not necessarily lead to deadlock, it just means that we cannot guarantee that deadlock will not occur. Thus, it is possible that a system in an unsafe state may still allow all processes to complete without deadlock occurring. Consider the situation where a system has twelve resources allocated among processes *P*0, *P*1, and *P*2. The resources are allocated according to the following policy:

	Max	Current	Need
*P*0	10	5	5
*P*1	4	2	2
*P*2	9	3	6

Currently, there are two resources available. This system is in an unsafe state. Process *P*1 could complete, thereby freeing a total of four resources, but we cannot guarantee that processes *P*0 and *P*2 can complete. However, it is possible that a process may release resources before requesting any further resources. For example, process *P*2 could release a resource, thereby increasing the total number of resources to ve. This allows pro cess *P*0 to complete, which would free a total of nine resources, thereby allowing process *P*2 to complete as well.

8.3 Consider the following snapshot of a system:

	Allocation	***Max***	***Available***
	A B C D	*A B C D*	*A B C D*
*T*0	0 0 1 2	0 0 1 2	1 5 2 0
*T*1	1 0 0 0	1 7 5 0	
*T*2	1 3 5 4	2 3 5 6	
*T*3	0 6 3 2	0 6 5 2	
*T*4	0 0 1 4	0 6 5 6	

Answer the following questions using the banker's algorithm:

a. What is the content of the matrix ***Need***?

b. Is the system in a safe state?

c. If a request from thread *T*1 arrives for (0,4,2,0), can the request be granted immediately?

Answer:

a. The values of ***Need*** for processes *P*0 through *P*4, respectively, are (0, 0, 0, 0), (0, 7, 5, 0), (1, 0, 0, 2), (0, 0, 2, 0), and (0, 6, 4, 2).

b. The system is in a safe state. With ***Available*** equal to (1, 5, 2, 0), either process *P*0 or *P*3 could run. Once process *P*3 runs, it releases its resources, which allows all other existing processes to run.

c. The request can be granted immediately. The value of ***Available*** is then (1, 1, 0, 0). One ordering of processes that can nish is *P*0, *P*2, *P*3, *P*1, and *P*4.

8.9 Consider the following snapshot of a system:

	Allocation	***Max***
	A B C D	*A B C D*
*T*0	3 0 1 4	5 1 1 7
*T*1	2 2 1 0	3 2 1 1
*T*2	3 1 2 1	3 3 2 1
*T*3	0 5 1 0	4 6 1 2
*T*4	4 2 1 2	6 3 2 5

Using the banker's algorithm, determine whether or not each of the following states is unsafe. If the state is safe, illustrate the order in which the threads may complete. Otherwise, illustrate why the state is unsafe.
a. ***Available*** = (0, 3, 0, 1)
b. ***Available*** = (1, 0, 0, 2)

Answer:

a. Not safe. Processes *P*2, *P*1, and *P*3 are able to nish, but no remaining processes can finish.
b. Safe. Processes *P*1, *P*2, and *P*3 are able to nish. Following this, processes *P*0 and *P*4 are also able to finish.

- Consider the following snapshot of a system with 4 types of resources: 9 A, 9 B, 8 C, 6 D. What is the content of the matrix Need? Is the system in a safe state?

Process	**Allocation**				**Max**			
	A	**B**	**C**	**D**	**A**	**B**	**C**	**D**
P1	1	2	0	1	**4**	**4**	**1**	**1**
P2	1	0	1	1	**7**	**5**	**6**	**2**
P3	1	1	1	1	**3**	**2**	**5**	**1**
P4	2	2	0	1	**4**	**2**	**3**	**1**
P5	**0**	**2**	**0**	**1**	**7**	**4**	**5**	**2**

- Available

A	**B**	**C**	**D**
4	2	6	1

- Need

	A	**B**	**C**	**D**
P1	3	2	1	0
P2	6	5	5	1
P3	2	1	4	0
P4	2	0	3	0
P5	7	2	5	1

- **Safe state: P1 → P3 → P4 → P5 → P2**

8.21 Consider the following snapshot of a system:

	Allocation	***Max***
	A B C D	*A B C D*
*T*0	2 1 0 6	6 3 2 7
*T*1	3 3 1 3	5 4 1 5
*T*2	2 3 1 2	6 6 1 4
*T*3	1 2 3 4	4 3 4 5
*T*4	3 0 3 0	7 2 6 1

What are the contents of the ***Need*** matrix?

8.27 Consider the following snapshot of a system:

	Allocation	***Max***
	A B C D	*A B C D*
*T*0	1 2 0 2	4 3 1 6
*T*1	0 1 1 2	2 4 2 4
*T*2	1 2 4 0	3 6 5 1
*T*3	1 2 0 1	2 6 2 3
*T*4	1 0 0 1	3 1 1 2

Using the banker's algorithm, determine whether or not each of the following states is unsafe. If the state is safe, illustrate the order in which the threads may complete. Otherwise, illustrate why the state is unsafe.

a. ***Available*** = (2, 2, 2, 3)
b. ***Available*** = (4, 4, 1, 1)
c. ***Available*** = (3, 0, 1, 4)
d. ***Available*** = (1, 5, 2, 2)

a. Yes. Possible safe sequences <T4, T0, …> and <T4, T1, …> the dots indicating where any ordering of the remaining threads is correct.

Full working:

Available = (2, 2, 2, 3)

- The only thread that can execute to the completion is T4. Assuming T4 has completed, the new resources available are: **Available = (2, 2, 2, 3) + (2, 1, 1, 1,) = (4, 3, 3, 4)**

- The next threads that can be completed are: T0 and T1. So, <T4, T0, ..> and <T4, T1, …> are two acceptable sequences. Assuming T0 or T1 has completed then, the resources available are:
Available = (4, 3, 3, 4) + (1, 2, 0, 2) = (5, 5, 3, 6) {for T0} <T4, T0, …> any other thread can complete
Available = (4, 3, 3, 4) + (0, 1, 1, 2) = (4, 4, 4, 6) {for T1} <T4, T1, …> any other thread can complete
All possible safe sequences: <T4, T0, …> and <T4, T1, …>

b. Yes. One possible ordering are:
{<T2 T4 T1 T0 T3> , <T2 T4 T1 T3 T0>, <T2, T4, T3, T1, T0>, {<T4 T1 T0 T2 T3> <T4 T1 T0 T3 T2> <T4 T1 T2 T0 T3> <T4 T1 T2 T3 T0> <T4 T1 T3 T0 T2> < T4 T1 T3 T2 T0>, <T4, T2, T1, T0, T3>, <T4, T2 T1 T3 T0>, <T4, T2, T3, T1, T0>

Full working:

Available = (4, 4, 1, 1)

- The threads that can execute to the completion are: T2 and T4. Assuming T2 or T4 complete then, the resources available become:
T2 completes: Available = (4, 4, 1, 1) + (1, 2, 4, 0) = (5, 6, 5, 1) only T4 can complete <T2, T4, …>
T4 completes: Available = (4, 4, 1, 1) + (1, 0, 0, 1) = (5, 4, 1, 2) only T1 and T2 can complete

1) After the execution of T2

- Assuming T4 is executed after T2 then, the resources available become:
Available = (5, 6, 5, 1) + (1, 0, 0, 1) = (6, 6, 5, 2) T1 and T3 can complete but, not T0. So,

After the execution of T2 followed by T4

-Assuming T1 is executed then, Available = (6, 6, 5, 2) + (0, 1, 1, 2) = (6, 7, 6, 4) To and T3 can complete
The safe sequences: <T2, T4, T1, ..> {<T2 T4 T1 T0 T3> and <T2 T4 T1 T3 T0>}
-Assuming T3 is executed after T4 then,
Available = (6, 6, 5, 2) + (1, 2, 0, 1) = (7, 8, 5, 3) , T1 can complete but not T0
-Assuming T1 is completed after T2, T4 and T3 then, Available = (7, 8, 5, 3) +(0, 1, 1, 2) = (7, 9, 6, 5) T0 can complete and so, <T2, T4, T3, T1, T0> is safe.

2) After the execution of T4

- Assuming T1 is completed after T4 then,
Available = (5, 4, 1, 2) + (0, 1, 1, 2) = (5, 5, 2, 4) T0, T2 and T3 can complète.
<T4, T1, …> {<T4 T1 T0 T2 T3> <T4 T1 T0 T3 T2> <T4 T1 T2 T0 T3> <T4 T1 T2 T3 T0> <T4 T1 T3 T0 T2> < T4 T1 T3 T2 T0>}

- Assuming T2 is completed after T4 then,
  Available = (5, 4, 1, 2) + (1, 2, 4, 0) = (6, 6, 5, 2) T1 and T3 can complete but, not T0
- Assuming T1 is completed after T2
  Available = (6, 6, 5, 2) + (0, 1, 1, 2) = (6, 7, 6, 4), T0 and T3 can complete so,
  <T4, T2, T1, ..> is safe {<T4, T2, T1, T0, T3>, <T4, T2 T1 T3 T0>}

- Assuming T3 is completed after T2 then

Available = (6, 6, 5, 2) + (1, 2, 0, 1) = (7, 8, 5, 3) T1 can complete but not T0
- Assuming T1 is completed then, Available = (7, 8, 5, 3) + (0, 1, 1, 2) = (7, 9, 6, 5) T0 can complete.

<T4, T2, T3, T1, T0>

c. No.

Full working:

Available = (3, 0, 1, 4)

- T0 cannot complete as it needs 1 copy of the second resource which is not available
- T1 cannot complete as it needs 3 copies of the second resource which is not available
- T2 cannot complete as it needs 4 copies of B which is not available
- T3 cannot complete as it needs 4 copies of B
- T4 cannot complete as it needs 1 copy of B which is not available

d. Yes. Possible orderings: <T3, T2, T1, T4, T0>, <T3, T2, T4, T1, T0>, <T3, T4, T0, T1, T2>, <T3, T4, T0, T2, T1>, <T3, T4, T1, T0, T2>, <T3, T4, T1, T2, T0>, <T3, T4, T2, T0, T1>, <T3, T4, T2, T1, T0>.

Full working:

Available = (1, 5, 2, 2) on T3 can complete
- Assuming T3 has completed its execution then,

Available = (1, 5, 2, 2) + (1, 2, 0, 1) = (2, 7, 2, 3) T1, T2 and T4 can execute to the completion.

1) We now consider the case <T3, T1, … >. That is, when T1 completes after T3

Available = (1, 5, 2, 2) + (0, 1, 1, 2) = (1, 6, 3, 4) not thread can complete

2) Consider the case <T3, T2> that is when T2 completes its execution after T3

Available = (1, 5, 2, 2) + (1, 2, 4, 0) = (2, 7, 6, 2), T1 and T4 can complete

- Assuming T1 has completed <T3, T2, T1>,

Available = (2, 7, 6, 2) + (0, 1, 1, 2) = (2, 8, 7, 4) T4 can complete

➢ Assuming T4 has completed <T3, T2, T1, T4> then,

Available = (2, 8, 7, 4) + (1, 0, 0, 1) = (3, 8, 7, 5) T0 can complete <T3, T2, T1, T4, T0>

- Assuming T4 has completed <T3, T2, T4> then

Available = (2, 7, 6, 2) + (1, 0, 0, 1) = (3, 7, 6, 3) only T1 can complete

➢ Assuming T1 has completed <T3, T2, T4, T1> then,

Available = (3, 7, 6, 3) + (0, 1, 1, 2) = (3, 8, 7, 5) T0 can complete <T3, T2, T4, T1, T0>

3) Consider the case <T3, T4>

Available = (2, 7, 2, 3) + (1, 0, 0, 1) = (3, 7, 2, 4) so, T0, T1 and T2 can complete.

<T3, T4, any permutation of T0, T1 and T2 here>

8.28 Consider the following snapshot of a system:

	Allocation	***Max***	***Available***
	A B C D	*A B C D*	*A B C D*
*T*0	3 1 4 1	6 4 7 3	2 2 2 4
*T*1	2 1 0 2	4 2 3 2	
*T*2	2 4 1 3	2 5 3 3	
*T*3	4 1 1 0	6 3 3 2	
*T*4	2 2 2 1	5 6 7 5	

Answer the following questions using the banker's algorithm:

a. Illustrate that the system is in a safe state by demonstrating an order in which the threads may complete.
b. If a request from thread *T*4 arrives for (2, 2, 2, 4), can the request be granted immediately?
c. If a request from thread *T*2 arrives for (0, 1, 1, 0), can the request be granted immediately?
d. If a request from thread *T*3 arrives for (2, 2, 1, 2), can the request be granted immediately?

a. The system is in a **safe state**, and the threads can complete in the following order: T2, T1, T3, T0, T4.
b. Yes, the request from thread T4 for (2, 2, 2, 4) can be granted immediately.
c. Yes, the request from thread T2 for (0, 1, 1, 0) can be granted immediately.
d. Yes, the **request** from thread T3 for (2, 2, 1, 2) can be granted immediately.

How to answer the questions
a. The **safe sequence** is T2, T1, T3, T0, T4.
b. If a request from thread T4 arrives for (2,2,2,4), can the request be granted immediately?
To check if the request can be granted, we need to compare the request (2 2 2 4) with the available resources (2 2 2 4) and the need of thread T4 (3 4 5 4). Since the request is less than or equal to both the available resources and the need, the request can be granted immediately.
c. If a request from thread T2 arrives for (0,1,1,0), can the request be granted immediately?
To check if the request can be granted, we need to compare the request (0 1 1 0) with the **available resources** (2 2 2 4) and the need of thread T2 (0 1 2 0). Since the request is less than or equal to both the available resources and the need, the request can be granted immediately.
d. If a request from **thread T3** arrives for (2,2,1,2), can the request be granted immediately?
To check if the request can be granted, we need to compare the request (2 2 1 2) with the available resources (2 2 2 4) and the need of thread T3 (2 2 2 2). Since the request is less than or equal to both the available resources and the need, the request can be granted immediately.

8.20 In a real computer system, neither the resources available nor the demands of threads for resources are consistent over long periods (months). Resources break or are replaced, new processes and threads come and go, and new resources are bought and added to the system. If deadlock is controlled by the banker's algorithm, which of the following changes can be made safely (without introducing the possibility of deadlock), and under what circumstances?

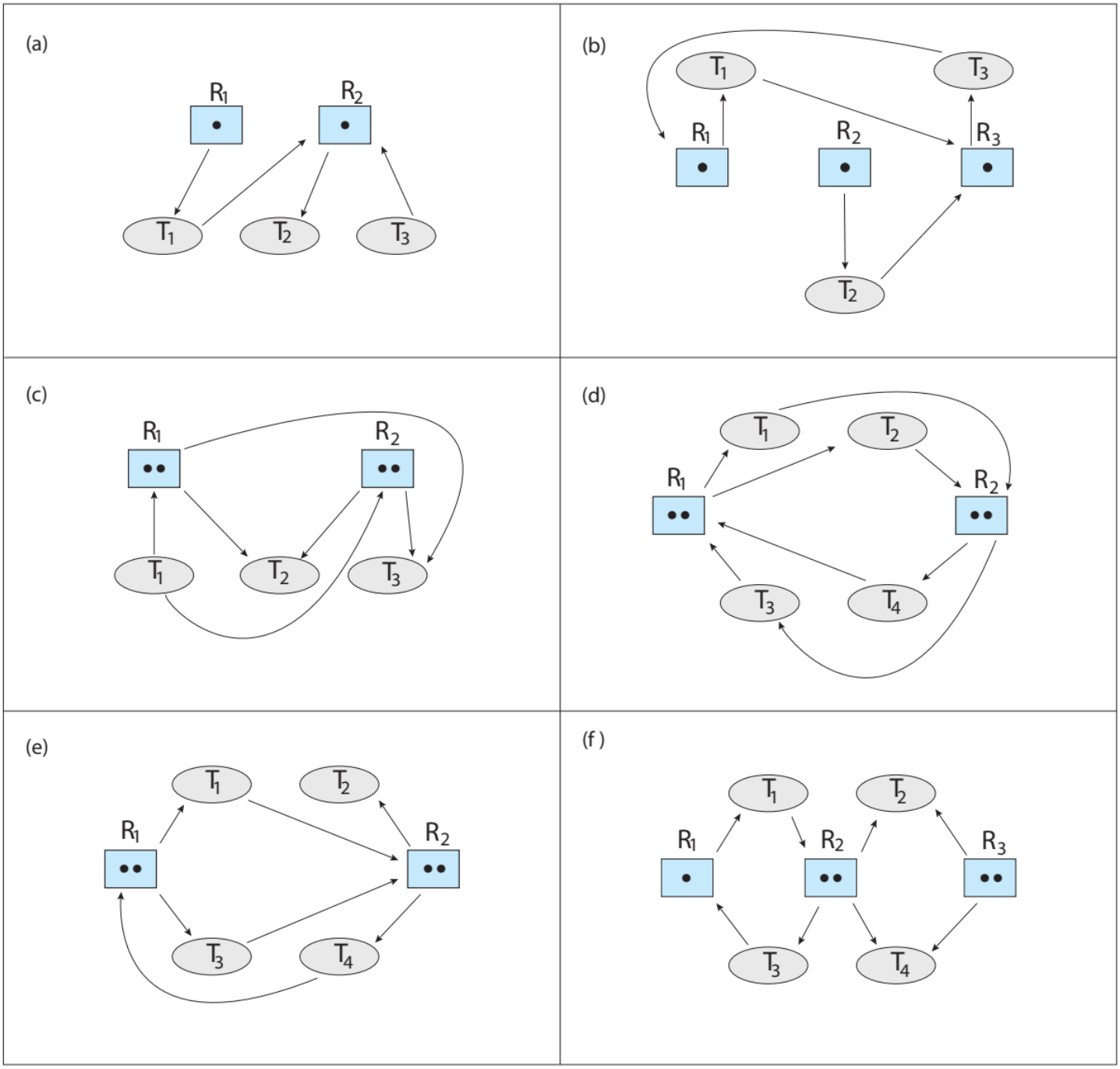


Figure 8.12 Resource-allocation graphs for Exercise 8.18.

a. Increase ***Available*** (new resources added).
b. Decrease ***Available*** (resource permanently removed from system).
c. Increase ***Max*** for one thread (the thread needs or wants more resources than allowed).
d. Decrease ***Max*** for one thread (the thread decides it does not need that many resources).
e. Increase the number of threads.
f. Decrease the number of threads.

a) Increase Available (new resources added)- This could safely be changed without any problems.
b) Decrease Available (resource permanently removed from system)- This could have an effect on the system and introduce the possibility of deadlock as the safety of the system assumed there were a certain number of available resources.
c) Increase Max for one process (the process needs more resources than allowed, it may want more)-This could have an effect on the system and introduce the possibility of deadlock.
d) Decrease Max for one process (the process decides it does not need that many resources)—This could safely be changed without any problems.
e) Increase the number of processes—This could be allowed assuming that resources were allocated to the new process such that the system does not enter an unsafe state.
f) Decrease the number of processes—This could safely be changed without any problems.

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

✍ Consider a logical address space of eight pages of 1024 words each, mapped onto a physical memory of 32 frames.

a. How many bits are there in the logical address?
b. How many bits are there in the physical address?

✍ Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of the First-fit, Best-fit and worst fit place processes of 210 KB, 415 KB, 100 KB, and 586 KB (in order)?

First-fit

	100, 500, 200, 300, 600
P1(210)	100, 290,200, 300, 600
P2(415)	100, 290, 200, 300, 185
P3(100)	0, 290, 200, 300, 185
P4(586)	Wait

Best-fit

	100, 500, 200, 300, 600
P1(210)	100, 500, 200, 90, 600
P2(415)	100, 85, 200, 90, 600
P3(100)	0, 85, 200, 90, 600
P4(586)	0, 85, 200, 90, 14

Worst-fit

	100, 500, 200, 300, 600
P1(210)	100, 500, 200, 300, 390
P2(415)	100, 85, 200, 300, 390
P3(100)	100, 85, 200, 300, 290
P4(586) wait	

✍ Consider the following segment table

What are the physical addresses for the following logical addresses: 0.31; 1.300; 2.10; 3.678; 4.1025

Segment	Base	Limit
0	114	64
1	400	200
2	1200	19
3	1540	800
4	1300	768

0,31
114 + 31 = 145
1,300
Illegal address
2,10
1200 + 10 = 1210
3,678
1540 + 678 = 2218
4,1025
Illegal address

✍ What are the physical addresses in the multi-partition for the following addresses:
115, 256, 2048, 300, 512, 3002, 3000, 6234, 1024, 1111. The base register is 0x2357, the limit register is 5120.

Thanh ghi nền: 0x2357 = 9047 = 2*16^3+3*16^2+5*16+7, thanh ghi giới hạn: 5120
115
9047 + 115 = 9162
256
9047 + 256 = 9303
2048
9047 + 2048 = 11095
300
9047 + 300 = 9347
512
9047 + 512 = 9559
3002
9047 + 3002 = 12049
3000
9047 + 3000 = 12047
6234
Illegal Address
1024
9047 + 1024 = 10071
1111
9047 + 1111 = 10158

9.4 Consider a logical address space of 64 pages of 1,024 words each, mapped onto a physical memory of 32 frames.
a. How many bits are there in the logical address?
b. How many bits are there in the physical address?

a. Logical address: 16 bits
b. Physical address: 15 bits

9.6 Given six memory partitions of 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, and 125 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 115 KB, 500 KB, 358 KB, 200 KB, and 375 KB (in order)?

Answer:

a. **First fit**:
- 115 KB is put in 300-KB partition, leaving 185 KB, 600 KB, 350 KB, 200 KB, 750 KB, 125 KB
- 500 KB is put in 600-KB partition, leaving 185 KB, 100 KB, 350 KB, 200 KB, 750 KB, 125 KB
- 358 KB is put in 750-KB partition, leaving 185 KB, 100 KB, 350 KB, 200 KB, 392 KB, 125 KB
- 200 KB is put in 350-KB partition, leaving 185 KB, 100 KB, 150 KB, 200 KB, 392 KB, 125 KB
- 375 KB is put in 392-KB partition, leaving 185 KB, 100 KB, 150 KB, 200 KB, 17 KB, 125 KB

b. **Best fit**:
- 115 KB is put in 125-KB partition, leaving 300 KB, 600 KB, 350 KB, 200KB, 750 KB, 10 KB
- 500 KB is put in 600-KB partition, leaving 300 KB, 100 KB, 350 KB, 200 KB, 750 KB, 10 KB
- 358 KB is put in 750-KB partition, leaving 300 KB, 100 KB, 350 KB, 200 KB, 392 KB, 10 KB
- 200 KB is put in 200-KB partition, leaving 300 KB, 100 KB, 350 KB, 0 KB, 392 KB, 10 KB
- 375 KB is put in 392-KB partition, leaving 300 KB, 100 KB, 350 KB, 0 KB, 17 KB, 10 KB

c. **Worst fit**:
- 115 KB is put in 750-KB partition, leaving 300 KB, 600 KB, 350 KB, 200 KB, 635 KB, 125 KB
- 500 KB is put in 635-KB partition, leaving 300 KB, 600 KB, 350 KB, 200 KB, 135 KB, 125 KB
- 358 KB is put in 600-KB partition, leaving 300 KB, 242 KB, 350 KB, 200 KB, 135 KB, 125 KB
- 200 KB is put in 350-KB partition, leaving 300 KB, 242 KB, 150 KB, 200 KB, 135 KB, 125 KB
- 375 KB must wait

9.7 Assuming a 1-KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers):
a. 3085
b. 42095
c. 215201
d. 650000
e. 2000001

Answer:
a. page = 3; offset = 13
b. page = 41; offset = 111
c. page = 210; offset = 161
d. page = 634; offset = 784
e. page = 1953; offset = 129

9.8 The BTV operating system has a 21-bit virtual address, yet on certain embedded devices, it has only a 16-bit physical address. It also has a 2-KB page size. How many entries are there in each of the following?
a. A conventional, single-level page table
b. An inverted page table
What is the maximum amount of physical memory in the BTV operating system?

Answer:
Conventional, single-level page table will have 2^{10} = 1024 entries. Inverted page table will have 2^5 = 32 entries. The maximum amount of physical memory is 2^{16} = 65536 (or 64 KB.)

9.9 Consider a logical address space of 256 pages with a 4-KB page size, mapped onto a physical memory of 64 frames.
a. How many bits are required in the logical address?
b. How many bits are required in the physical address?

Answer:
a. 12 + 8 = 20 bits.
b. 12 + 6 = 18 bits.

9.10 Consider a computer system with a 32-bit logical address and 4-KB page size. The system supports up to 512 MB of physical memory. How many entries are there in each of the following?
a. A conventional, single-level page table
b. An inverted page table

Answer:
a. 220 entries.
b. 512 K K/4K = 128K entries.

9.13 Given six memory partitions of 100 MB, 170 MB, 40 MB, 205 MB, 300 MB, and 185 MB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 200 MB, 15 MB, 185 MB, 75 MB, 175 MB, and 80 MB (in order)? Indicate which—if any—requests cannot be satisfied. Comment on how efficiently each of the algorithms manages memory.

Given memory spaces P1=200MB, P2= 15MB, P3=185MB, P4=75MB, P5=175MB, and P6=80MB.
Available spaces F1=100MB, F2=170MB, F3=40MB, F4205MB, F5=300MB, and F6=185MB.
Three proposed algorithms were used to define memory space and properly labelled processes.

First-fit algorithm

P1 allocated to F4 with remaining space of 205-200 = 5MB.
P2 allocated to F1 with remaining space of 100-15= 85MB
P3 allocated to F5 with remaining space of 300-185= 115MB.
P4 allocated to F1 with remaining space of 85-75= 10MB
P5 allocated to F6 with remaining space of 185-175=10MB
P6 allocated to F2 with remaining space of 170-80= 90MB

Best-fit algorithm

P1 allocated to F4 with remaining space of 205-200 = 5MB.
P2 allocated to F3 with remaining space of 40-15= 85MB
P3 was allocated to F6 but is entirely occupied, so no remaining space will be there

P4 allocated to F1 with remaining space of 100-75= 25MB
P5 allocated to F5 with remaining space of 300-175= 125MB
P6 allocated to F5 with remaining space of 125-80=45MB.

Worst-fit Algorithm

P1 allocated to F5 with remaining space of 300-200 = 100MB.
P2 allocated to F4 with remaining space of 205-15=190MB P3 allocated to F4 with the remaining space of 190-185= 5MB.
P4 allocated to F6 with remaining space of 185-75=110MB.
P5 does not contain any memory available spaces.
P6 allocated to F2 with remaining space of 170-80= 90MB.
According to this, the best-fit algorithm located and manages the memory very well whereas the worst-fit algorithm does not allocate the memory properly.

9.21 Assuming a 1-KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers)?
a. 21205
b. 164250
c. 121357
d. 16479315
e. 27253187

To determine the page numbers and offsets for the given address references, we need to consider the page size of 1-kilobyte (1KB).

To find the page number, we divide the decimal address by the page size. The quotient gives us the page number, and the remainder gives us the offset within that page.

Let's calculate the page numbers and offsets for each address reference:

(a) Address 21205:
Page Number = 21205 / 1024 = 20
Offset = 21205 % 1024 = 205

Therefore, for address 21205, the page number is 20 and the offset is 205.

(b) Address 164250:
Page Number = 164250 / 1024 = 160
Offset = 164250 % 1024 = 362

For address 164250, the page number is 160 and the offset is 362.

(c) Address 121357:
Page Number = 121357 / 1024 = 118
Offset = 121357 % 1024 = 389

So, for address 121357, the page number is 118 and the offset is 389.

(d) Address 16479315:
Page Number = 16479315 / 1024 = 16077
Offset = 16479315 % 1024 = 931

For address 16479315, the page number is 16077 and the offset is 931.

(e) Address 27253187:
Page Number = 27253187 / 1024 = 26613
Offset = 27253187 % 1024 = 619

Finally, for address 27253187, the page number is 26613 and the offset is 619.

To summarize:
(a) Page number: 20, Offset: 205
(b) Page number: 160, Offset: 362
(c) Page number: 118, Offset: 389
(d) Page number: 16077, Offset: 931
(e) Page number: 26613, Offset: 619

By using the page size and performing the division and modulo operations, we can find the page numbers and offsets for the given address references.

9.22 The MPV operating system is designed for embedded systems and has a 24-bit virtual address, a 20-bit physical address, and a 4-KB page size. How many entries are there in each of the following?
a. A conventional, single-level page table
b. An inverted page table
What is the maximum amount of physical memory in the MPV operating system?

a. Conventional, single-level page table:
The number of entries in a conventional, single-level page table is equal to the number of pages that can be addressed in the virtual memory.
Given that the virtual address is 24-bit and the page size is 4-KB (which is 2^{12} bytes), the number of pages in the virtual memory is $2^{(24-12)} = 2^{12}$.
So, there are **4096 entries** in a conventional, single-level page table.
b. Inverted page table:
The number of entries in an inverted page table is equal to the number of frames in the physical memory.
Given that the physical address is 20-bit and the page size is 4-KB (which is 2^{12} bytes), the number of frames in the physical memory is $2^{(20-12)} = 2^{8}$.
So, there are **256 entries** in an inverted page table.
Maximum amount of physical memory in the MPV operating system:
The maximum amount of physical memory that can be addressed is determined by the size of the physical address.
Given that the physical address is 20-bit, the maximum amount of physical memory that can be addressed is 2^{20} bytes, which is **1 Megabyte (MB)**.
Please note that these calculations assume that every bit in the address can be used and that there are no reserved addresses or other constraints on the address space. If such constraints exist, the actual amount of addressable memory could be less.

9.23 Consider a logical address space of 2,048 pages with a 4-KB page size, mapped onto a physical memory of 512 frames.
a. How many bits are required in the logical address?
b. How many bits are required in the physical address?

a. The number of bits required in the logical address can be determined by calculating the total number of pages in the logical address space. In this case, the logical address space has 2,048 pages. To find the number of bits required, we can calculate the logarithm base 2 of 2,048.

log2(2048) = 11

Therefore, 11 bits are required in the logical address.

b. The number of bits required in the physical address can be determined by calculating the total number of frames in the physical memory. In this case, the physical memory has 512 frames. To find the number of bits required, we can calculate the logarithm base 2 of 512.

log2(512) = 9

Therefore, 9 bits are required in the physical address.

c. The maximum amount of physical memory in this system can be determined by multiplying the number of frames by the page size. In this case, the page size is 4 KB.

Maximum amount of physical memory = 512 frames * 4 KB/frame
= 512 * 4 KB
= 2048 KB

Therefore, the maximum amount of physical memory in this system is 2048 KB.

9.24 Consider a computer system with a 32-bit logical address and 8-KB page size. The system supports up to 1 GB of physical memory. How many entries are there in each of the following?
a. A conventional, single-level page table
b. An inverted page table

A conventional, single-level page table in a system with a 32-bit logical address and 8-KB page size would have 8388608 entries. An inverted page table in the same system would have 2097152 entries.

Explanation:
The logical address space in a paging system is divided into fixed-size units called pages. The number of entries in a single-level page table corresponds with the number of pages that can be addressed.

To calculate this, we can use the formula: Number of pages = Size of logical address space / Page size. Here, the logical address is 32-bit, and the page size is 8-KB (or 8192 bytes, since 1 KB = 1024 bytes). Given a 32-bit logical address, the maximum addressable space is 2^32 bytes. Hence, the Number of pages = 2^32 / 8192 = 2^23 = 8388608 entries. So, a conventional, single-level page table would have 8388608 entries.

An inverted page table, however, has one entry per frame in physical memory, rather than per page of logical address space. Given that there is 1 GB (or 2^30 bytes) of physical memory and a page size of 8-KB, the number of frames is 2^30 / 8192 = 2^21 = 2097152 entries. So, an inverted page table would have 2097152 entries.

9.25 Consider a paging system with the page table stored in memory.
a. If a memory reference takes 50 nanoseconds, how long does a paged memory reference take?
b. If we add TLBs, and if 75 percent of all page-table references are found in the TLBs, what is the effective memory reference time? (Assume that finding a page-table entry in the TLBs takes 2 nanoseconds, if the entry is present.)

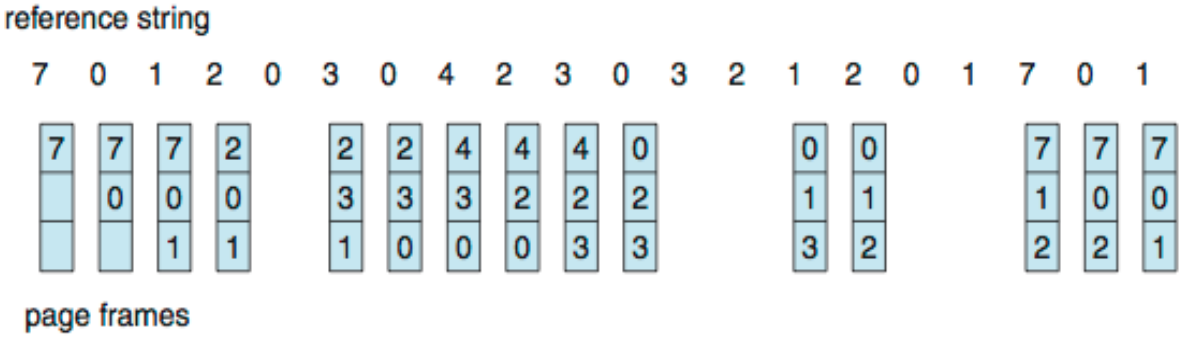
a. 100 ns : 50 ns for accessing page table , 50 ns for accessing the target data in the memory.

b. timeavg = 0.75∗(50ns+2ns) + 0.25∗(2ns+50ns+50ns) = 64.5ns

Chapter 10: Virtual Memory

First-In-First-Out (FIFO) Algorithm

- Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
- 3 frames (3 pages can be in memory at a time per process)



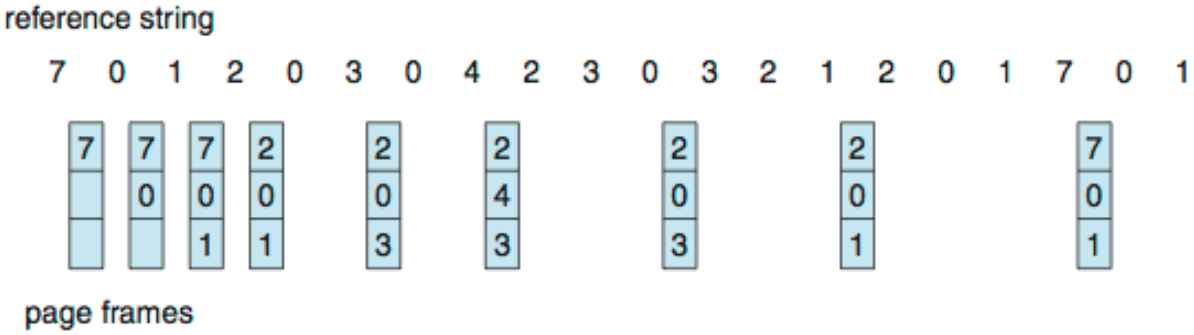
15 page faults

- Can vary by reference string: consider 1,2,3,4,1,2,5,1,2,3,4,5
  - Adding more frames can cause more page faults!
    - **Belady's Anomaly**

- How to track ages of pages?
  - Just use a FIFO queue

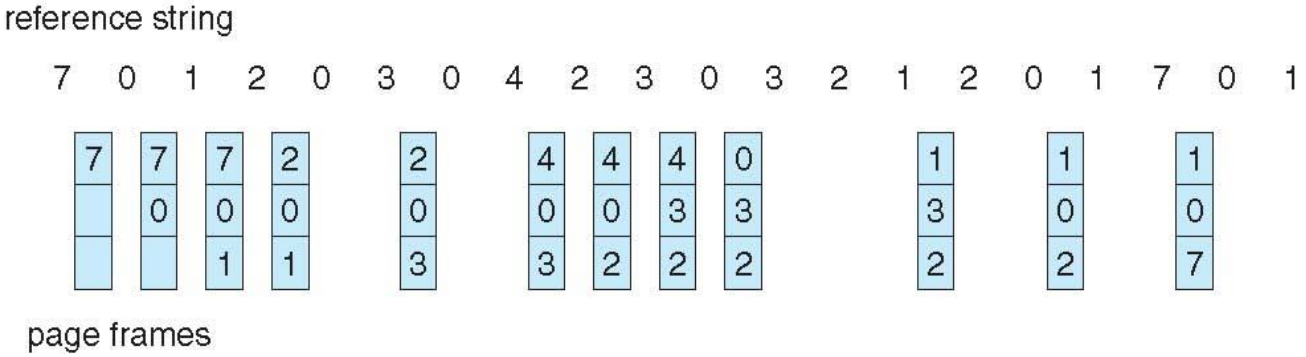
Optimal Algorithm (OPT)

- Replace page that will not be used for longest period of time
  - 9 is optimal for the example
- How do you know this?
  - Can't read the future
- Used for measuring how well your algorithm performs



Least Recently Used (LRU) Algorithm

- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time
- Associate time of last use with each page



- 12 faults – better than FIFO but worse than OPT
- Generally good algorithm and frequently used
- But how to implement?

10.4 An operating system supports a paged virtual memory. The central processor has a cycle time of 1 microsecond. It costs an additional 1 microsecond to access a page other than the current one. Pages have 1,000 words, and the paging device is a drum that rotates at 3,000 revolutions per minute and transfers 1 million words per second. The following statistical measurements were obtained from the system:

- One percent of all instructions executed accessed a page other than the current page.
- Of the instructions that accessed another page, 80 percent accessed a page already in memory.
- When a new page was required, the replaced page was modified 50 percent of the time.

Calculate the effective instruction time on this system, assuming that the system is running one process only and that the processor is idle during drum transfers.

Answer:

effective access time = 0.99 × (1 μsec + 0.008 × (2 μsec) + 0.002 × (10,000 μsec + 1,000 μsec) + 0.001 × (10,000 μsec + 1,000 μsec)
= (0.99 + 0.016 + 22.0 + 11.0) μsec
= 34.0 μsec

10.5 Consider the page table for a system with 12-bit virtual and physical addresses and 256-byte pages.

Page	Page Frame
0	–
1	2
2	C
3	A
4	–
5	4
6	3
7	–
8	B
9	0

The list of free page frames is *D*, *E*, *F* (that is, *D* is at the head of the list, *E* is second, and *F* is last). A dash for a page frame indicates that the page is not in memory.

Convert the following virtual addresses to their equivalent physical addresses in hexadecimal. All numbers are given in hexadecimal.

- 9EF
- 111
- 700
- 0FF

Answer:

- 9*EF* → 0*EF*
- 111 → 211
- 700 → *D*00
- 0*FF* → *EFF*

10.8 Consider the following page reference string:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

How many page faults would occur for the following replacement algorithms, assuming one, two, three, four, five, six, and seven frames? Remember that all frames are initially empty, so your first unique pages will cost one fault each.

- LRU replacement
- FIFO replacement
- Optimal replacement

Answer:

Number of frames	LRU	FIFO	Optimal
1	20	20	20
2	18	18	15
3	15	16	11
4	10	14	8
5	8	10	7
6	7	10	7
7	7	7	7

10.9 Consider the following page reference string:

7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0, 1.

Assuming demand paging with three frames, how many page faults would occur for the following replacement algorithms?

- LRU replacement
- FIFO replacement
- Optimal replacement

Answer:

- 18
- 17
- 13

10.13 Consider a demand-paged computer system where the degree of multiprogramming is currently fixed at four. The system was recently measured to determine utilization of the CPU and the paging disk. Three alternative results are shown below. For each case, what is happening? Can the degree of multiprogramming be increased to increase the CPU utilization? Is the paging helping?

a. CPU utilization 13 percent; disk utilization 97 percent
b. CPU utilization 87 percent; disk utilization 3 percent
c. CPU utilization 13 percent; disk utilization 3 percent

Answer:

a. Thrashing is occurring.
b. CPU utilization is sufciently high to leave things alone and increase the degree of multiprogramming.
c. Increase the degree of multiprogramming.

10.18 The following is a page table for a system with 12-bit virtual and physical addresses and 256-byte pages. Free page frames are to be allocated in the order 9, F, D. A dash for a page frame indicates that the page is not in memory.

Page	Page Frame
0	0 x 4
1	0 x B
2	0 x A
3	–
4	–
5	0 x 2
6	–
7	0 x 0
8	0 x C
9	0 x 1

Convert the following virtual addresses to their equivalent physical addresses in hexadecimal. All numbers are given in hexadecimal. In the case of a page fault, you must use one of the free frames to update the page table and resolve the logical address to its corresponding physical address.

- 0x2A1
- 0x4E6
- 0x94A
- 0x316

- 0x2A1 → 0xAA1
- 0x4E6 → 0x9E6
- 0x94A → 0x14A
- 0x316 → 0xF16

10.20 A certain computer provides its users with a virtual memory space of 232 bytes. The computer has 222 bytes of physical memory. The virtual memory is implemented by paging, and the page size is 4,096 bytes. A user process generates the virtual address 11123456. Explain how the system establishes the corresponding physical location. Distinguish between software and hardware operations.

Answer: The virtual address in binary form is 0001 0001 0001 0010 0011 0100 0101 0110 Since the page size is 2^{12}, the page table size is 2^{20}. Therefore the low-order 12 bits “0100 0101 0110” are used as the displacement into the page, while the remaining 20 bits “0001 0001 0001 0010 0011” are used as the displacement in the page table.

10.21 Assume that we have a demand-paged memory. The page table is held in registers. It takes 8 milliseconds to service a page fault if an empty frame is available or if the replaced page is not modified and 20 milliseconds if the replaced page is modified. Memory-access time is 100 nanoseconds.

Assume that the page to be replaced is modified 70 percent of the time. What is the maximum acceptable page-fault rate for an effective access time of no more than 200 nanoseconds?

0.2 _sec = (1 − P) × 0.1 _sec + (0.3P) × 8 millisec + (0.7P) × 20 millisec

0.1 = −0.1P + 2400 P + 14000 P
0.1 _ 16,400 P
P _ 0.000006

10.22 Consider the page table for a system with 16-bit virtual and physical addresses and 4,096-byte pages.

Page	Page Frame	Reference Bit
0	9	0
1	–	0
2	10	0
3	15	0
4	6	0
5	13	0
6	8	0
7	12	0
8	7	0
5	10	0
4	11	0
1	12	0
0	13	0
2	15	0
9	0	–
14	0	–

The reference bit for a page is set to 1 when the page has been referenced. Periodically, a thread zeroes out all values of the reference bit. A dash for a page frame indicates that the page is not in memory. The page-replacement algorithm is localized LRU, and all numbers are provided in decimal.
a. Convert the following virtual addresses (in hexadecimal) to the
equivalent physical addresses. You may provide answers in either
hexadecimal or decimal. Also set the reference bit for the appropriate entry in the page table.
• 0x621C
• 0xF0A3
• 0xBC1A
• 0x5BAA
• 0x0BA1
b. Using the above addresses as a guide, provide an example of a logical address (in hexadecimal) that results in a page fault.
c. From what set of page frames will the LRU page-replacement algorithm choose in resolving a page fault?

a. Convert the following virtual addresses (in hexadecimal) to the equivalent physical addresses:
Given that the page size is 4,096 bytes, the lower 12 bits of the virtual address are used as the offset within the page, and the upper 4 bits are used as the page number.
Here's how we can do it:
1. Virtual Address: 0x621C
- Page number = 6 (from the upper 4 bits of the virtual address)
- Offset = 21C (from the lower 12 bits of the virtual address)

The page table shows that page 6 is in frame 8. So, the physical address is 821C.
2. Virtual Address: 0xF0A3
- Page number = 15 (from the upper 4 bits of the virtual address)
- Offset = 0A3 (from the lower 12 bits of the virtual address)

The page table shows that page 15 is not in memory (indicated by a dash). This is a page fault. We need to allocate a free frame to this page.
3. Virtual Address: 0xBC1A
- Page number = 11 (from the upper 4 bits of the virtual address)
- Offset = C1A (from the lower 12 bits of the virtual address)

The page table does not have an entry for page 11. This is a page fault. We need to allocate a free frame to this page.
4. Virtual Address: 0x5BAA

- Page number = 5 (from the upper 4 bits of the virtual address)
- Offset = BAA (from the lower 12 bits of the virtual address)

The page table shows that page 5 is in frame 13. So, the physical address is DBAA.

5. Virtual Address: 0x0BA1

- Page number = 0 (from the upper 4 bits of the virtual address)
- Offset = BA1 (from the lower 12 bits of the virtual address)

The page table shows that page 0 is in frame 13. So, the physical address is DBA1.

b. Using the above addresses as a guide, provide an example of a logical address (in hexadecimal) that results in a page fault.

Any logical address with a page number that is not in the page table or is marked as not in memory (with a dash) will result in a page fault. For example, a logical address starting with F (such as 0xF123) would result in a page fault because page 15 is not in memory according to the page table.

c. From what set of page frames will the LRU page-replacement algorithm choose in resolving a page fault?

The LRU (Least Recently Used) page-replacement algorithm will choose the page frame that has not been used for the longest time. This is determined by the reference bit in the page table. A page with a reference bit of 0 is a candidate for replacement. If all pages have a reference bit of 1, then the LRU algorithm will need additional information (such as a timestamp of the last reference) to determine which page to replace. In this case, since all reference bits are 0, any page could be a candidate for replacement.

10.24 Apply the (1) FIFO, (2) LRU, and (3) optimal (OPT) replacement algorithms for the following page-reference strings:

- 2, 6, 9, 2, 4, 2, 1, 7, 3, 0, 5, 2, 1, 2, 9, 5, 7, 3, 8, 5
- 0, 6, 3, 0, 2, 6, 3, 5, 2, 4, 1, 3, 0, 6, 1, 4, 2, 3, 5, 7
- 3, 1, 4, 2, 5, 4, 1, 3, 5, 2, 0, 1, 1, 0, 2, 3, 4, 5, 0, 1
- 4, 2, 1, 7, 9, 8, 3, 5, 2, 6, 8, 1, 0, 7, 2, 4, 1, 3, 5, 8
- 0, 1, 2, 3, 4, 4, 3, 2, 1, 0, 0, 1, 2, 3, 4, 4, 3, 2, 1, 0

Indicate the number of page faults for each algorithm assuming demand paging with three frames.

- 2, 6, 9, 2, 4, 2, 1, 7, 3, 0, 5, 2, 1, 2, 9, 5, 7, 3, 8, 5

  FIFO = 18, LRU = 17, OPT = 13
- 0, 6, 3, 0, 2, 6, 3, 5, 2, 4, 1, 3, 0, 6, 1, 4, 2, 3, 5, 7

  FIFO = 16, LRU = 19, OPT = 13
- 3, 1, 4, 2, 5, 4, 1, 3, 5, 2, 0, 1, 1, 0, 2, 3, 4, 5, 0, 1

  FIFO = 15, LRU = 16, OPT = 11
- 4, 2, 1, 7, 9, 8, 3, 5, 2, 6, 8, 1, 0, 7, 2, 4, 1, 3, 5, 8

  FIFO = 20, LRU = 20, OPT = 16
- 0, 1, 2, 3, 4, 4, 3, 2, 1, 0, 0, 1, 2, 3, 4, 4, 3, 2, 1, 0

  FIFO = 12, LRU = 11, OPT = 11

10.35 A page-replacement algorithm should minimize the number of page faults. We can achieve this minimization by distributing heavily used pages evenly over all of memory, rather than having them compete for a small number of page frames. We can associate with each page frame a counter of the number of pages associated with that frame. Then,
to replace a page, we can search for the page frame with the smallest counter.

a. Define a page-replacement algorithm using this basic idea. Specifically address these problems:

- What is the initial value of the counters?
- When are counters increased?
- When are counters decreased?
- How is the page to be replaced selected?

b. How many page faults occur for your algorithm for the following reference string with four page frames?

1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2.

c. What is the minimum number of page faults for an optimal pagereplacement strategy for the reference string in part b with four page frames?

a. Define a page-replacement algorithm addressing the problem of:
   1. initial value of the counters- 0
   2. Counters are increased - whenever a nee page is associated with that frame.
   3. Counters are decreased - whenever one of the pages associated with that frame is no longer required.
   4. How the page to be replaced is selected - find a frame with the smallest counter.Use FIFO for breaking ties.

b. 14 page faults

c. 11 page faults

10.36 Consider a demand-paging system with a paging disk that has an average access and transfer time of 20 milliseconds. Addresses are translated through a page table in main memory, with an access time of 1 microsecond per memory access. Thus, each memory reference through the page table takes two accesses. To improve this time, we have added an associative memory that reduces access time to one memory reference if the page-table entry is in the associative memory. Assume that 80 percent of the accesses are in the associative memory and that, of those remaining, 10 percent (or 2 percent of the total) cause page faults. What is the effective memory access time?

Access Time = (0.8) * (1 microsecond) + (0.18) * (2 microsecond) + (0.02) * (20002 microsecond)
= 401.2 microsecond
= 0.4 millisecond

Consider the following page reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
How many page faults would occur for the following replacement algorithms, assuming four frames? Remember all frames are initially empty, so your first unique pages will all cost one fault each.
a) LRU replacement
b) FIFO replacement
c) Optimal replacement