8.13 Compare the memory organization schemes of contiguous memory allocation, pure segmentation, and pure paging with respect to the following issues.

(a) External Fragmentation

(b) Internal Fragmentation

With contiguous memory as new processes are allocated and released, because they are tightly packed, processes leaving memory will leave gaps which may or may not be filled, thus contiguous memory suffers from external fragmentation.

Pure paging will similarly suffer from external fragmentation as a process segment is still aligned continuously in memory. Segments of processes which have been released my be replaced by smaller processes, still leaving holes.

By using pure paging external fragmentation can be removed as each processes is asigned discrete well defined, equal sized blocks of memory. Thus the system can guarantee that each block of memory is assigned to some process (if need be). However, pure paging will now suffer from internal fragmentation, pages not completely utilized by a process lead to a waste of space (internal fragmentation).

8.20 Assuming a 1-KB page size, what are the page numbers for the following address references (provided as decimal numbers):

(a) 3085

$$\frac{3085}{1024} = 3.012695 \Rightarrow 3$$

(b) 42095

$$\frac{42095}{1024} = 41.1084 \Rightarrow 41$$

(c) 215201

$$\frac{215201}{1024} = 210.1572 \Rightarrow 210$$

(d) 650000

$$\frac{650000}{1024} = 634.7656 \Rightarrow 634$$

(e) 2000001

$$\frac{2000001}{1024} = 1953.126 \Rightarrow 1953$$

8.23 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?

$$256 \times 4 \times 1024 = 1048576 \text{ logical memory locations } \Rightarrow log_2(1048576) = 20 \text{ bits}$$

(b) How many bits are required in the physical address?

$$64 \times 4 \times 1024 = 262144 \text{ physical memory locations } \Rightarrow log_2(262144) = 18 \text{ bits}$$

9.21 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?

(a) LRU Replacement

7_m	2_m	3_m	1_m	2_h	5_m	3_m	4_m	6_m	7_m	7_h	1_m	0_m	5_m	4_m	6_m	2_m	3_m	0_m	1_m
7	7	7	1	1	1	3	3	3	7	7	7	7	5	5	5	2	2	2	1
	2	2	2	2	2	2	4	4	4	4	1	1	1	4	4	4	3	3	3
		3	3	3	5	5	5	6	6	6	6	0	0	0	6	6	6	0	0

18 page faults.

(b) FIFO Replacement

7_m	2_m	3_m	1_m	2_h	5_m	3_h	4_m	6_m	7_m	7_h	1_m	0_m	5_m	4_m	6_m	2_m	3_m	0_m	1_m
7	7	7	1	1	1	1	1	6	6	6	6	0	0	0	6	2	2	2	1
	2	2	2	2	5	5	5	5	7	7	7	7	5	5	5	5	3	3	3
		3	3	3	3	3	4	4	4	4	1	1	1	4	4	4	4	0	0

17 page faults.

(c) Optimal Replacement

7_m	2_m	3_m	1_m	2_h	5_m	3_h	4_m	6_m	7_m	7_h	1_h	0_m	5_h	4_m	6_m	2_m	3_m	0_h	1_h
7	7	7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	2	2	2	5	5	5	5	5	5	5	5	5	4	6	2	3	3	3
		3	3	3	3	3	3	6	7	7	7	0	0	0	0	0	0	0	0

13 page faults.

9.27 Consider a demand-paging system with the following time-measured utilizations:
CPU utilization 20%
Paging disk 97.7%
Other I/O devices 5%
For each of the following, indicate whether it will (or is likely to) improve CPU utilization. Explain your answers.

(a) Install a faster CPU.
This will obviously not improve CPU utilization, as it stands, we are currently only using 20%, our current CPU has plenty of room for growth, having faster processing will only decrease current utilization.

(b) Install a bigger paging disk.
Could help, but not likely. Our high paging usage is due to insufficient main memory such that our processes need to frequently access the paging disk. Increasing the paging disk does not help solve this main problem.

(c) Increase the degree of multiprogramming.
This will not help. Increasing the degree of multiprogramming will require us to keep more pages in memory, but clearly as our utilization is 97.7% we cannot afford this, our other processes will suffer and hurt our CPU utilization.

(d) Decrease the degree of multiprogramming.
On the contrary, this will indeed help. Each process can keep pages around in memory and will decrease paging disk utilization. Less time will be spent by processes waiting for the paging disk and processes will then run more efficiently.

(e) Install more main memory.
This will probably help the most, our biggest bottle neck is waiting for the paging disk. With more memory, the paging disk will need to be used much less, therefore processes can continue executing longer increasing CPU utilization.

(f) Install a faster hard disk or multiple controllers with multiple hard disks.
We are only using 5% I/O utilization, improving the speed of I/O operations is not likely to help much as noted before our problem is with paging not I/O.

(g) Add pre paging to the page-fetch algorithms.
This could help, paging disk utilization is already very high, and pre paging does not come for free so this is not likely practical. At best this will allow processes to run sooner and for longer and may slightly bump CPU utilization.

(h) Increase the page size.
Again, this might help. But decreasing the degree of multiprogramming or installing more main memory would be a much better solution. The best this could do is allow a current running process to run for a little bit longer before it page faults. However paging disk operations will take longer, so these may cancel out the benefit.

9.32 What is the cause of thrashing? How does the system detect thrashing? Once it detects thrashing, what can the system do to eliminate this problem?

If a system does not meet the minimum number of pages required by a process, pages will continuously fault causing disk operations, which in turn cause context switches, which cause further page faults taking up necessary pages for the original process. This cycle of page faults leading to no net work is known as thrashing.
To eliminate this problem, the system can detect the thrashing by analyzing and evaluating CPU utilization relative to degree of multiprogramming. If thrashing occurs, the system can decrease the level of multiprogramming, allowing processes time to store all necessary pages in memory to perform their work (without a context switch interrupting and disrupting them).

9.34 Consider the parameter Δ used to define the working-set window in the working-set model. When Δ is set to a small value, what is the effect on the page-fault frequency and the number of active (non-suspended) processes currently executing in the system? What is the effect when Δ is set to a very high value?

With a small Δ value, required pages for a processes may not be met, causing processes to run which do not have the resources to run. This results in a large number of page faults, with many active processes.

Conversely, with a large Δ value, required pages for a process will be overestimated, this will decrease the current number of working processes, but each process will have the resources to run. Thus there will be a small number of page faults, and fewer processes, but due to the overestimating processes may not be scheduled even if there are enough resources available.