1. Contiguous Memory Allocation

(a) see figure on the last page

- Advantages:

  -- OS can easily move a process during execution.
  -- OS can allow a process to grow over time.
  -- Simple, fast hardware: two special registers, an add, and a compare.

- Disadvantages:

  -- Slows down hardware due to the add on every memory reference.
  -- Can't share memory (such as program text) between process.
  -- Process is still limited to physical memory size.
  -- Degree of multiprogramming is very limited since all memory of all active processess must fit in memory.
  -- Complicates memory management.

(b) **Fragmentation**

(i)According 50% rules --- for every 2N chunk of memory allocation, N chunks are wasted due to fragmentation, so statistically 1/3 of 128 MB that is 42 MB is wasted due to external fragmentation.

In theory, contiguous memory allocation doesn't suffer from internal fragmentation, since a process can be allocated exactly what it needs. But in practice, if the OS allocates memory than requested to avoid tracking small holes, a small fraction may be lost of the internal fragmentation

(ii)pure paging --- no external fragmentation
internal: ½ a page(4 KB) per process.

2. Page Replacement

(a) Belady's anomaly: Increasing the number of frames allocated to a process increases the number of page faults suffered by the process.

To avoid: any replacement algorithm that caches a superset of the pages cached with a small number of frames won't suffer from Belady's anomaly.

(b)

	A	B	C	D	A	B	E	A	B	C	D	E
frame1	A	A	A	A	A	A	A	A	A	C	D	D
frame2		B	B	B	B	B	B	B	B	B	B	B
frame3			C	D	D	D	E	E	E	E	E	E
page fault(y/n)	Y	Y	Y	Y	N	N	Y	N	N	Y	Y	N

Number of page faults = 7

	A	B	C	D	A	B	E	A	B	C	D	E
frame1	A	A	A	A	A	A	A	A	A	A	D	D
frame2		B	B	B	B	B	B	B	B	B	B	B
frame3			C	C	C	C	C	C	C	C	C	C
frame4				D	D	D	E	E	E	E	E	E
page fault(y/n)	Y	Y	Y	Y	N	N	Y	N	N	N	Y	N

Number of page faults = 6

(c) OPT should not suffer from Belady's anomaly.
OPT cashes a superset of pages with 4 frames compared to 3 frames.

3.Paging and Segmentation Paging

(a) page table = $2^8 / 2^5 = 2^3 = 8$ entries
physical memory = 2^9 bytes
number of frames = $2^9 / 2^5 = 2^4 \Rightarrow$ 4 bits for p
page size = $2^5 \Rightarrow$ 5 bits for d
p + d = 4 + 5 = 9 bits

(b) 3ma
h (s + t + ma) + (1 - h) (s + t + 2ma) (h is the hit rate)

4.File Systems

(a)

(i) totally 2 I/Os needed :
1 I/O to find a free block & update the free block list
1 I/O to write the block & link it to the 2nd block
Since the file descriptor is in memory, no I/O needed to update it, in practice the file descriptor is no disk, that would need another I/O.

(ii) totally 52 I/Os needed :
50 I/Os to get the 50th block
1 I/O needed to have the 49th block point to the 51st block & write out the 49th block
1 I/O needed to insert the freed block to the head of free block list & write it out.

(iii) totally 103 I/Os needed :
100 I/Os to get the 100th block
1 I/O to get a free block & update the free block list
1 I/O to link the 100th block to 101st block
1 I/O to write the 101st block out.

(b) $(12 + 1000 + 1000^2) * 8$

(c) Start with a bitmap with 1 bit for each block
Initially all bits are set to 0
Scan each file, traverse its linked list & set the bit of each block belonging to a file to 1
Now, all blocks allocated to files will have their bits set to 1
All blocks with bits 0 belong to the free block list
The head of the free block list can be fine by checking each free block & see if it points to another block, if so, mark that block as 1
At the end of this, you will be left with 1 free block that no other free block points to , this is the head.