1.

a) Page 3
b) Page 0
c) Page 0
d) Page 1

2.

In a Linux system, the size of a block and the number of blocks are stored in the superblock of the active partition. As for the used block information for a file, it's stored in its inode. Therefore, we can obtain information about used blocks by scanning all i-nodes.

```
Create new bitmap (size of bit = number of block from super block)
    Reset (set as 0)
For each i-node do
    For each entry of i-node do
        Set bitmap to 1 (based on used block information)
```

3.

a.

A multilevel page table reduces the number of actual pages of the page table that need to be in memory because of its hierarchic structure. In fact, in a program with lots of instruction and data locality, we only need the top level page table (one page), one instruction page, and one data page.

b.

- Since page size = 16KB =2^{14} , 14 bit for offset. That leaves 24 bits for the page fields.
- Since each entry is 4 bytes, one page can hold 2^{14} /4 = 2^{12} page table entries and therefore requires 12 bits to index one page. So allocating 12 bits for each of the page fields will address all 2^{38} bytes.
- 12 + 12 + 14

4.

Since 1 block is 4 KB, and 64bit = 8 Byte per block address, it can save $4 \times 2^{10} / 8 = 2^9 = 512$ block information

Total = 512 + 10 = 522 block information.

Since a block size is 4 KB, largest file will be 4 KB × 522 = 2088 KB

5.

a)

$$R = \begin{pmatrix} 0\ 0\ 0\ 0 \\ 0\ 7\ 5\ 0 \\ 1\ 0\ 0\ 2 \\ 0\ 0\ 2\ 0 \\ 0\ 6\ 4\ 2 \end{pmatrix},\ A = (1, 5, 2, 0)$$

b)

$$A = (1, 5, 2, 0) \xRightarrow{P_0} (1, 5, 3, 2) \xRightarrow{P_2} (2, 8, 8, 6) \xRightarrow{P_1} (3, 8, 8, 6) \xRightarrow{P_3} (3, 14, 11, 8) \xRightarrow{P_4} (3, 14, 12, 12)$$

c)

If granted, the snap shot will be change to

$$R = \begin{pmatrix} 0\ 0\ 0\ 0 \\ 0\ 3\ 3\ 0 \\ 1\ 0\ 0\ 2 \\ 0\ 0\ 2\ 0 \\ 0\ 6\ 4\ 2 \end{pmatrix}, C = \begin{pmatrix} 0\ 0\ 1\ 2 \\ 1\ 4\ 2\ 0 \\ 1\ 3\ 5\ 4 \\ 0\ 6\ 3\ 2 \\ 0\ 0\ 1\ 4 \end{pmatrix}, A = (1\ 1\ 0\ 0)$$

$$A = (1, 1, 0, 0) \xRightarrow{P_0} (1, 1, 1, 2) \xRightarrow{P_2} (2, 4, 6, 6) \xRightarrow{P_1} (3, 8, 8, 6) \xRightarrow{P_3} (3, 14, 11, 8) \xRightarrow{P_4} (3, 14, 12, 12)$$

6.

a.

1) Mutual exclusion
2) Hold-and Wait
3) No preemption
4) Circular wait

b.

1) Ignore
2) Detection and recovery
3) Avoidance with dynamic allocation
4) By attacking one of necessary deadlock condition

7. (5 pt.) About Log-Structured File System

a. files are cached in the RAM or swap area when it is opened.

b.

- In LSF, each i-node is not at a fixed location; they are written to the log.
- LFS uses a data structure called an **i-node map** to maintain the current location of each i-node.
- Opening a file consists of using the map to locate the i-node for the file.

8.

a.

- Maximum virtual address space = $2^{64} = 2^{54} \times 2^{10} = 2^{54}$ **KB**
- $\therefore$ Maximum # of pages per a process = virtual space / a page size = $2^{54}/16 = 2^{50}$ pages .
- Maximum size of page table per a process = number of page $\times$ one entry size

  $= 2^{50} \times 64$ bits $= (2^{50} \times 64)/8$ Byte$= 2^{50} \times 8$ byte$=$ **8** $\times 2^{50}$ Byte

b.

- need calculate the number of page frame

  # of page frame = size of RAM / size of page

  $= 32\text{GB} / 16\text{ KB} = 32 \times 2^{30} / 16 \times 2^{10} = 2^{21}$ page frames

  $\therefore$ **21 bits for page frame number.**

9. By using the deadlock detection algorithm, show that where there is a deadlock or not in the system.

$$[0\ 1\ 0\ 2\ 1] \xrightarrow{P_2} [0\ 2\ 0\ 3\ 1] \xrightarrow{P_3} [0\ 2\ 0\ 3\ 2]$$

10.

	Allocated	Need More	Available
	R_1 R_2 R_3	R_1 R_2 R_3	R_1 R_2 R_3
P_1	0 1 0	7 4 3	0 1 2
P_2	2 0 0	1 2 2	
P_3	3 0 2	6 0 0	
P_4	2 1 1	0 1 1	
P_5	3 2 2	1 1 1	

$$A = [0\ 1\ 2] \xrightarrow{P_4} [2\ 2\ 3] \xrightarrow{P_5} [5\ 4\ 5] \xrightarrow{P_2} [7\ 4\ 5] \xrightarrow{P_3} [10\ 4\ 7] \xrightarrow{P_1} [10\ \ 5\ \ 7] \quad \text{Safe}$$

or

$$A = [0\ 1\ 2] \xrightarrow{P_4} [2\ 2\ 3] \xrightarrow{P_2} [4\ 2\ 3] \xrightarrow{P_5} [7\ 4\ 5] \xrightarrow{P_3} [10\ 4\ 7] \xrightarrow{P_1} [10\ \ 5\ \ 7] \quad \text{Safe}$$

11.

a.

- Size of Each block = $8 \times 8 \times 2^{10}$ bits = 2^{16} bits
- One block can keep = size of block/size of a block address = 2^{16} bits / 32 bits =2^{16} / 2^5 = 2^{11} -1 = 2047 block information
- Total # of blocks in the disk = size of disk / block size
  = 128GB / 8KB blocks =128 × 2^{30} / 8 × 2^{10}
  = 2^7 × 2^{30} / 2^3 × 2^{10}= 2^{37}/ 2^{13}= 2^{24} blocks
- # of blocks need to keep track of free blocks = 2^{24} blocks /2047 = 8196.002
  ∴ 8197 blocks

b.

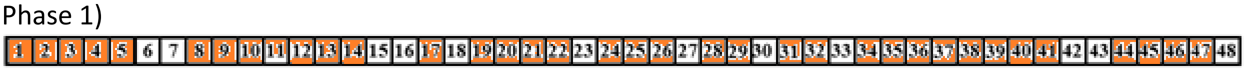
- Total # of blocks in the disk ==2^{24} blocks
- Need 2^{24} bits for bit map= 2^{24} / 8 = 2^{24} / 2^3 = 2^{21} Byte
- # of blocks need for bitmap = 2^{21} / (8 × 2^{10}) = 2^{21} / 2^{13}) = 2^8 blocks

c.

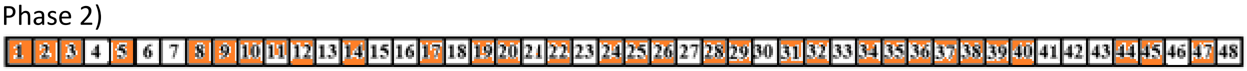
- Since this system use 32bit disk block number, this system support 2^{32} blocks
- Maximum disk size = 2^{32} × 8 × 2^{10} Byte =32 × 2^{40} = 32 TB

12.

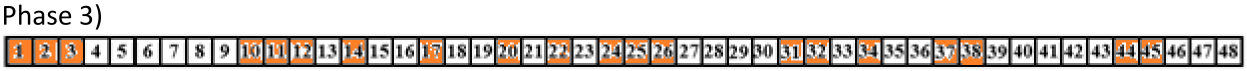
Phase 1)



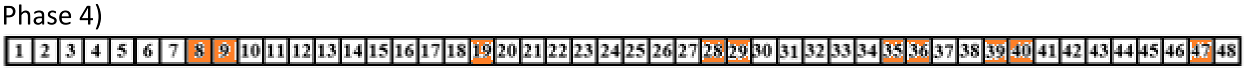
Phase 2)



Phase 3)



Phase 4)



13.

Seek time + rotation delay = 6 + 5 = 11 msec

Average file size = 8 × 2^{10} Byte = 2^{13} Byte,

Transfer rate = 16MB/sec = 16 × 2^{20} Byte/sec = 2^{24} Byte/sec

A file with average size can transfer 11 + (2^{13} Byte /2^{24} Byte/sec) × 10^3 = 11.49 msec

Read + write takes 11.49 + 11.49 = 22.98 msec

8KB takes 22.98 msec

16G / 8 K = 2 × 2^{20}, so 16GB space take 22.98 × 2^{21} msec = 48,192,552.96 msec = 48,192.55296 sec = 13.387 hour