Exercise 8.

Problem 1. (20 points) Suppose 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 page is available or 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% of the time.

What is the maximum acceptable page-fault rate for an effective access time of no more than 200 nanoseconds?

First, think about the page fault service time. Now, the page to be replaced is modified 70% of the time, and not modified 30% of the time. Then page fault service time is

page_fault_service_time = 20[ms] * 0.7 + 8[ms] * 0.3 = 14.0 + 2.4 = **16.4**[ms]

The effective access time(EAT) is calculated in this way(p is the page fault rate):

EAT = (1 – p) * memory_access_time + p * page_fault_service_time

Now, the memory access time is 100[ns], and the baseline time for EAT is 200[ns], then the allowed page-fault rate p at the EAT = 200[ns] is

200[ns] = (1 – p) * 100[ns] + p * 16.4[ms]

$p = 100[ns] / (16.4 * 10^6[ns] – 100[ns])$

$p = 0.0000060979 \fallingdotseq 6.1 * 10^{-6}$

To the maximum acceptable page-fault rate is $6.1 * 10^{-6}$.

To make the EAT no more than 200 nanoseconds, the page-fault rate for EAT mustn't be more than $6.1 * 10^{-6}$.

Problem 2. (40 points) 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 three and four frames ? Remember all frames are initially empty, so your first unique pages will all cost one fault each.

1. LRU replacement
2. FIFO replacement
3. Optimal replacement

# frames = 3 Number of page faults: 15																				
LRU	**1**	**2**	**3**	**4**	**2**	**1**	**5**	**6**	**2**	**1**	**2**	**3**	**7**	**6**	**3**	**2**	**1**	**2**	**3**	**6**
	1	1	1	4	4	4	5	5	5	1	1	1	7	7	7	2	2	2	2	2
		2	2	2	2	2	2	6	6	6	6	3	3	3	3	3	3	3	3	3

			3	3	3	1	1	1	2	2	2	2	2	6	6	6	1	1	1	6
Page Fault				1		3	4	2	1	5		6	1	2		7	6			1

# frames = 3 Number of page faults: 16																				
FIFO	**1**	**2**	**3**	**4**	**2**	**1**	**5**	**6**	**2**	**1**	**2**	**3**	**7**	**6**	**3**	**2**	**1**	**2**	**3**	**6**
	1	1	1	4	4	4	4	6	6	6	6	3	3	3	3	2	2	2	2	6
		2	2	3	3	3	5	5	5	1	1	1	1	6	6	6	6	6	3	3
			3	2	2	1	1	1	2	2	2	2	7	7	7	7	1	1	1	1
Page Fault				1		2	3	4	1	5		6	2	1		3	7		6	2

# frames = 3 Number of page faults: 11																				
OPT	**1**	**2**	**3**	**4**	**2**	**1**	**5**	**6**	**2**	**1**	**2**	**3**	**7**	**6**	**3**	**2**	**1**	**2**	**3**	**6**
	1	1	1	1	1	1	1	1	1	1	1	3	3	3	3	3	3	3	3	6
		2	2	2	2	2	2	2	2	2	2	2	7	7	7	2	2	2	2	2
			3	4	4	4	5	6	6	6	6	6	6	6	6	6	1	1	1	1
Page Fault				3			4	5				1	2			7	6			3

# frames = 4 Number of page faults: 10																				
LRU	**1**	**2**	**3**	**4**	**2**	**1**	**5**	**6**	**2**	**1**	**2**	**3**	**7**	**6**	**3**	**2**	**1**	**2**	**3**	**6**
	1	1	1	1	1	1	1	1	1	1	1	1	1	6	6	6	6	6	6	6
		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
			3	3	3	3	5	5	5	5	5	3	3	3	3	3	3	3	3	3
				4	4	4	4	6	6	6	6	6	7	7	7	7	1	1	1	1
Page Fault							3	4				5	6	1			7			

# frames = 4 Number of page faults: 14																				
FIFO	**1**	**2**	**3**	**4**	**2**	**1**	**5**	**6**	**2**	**1**	**2**	**3**	**7**	**6**	**3**	**2**	**1**	**2**	**3**	**6**
	1	1	1	1	1	1	5	5	5	5	5	3	3	3	3	3	1	1	1	1
		2	2	2	2	2	2	6	6	6	6	6	7	7	7	7	7	7	3	3
			3	3	3	3	3	3	2	2	2	2	2	6	6	6	6	6	6	6
				4	4	4	4	4	4	1	1	1	1	1	1	2	2	2	2	2
Page Fault							1	2	3	4		5	6	2		1	3		7	

# frames = 4 Number of page faults: 8																				
OPT	**1**	**2**	**3**	**4**	**2**	**1**	**5**	**6**	**2**	**1**	**2**	**3**	**7**	**6**	**3**	**2**	**1**	**2**	**3**	**6**
	1	1	1	1	1	1	1	1	1	1	1	1	7	7	7	7	1	1	1	1

		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
			3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
				4	4	4	5	6	6	6	6	6	6	6	6	6	6	6	6	6
Page Fault							4	5					1				7			

Exercise 9.

Problem 1 (points 30)**.**

A process has 4 page frames allocated to it. (All the following numbers are decimal, and everything is numbered starting from zero). The time of the last loading of a page into each page frame, time of last access, and the referenced (***R***) and modified (***M***) bits for each page are as shown below (the times are in clock ticks from the process start at time 0 to the event - not the number of ticks since the event to the present):

Page	Time Loaded	Time Last. ref.	*R*	*M*
0	210	255	1	1
1	130	281	1	0
2	155	273	0	0
3	187	261	1	1

A page fault to the page 4 has occurred.

(a) Which page will FIFO replace?

Page 1 will replaced first

(b) Which page will LRU replace?

Page 0 will replaced first

c. Which page will second chance replace?

Page 2 will replaced first

Problem 2. (points 70)

Consider the two-dimensional array **a** like this

long int a[150][150], (32 bits in length)

Where **a[0][0]** is at location 300, in a paged memory system with pages of size 300 (words; i.e. 1200 bytes). A small process is in page 0 (locations 0 to 299) for manipulating the matrix; thus, every instruction fetch will be from page 0. For three page frames, how many page faults are generated by the following array-initialization loops,

using LRU replacement, and assuming page frame 1 has the process in it, and the other two are initially empty.

```
/* C program 1 */

for(i = 0; i < 150; i++)

for(j = 0; j < 150; j++)

a[i][j] = 0;
```

In this case, loop access words for matrix in row major and store it in the page like...

{a[0][0], a[0][1], a[0][2], … , a[0][149], a[1][0], … , a[1][149]}

Then page fault happens when row changes or each words row access.

Thus, the number of page faults is 150 / 2 = 75.

```
/* C program 2 */

for(j = 0; j < 150; j++)

for(i = 0; i < 150; i++)

a[i][j] = 0;
```

In this case, loop access words for matrix in column major and store it in the page like...

{a[0][0], a[1][0], a[2][0], … , a[149][0], a[0][1], … , a[149][1]}

Then page fault happens when it access to all words a[i][j].

Thus, the number of page faults is 150 * (150 / 2) = 11250.

Exercise 10.

Problem 2. (10 points) Explain the purpose of the Open and Close operations.

1. Open operation

Prepare a file to be referenced.

2. Close operation

Prevent further reference to a file until it is reopened.

Problem 3. (40 points) Consider a system that supports 5000 users. Suppose you want to allow 4990 of these users to be able to access one file.

a) How would you specify this protection scheme in UNIX?

First, add these 10 users except 4990 users will able to access the file to the group. Then, I can allow access to the file only for users not added to the group: these 4990 users.

b) Could you suggest another protection scheme that can be used more effectively for this purpose than the scheme provided by UNIX?

As an another protection scheme, I can set the password to specific file. Then I can allow access only for these 4990 users by giving the password for them.

Problem 4. (25 points) Give an example of an application in which data in a file should be accessed in the following order:

a) Sequentially

Application access sequentially when it access whole content of the file.

For example, text editor such as Emacs access the data sequentially when it loads the program code. When it loads file content, whole part of the program code is loaded.

b) Randomly

Application access randomly when it access a specific part of the file.

For example, database program such as sqlite access the data in database randomly when there's a request with specific key. When the request comes, the database returns the data linked to the key.

Exercise 11.

Problem 1 (40points)

Consider a file currently consisting of 50 blocks numbered from 1 to 50. Assume that the file control block (and the index block, in case of indexed allocation) is already in memory. Calculate how many disk I/O operations are required for each operation in the table in the case of contiguous, linked, and indexed (single-level) allocation strategies. In the contiguous-allocation case, assume that there is no room to grow in the beginning, but there is room to grow in the end. Assume that the block information to be added is stored in memory.

Fill the following table: (show the number of read (r) and write (w) O/I operations separately)

		contiguous-allocation	linked allocation	indexed (single-level) allocation
a	A block is added in the beginning	r: 50 w: 51	r: 0 w: 1	r: 0 w: 1
b	A block is added in the middle	r: 25 w: 26	r: 25 w: 2	r: 0 w: 1
c	A block is added in the end	r: 0 w: 1	r: 1 w: 2	r: 0 w: 1
d	A block is removed from the beginning	r: 0 w: 0	r: 1 w: 0	r: 0 w: 0
e	A block is removed from the middle (block number 25)	r: 24 w: 24	r: 26 w: 1	r: 0 w: 0
f	A block is removed from the end	r: 0 w: 0	r: 49 w: 1	r: 0 w: 0

Problem 2 (35 points)

One way to use contiguous allocation of the disk and not suffer from holes is to compact the disk every time a file is removed. Since all files are contiguous, copying a file requires a seek and rotational delay to read the file, followed by the transfer at full speed.

Writing the file back requires the same work.

a) Assuming a seek time of 5 msec, a rotational delay of 4 msec, a transfer rate of 8 MB/sec, and an average file size of 8 KB, how long does it take to read a file into main memory then write it back to the disk at a new location?

It takes 9 msec to start the transfer due to 5 msec seek and 4msec rotation delay. To read 8KB(= 2^{13} bytes) at the transfer rate of 8 MB/sec(= 2^{23} bytes/sec) requires 2^{-10} sec (cause $(1/2^{23}) * 2^{13} = 2^{-10}$) = 0.97656 msec ≒ 0.977 msec.

Then, the total time to seek, rotate and transfer is 5 + 4 + 0.977 = 9.977 msec

Writing the file back needs same work, so it takes another 9.977 msec, then copying an average file takes 9.977 * 2 = **19.954 msec**.

b) Using these numbers, how long would it take to compact half of a 16-GB disk?

To compact half of a 16-GB dick would involve copying 8GB of storage, which contains 2^{20} files (cause 8GB (= 2^{33}B) / 8KB (= 2^{13}B) = 2^{20})average. Now, it takes 19.954 msec for every file, then it takes 2^{20} * 19.954 = 20923285.504 msec ≒ 20923 seconds.

Problem 3 (25 points)

The beginning of a free space bitmap looks like this after the disk partition is first formatted: 0111 1111 1111 1111 (the first block is used by the root directory). The system always searches for free blocks starting at the lowest numbered block, so after writing file A, which uses 6 blocks, the bitmap looks like this: 0000 0001 11111111. Show the bitmap after each of the following additional actions:

(a) File B is written, using 5 blocks Answer: 0000 0000 0000 1111

(b) File A is deleted Answer: 0111 1110 0000 1111

(c) File C is written, using 8 blocks Answer: 0000 0000 0000 0011

(d) File B is deleted. Answer: 0000 0001 1111 0011

Exercise 12.

Problem 1. (40 points)

Suppose that a disk drive has **5,000** cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder **143**, and the previous request was at cylinder **125**. The queue of pending requests, in FIFO order, is:

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests for each of the following disk-scheduling algorithms? Fill tables and count the total distance.

a) FCFS

Time	Cylinder number

	0	…	86	130	143	…	913	948	…	1022	1470	1509	1750	1774	…	4999
0					*											
1			*													
2											*					
3							*									
4														*		
5								*								
6												*				
7										*						
8													*			
9				*												

Total distance: 7081

b) SSTF

Time	**Cylinder number**															
	0	…	86	130	143	…	913	948	…	1022	1470	1509	1750	1774	…	4999
0					*											
1				*												
2			*													
3							*									
4								*								
5										*						
6											*					
7												*				

8													*			
9														*		

Total distance: 1745

c) SCAN

Time	Cylinder number															
	0	…	86	130	143	…	913	948	…	1022	1470	1509	1750	1774	…	4999
0					*											
1							*									
2								*								
3										*						
4											*					
5												*				
6													*			
7														*		
8																*
9				*												
10			*													

Total distance: 9769

d) LOOK

Time	Cylinder number															
	0	…	86	130	143	…	913	948	…	1022	1470	1509	1750	1774	…	4999
0					*											
1							*									
2								*								

3										*						
4											*					
5												*				
6													*			
7														*		
8				*												
9			*													

Total distance: 3319

e) C-SCAN

Time	**Cylinder number**															
	0	…	86	130	143	…	913	948	…	1022	1470	1509	1750	1774	…	4999
0					*											
1							*									
2								*								
3										*						
4											*					
5												*				
6													*			
7														*		
8																*
9	*															
10			*													

11				*												

Total distance: 9985

f) C-LOOK

Time	Cylinder number															
	0	…	86	130	143	…	913	948	…	1022	1470	1509	1750	1774	…	4999
0					*											
1							*									
2								*								
3										*						
4											*					
5												*				
6													*			
7														*		
8			*													
9				*												

Total distance: 3363

Problem 3. (25 points)
Consider a RAID Level 4 organization comprising five disks, with the parity for sets of four blocks on four disks stored on the fifth disk. How many disks are accessed in order to perform the following? Describe the process.

1. Updating one block of data on the first disk.

When it calculate the parity in horizontal line, it needs to access every four disks. In this case, it needed to write only one block on the first disk, but it also needed to access remaining 3 disks to re-calculate the parity on fifth block. And of cause it have to access fifth disk to write parity information.

Then, to update one block of data on the first disk, it's needed to access all of the 5 disks.

2. Writing seven continuous blocks of data starting from the first disk.

In RAID 4 organization, data is stored in horizontal line. In this case, every four disks for storing data has some parts of this continuous blocks. When it writes seven continuous blocks in this organization, it needs to access four disks to write main data and calculate the parity, and also access fifth disk to write re calculated parity information.

Then, to write seven continuous blocks of data starting from the first disk, it's needed to access all of the 5 disks.

Exercise 13.

Problem 3 (40 points).

There is a model to predict the performance of the Ethernet in terms of the percentage of time data packets are on the net. Let N be the number of systems (nodes) connected to the net and P be the probability that one system gains exclusive access at any point in time. Then,

(N systems have $1/N$ probability of gaining access at a time when the other N-1 systems, each with probability $1-1/N$, decide not to access the net).

This definition is used to define the mean time a station must wait before it gains access. The time unit, known as a *slot time*, is the amount of time a system waits before retransmitting when it finds the network busy. The probability of waiting 0 slots is equal to P; the probability of waiting one slot is. So, the probability of waiting i slot times is

The mean of the probability distribution over all i gives the expected number of slot times a system will have to wait before acquiring exclusive access to the Ethernet:

Finally, the efficiency of the network in this model is the percentage of time there are packets on the net. Let S be the packet size, in bits (the Ethernet is a bit-serial medium), let C be the peak capacity (in bits per second), and let T be the length of a time slot. Then the efficiency of the Ethernet is

i.e. the time to send a packet as a fraction of the total time.

(a) Fill the table which gives the efficiency of the Ethernet as function of the number of systems connected to the net (N=100, 200, 300, 500) and average packet size (S=100, 1000, 10000, 100000). For the slot time T use 16 microseconds and set C to 10^8 (100 Mbps).

N	S			
	100	1000	10000	100000
100	0.035367	0.268277	0.785701	0.973449
200	0.035231	0.267496	0.785030	0.973346
300	0.035186	0.267237	0.784806	0.973312

500	0.035151	0.267030	0.784628	0.973284

Now, value of C is 10^8 bit per second, and T is 16^{-6} seconds.
For example,
when N = 100 and S = 100, value of P is 0.369730, W is 1.704679, and E is 0.035367

(b) For a given number of systems (nodes) on the network, is the network more efficient with long packets or short packets? Is this what you would expect?

As an experiment above, the longer packet size, the better efficiency.

For example, in case of N (The number of systems connected to the net) is 100, the value of Efficiency when S = 1000 is bigger than when S = 100, when S = 10000 is bigger than when S = 1000,... It means the bigger packet size S, the better efficiency E.