Question 1. 10 pts.

Answer the following questions based on project 2 and 3.

a) When not taken from a file, which distribution did you use to generate CPU burst lengths in project 2?
It was uniform distribution. Randomly generated values that are distributed uniformly. Hence distribution is: Uniform.

b) What was the role of SE thread in project 2.
it is to simulate the scheduling and execution of the cpu bursts.

c) What is the information sent from a PS thread to a SE thread in project 2?
Information about a cpu burst is sent from a PS thread to the SE thread. It can include the ID of the thread as well.

d) In project 3, in library, could you put a pointer value (a virtual address) directly into the shared memory? Why or why not?
Not directly. Since each process can map the shared segment to a different region (address) in its VM, we need to covert a pointer in the address space of a process to a standard (offset) value. Vice versa.

e) How did you avoid race conditions in project 3?
We should use a semaphore initilized to 1. Before accessing the shared segment, we need to do a wait() op on the semaphore. When we are done, we need to do a signal() op on the semaphore.

Question 2. 10 pts.

We have 5 processes, A, B, C, D, E. The table below shows their arrival and CPU burst times. These processes are scheduled with round robin scheduling with time quantum q = 1 time unit (one time unit). Find the finish time of each process.

	Arrival	CPU Burst
A	0	110
B	40	85
C	100	45
D	130	80
E	210	50

	0	40	100	130	210	260	280	310	360	370
A	40	30	10	20	10					
B		30	10	20	10	5	10			
C			10	20	10	5				
D				20	10	5	10	25	10	
E					10	5	10	25		

Table shows the amount of cpu execution that each process got during each time interval. First time interval start at time 0 and ends at time 40.

Finish times (approximately):
A: 260
B: 310
C: 280
D: 370
E: 360

Question 3. 10 pts.

Assume we have a system where there are 3 resources types: A, B, and C. There exist 10 instances of A, 9 instances of B, and 7 instances of C in the system in total. That means the existing vector is [10, 9, 7]. Consider the following state. Is it safe or unsafe? Prove your answer. *(Note that in this state, the available vector are less than the existing vector).*

Max Demand			
	A	B	C
P1	6	2	2
P2	2	1	1
P3	4	8	1
P4	3	3	0
P5	4	4	8
P6	1	1	3

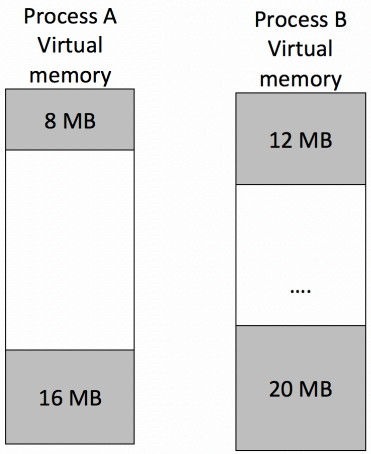
Allocation			
	A	B	C
P1	3	1	2
P2	2	0	0
P3	1	3	1
P4	1	1	0
P5	3	2	1
P6	0	1	1

Need is:	Available is: 0 1 2.
3 1 0	Wit this, need of P2 can be satisfied. Then available becomes: 2 1 2.
0 1 1	Wit this, need of P6 can be satisfied. Then available becomes: 2 2 3.
3 5 0	Wit this, need of P4 can be satisfied. Then available becomes: 3 3 3.
2 2 0	Wit this, need of P1 can be satisfied. Then available becomes: 6 4 5.
1 2 7	There need of P3 and P5 can not be satisfied. Hence this state is
1 0 2	not safe.

Question 4. 15 pts.

Consider a computer that uses 30 bits virtual addresses. The computer has 8 GB of RAM (physical memory). The page size is always 4 KB and a page table entry is 8 bytes long, no matter which page table scheme is used. The virtual memory layouts of 2 processes, A and B, are shown below. In the figure, the unused virtual memory space of a process is shown with white color. Find out the total physical memory space needed to hold the page table information of these two processes for:

a) single-level page table scheme.
b) two-level page table scheme
with address division as [9, 9, 12].
c) inverted page table scheme.



a) (4 pts) 30- 12 = 18 bits in page number. Hence 2^18 pages == 2^18 entries in a page table. That means a page table is
2^18 x 2^3 = 2^21 = 2 MB. We need two tables: hence **4 MB** in total.

We need two such tables. Hence 4 MB RAM required.
b) (7 pts) a second level table can map 2^9 x 2^12 = 2^21 bytes = 2. MB of VM. Hence we need for A: 1 top level + 12 second level tables. For B: 1 top level + 16 second level tables. A table needs 2^9 x 2^3 = 2^12 = 4 KB of RAM.

Hence we need RAM for 13+17 = 30 tables. Hence 30x4KB = **120 KB** of RAM needed.

c) (4 pts) We need one inverted table that has 2^33 / 2^12 = 2^21 entries. Each entry 8 bytes. Then we need
2^21 x 2^3 = 2^24 = **16 MB** of RAM.

Question 5. 15 pts.

Assume we have 4 frames that a process can use. All frames are initially empty. The page reference string of the process is as follows:

3 7 2 1 2 3 5 2 4 3 1 4 7 2 5

The system is using an R bit for each page. R bits are clearing after every 5 references. Assume when a page is newly loaded into memory, its R bit set to 1 (one). For each of the following page replacement algorithms, show the memory state after each page reference. Memory state means which pages are in memory (in 4 frames) and their R bits. For example, after the fourth reference, the memory state is shown below. Also, for each reference mark (with an X sign – meaning page fault) whether the reference causes a page fault or not.

- FIFO algorithm (5 pts)
- LRU algorithm (5 pts)
- Second change (clock) algorithm. (5 pts)

3^1
7^1
2^1
1^1

	3	7	2	1	2	3	5	2	4	3	1	4	7	2	5
FIF	x	x	x	x			x		x	x			x	x	x
	3	3	3	3	3	3	5	5	5	5	5	5	5	2	2
		7	7	7	7	7	7	7	4	4	4	4	4	4	5
			2	2	2	2	2	2	2	3	3	3	3	3	3
				1	1	1	1	1	1	1	1	1	7	7	7

	3	7	2	1	2	3	5	2	4	3	1	4	7	2	5
LRU	x	x	x	x			x		x		x		x	x	x
	3	3	3	3	3	3	3	3	3	3	3	3	3	2	2
		7	7	7	7	7	5	5	2	2	2	2	7	7	7
			2	2	2	2	2	2	5	5	1	1	1	1	5
				1	1	1	1	1	4	4	4	4	4	4	4

	3	7	2	1	2||	3	5	2	4	3||	1	4	7	2	5
SC	x	x	x	x			x		x		x		x		x
	3(1)	3(1)	3(1)	3(1)	3(1)*	3(1)*	3(0)	3(0)	3(0)*	3(1)*	1(1)	1(1)	1(1)	1(1)	1(0)
		7(1)	7(1)	7(1)	7(1)	7(0)	5(1)	5(1)	5(1)	5(1)	5(0)*	5(0)*	7(1)	7(1)	7(0)
			2(1)	2(1)	2(1)	2(0)	2(0)*	2(1)*	2(0)	2(0)	2(0)	2(0)	2(0)*	2(1)*	5(1)
				1(1)	1(1)	1(0)	1(0)	1(0)	4(1)	4(1)	4(0)	4(1)	4(1)	4(1)	4(0)*

Question 6. 20 pts.

Consider a disk that is 128 GB. The block size 4KB. A disk pointer is 4 bytes long, no matter which block allocation method is used. Answer the following questions based on this.

a) If FAT method is used, how many disk blocks are occupied by FAT. Assume a FAT entry is 4 bytes long.

b) If bitvector method is used to keep track of free space, how many blocks will be occupied by bitvector?

c) Assume three-level indexed allocation is used, no matter what the size of a file is. How many index blocks are needed for files A, B, C of sizes 2 MB, 1 GB, 32 GB, respectively.

d) Assume linked allocation scheme is used. A portion of the disk blocks are shown below. Assume a file X starts at block 3219. Assume in a block, pointer is in the beginning. What is the size of the file X at most? If we want to access byte with offset 9000 in file X, which disk block and which byte in that disk block do we need to access?

0	...	1345	.....	2567		3219		5673	5674	...
		2567		NULL		5674		12314	1345	
	...									

- a). (5 pts) 128 GB / 4 KB = 2^37 / 2^12 = 2^25 blocks on disk == entries needed. Size of FAT = 2^25 x 4 = 2^27 bytes = 2^27 / 2^12 = **2^15 blocks.**
- b) (5 pts) 128 GB / 4 KB = 2^37 / 2^12 = 2^25 blocks on disk. We need that many bits. 2^25 bits is 2^22 bytes. 2^22 / 2^12 = 2^10 blocks. Bitvector occupies **2^10 blocks.**
- c) (5 pts) The last level index block (3rd level block) can map: 2^10 x 2^12= 4 MB of contiguous file content. For a 2 MB file, 1 third level block is enough. We also need 1 second level block and 1 top level index block, hence a total of 3 index blocks. For a 1 GB file, we need 2^30 / 2^22 = 2^8 third level index blocks. Hence we need in total 256 + 1 + 1 = 258 index blocks. For a file of size 32 GB: a second level index block can map: 2^10 x 2^10 x 2^12 = 2^32 bytes (4 GB) of contagious file content. 32 GB / 4G = 8. Hence we need 8 second level index blocks. We need 2^35 / 2^22 = 2^13 third level index blocks. We need 1 top level index blocks. In total we need: 2^13 + 8 + 1 = 8192 + 8 + 1 = 8201 index blocks.
- d) (5 pts) The blocks allocated to the file are (in order): 3219, 5674, 1345, 2567. Hence size can be at most 4x4KB = **16KB.**
  4 bytes of a block is pointer, remaining 4092 bytes contain data.
  9000 = 2 x 4092 + 816. Then we need to access logical block 2, which is on disk block 1345. IN that disk block, we need to access 4 + 816 = 820 (offset). Hence we need to access **(diskblock=1345, offset=820).**

Question 7. 10 pts.

Consider the following disk requests (cylinder numbers given). The head is on cylinder 350 initially. The initial direction is towards right (towards larger cylinder numbers). Assume for circular algorithms, the serving occurs while going from left to right (i.e., from smaller to larger cylinder numbers). Find out the total seek distance for the following disk scheduling algorithms:

a) SSTF (5 pts)
b) C-SCAN (iptal)
c) C-LOOK (5 pts)

400 200 340 300 50 450 370

a) 350 ➔ 340 ➔ 370 ➔ 400 ➔ 450 ➔ 300 ➔ 200 ➔ 50 (total **520**)

c) 100 + 400 + 290 = **790**

Question 8. 10 pts.

How many different integer sequences (order is important) may be printed out by the following pseudo-code? Write down all of them. Show your work, otherwise you may not get any points. The semaphores X, Y, Z are initialized to 0.

```
int n;
main() {
        n = fork();
        if (n==0) {
                signal(X)
                print (40);
                wait (Y);
                print (30);
                wait(Z);
                print(45);
                exit (0);
         }
        signal (Y);
        print (60);
        wait (X);
        print (10);
        print(80);
        signal(Z);
}
```

- There are 10 possibilities.
- 45 should be at the end.
- Remaining 5 numbers.
- 5! / 3!2! = 10.
- 40 30 60 10 80 45
- 40 60 30 10 80 45
- 40 60 10 30 80 45
- 40 60 10 80 30 45
- 60 40 30 10 80 45
- 60 40 10 30 80 45
- 60 40 10 80 30 45
- 60 10 40 30 80 45
- 60 10 40 80 30 45
- 60 10 80 40 30 45