COMP 3511 Operating System (Spring 2024)

Final Exam

Date: 21 May 2024 (Tue)
Time: 08:30 – 11:00 (2.5 hours)

Name (Write the name printed on your Student ID card)	
Student ID	
ITSC email	@connect.ust.hk

Exam format and rules:

- It is an **open-book, open-notes exam**.
- The exam paper contains ??? single-side pages. You may use the back of the pages for your rough work, or continue your answers.
- It is an open-book, open-notes exam (Ref: Chapter 1)
- You can bring any useful hard copies, including lecture notes, lab slides, and past exam papers
- An electronic calculator is allowed. Other electronic devices for internet access are not allowed. No laptop computers or tablets
- Other details are already sent to students via exam-related emails

Q1: Multiple Choices (20 questions)	/20
Q2: Deadlock	/15
Q3: Memory Management & Virtual Memory	/30
Q4: File System and Disk Scheduling	/20
Q5: Synchronization	/15
Total	**/100**

Please write down your answers in the boxes below:

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20

[Chapter 6-7: Synchronization]

1.1) If a context switch takes `T` time, what would be an appropriate upper bound (in terms of `T`) for utilizing a *spinlock*, instead of a *mutex lock*, i.e., a *spinlock* is more effective than a *mutex lock*, in which a waiting process can be put to sleep

A) `T`
B) `2T`
C) `3T`
D) `4T`

Answer: B A mutex lock require two context switch times

1.2) In Linux, it usually does not allow a process to attempt to acquire a *semaphore* while holding a *spinlock*, why?

A) A *spinlock* may put a process to sleep, so the process is unable to acquire a *semaphore*
B) A *spinlock* wastes CPU cycles
C) Acquiring a *semaphore* may put a process to sleep, in that case holding *spinlock* for too long
D) *semaphore* and *spinlock* are nearly identical in operations

Answer: C Slide 6.22 Spinlock is useful when locks expected to be held for short times

1.3) Which of the following statements is incorrect for an *adaptive mutex* in Solarix?

A) The combination of *condition variables* and *semaphores* can not replace an *adaptive mutex*
B) A waiting process may spin while waiting for the lock to become available
C) A waiting proess may be put to sleep while waiting for the lock to become available
D) The *adaptive mutex* is only used to protect short segments of code

Answer A Slide 7.11 B-C-D are all correct

[Chapter 8: Deadlock]

1.4) Which of the following statements is true for *deadlock recovery*?

A) The system can choose to abort all deadlocked processes (those processes not included in any safe sequences)
B) The system can choose to abort one process at a time until the deadlock cycle is eliminated

C) The system can choose to preemot some resources from selected processes and give these resources to other processes until the deadlock cycle is broken
D) All of the above
Answer: D Slide 8.40-8.41

1.5) Assume that there are three processes P1, P2 and P3, and 10 instances of the same resource type R in the system. The maximum number of R which may be requested by P1, P2 and P3 are 10, 3 and 6 respectively. The number of resources currently allocated to Pl, P2 and P3 are 4, 1 and 4 respectively. Which of the following statements characterizes the state?
A) There exists one safe sequence, and the system is in safe state
B) There exist no safe sequence, and the system is in unsafe state
C) The system state can not be determined
D) There are multiple safe sequences, and the system is in safe state
Answer: B
The need is [6, 2, 2] and the available resource is 1, which can not satisfy any

1.6) Suppose that there are certain number of instances of the same resource R in the system. Three processes P1, P2, and P3 declares that they require 8, 4 and 6 instances of R to finish their tasks at the maximum, respectively. The system currently has allocated 4, 2 and 3 instances of R to P1, P2, and P3, respectively. What is the minimum number instances of R (total) that the system should provide to avoid deadlock?
A) 9
B) 10
C) 11
D) 12
Answer: C
The need is also [4, 2, 3], the available resource must be at least 2, making the total 11. The safe sequence would be P2, P3 and P1, or P2, P1 and P3

[Chapter 9: Memory Management Strategies]
1.7) Consider a system with a 30-bit logical address and 1 KB page size. Suppose that each page table entry (PTE) occupies 4 bytes and each page table must be contained within a page or frame. How many levels of page translation does this require?
A) one level
B) two level
C) three level
D) four level
Answer: C
The offset uses 10 bits, the page number has 20 bits. Given 4 bytes per PTE, each page table can have up to 256 or 2^8 number of entries, so the address should be divided as [4, 8, 8, 10], three level page table is required.

1.8) Consider a partial segment table of one process:

Segment #	Segment length	Starting address	Permission	Status
0	100	5000	Read-only	In memory

1	200	1500	Read/Write	In memory
2	300	-	Read/Write	Not in memory
3	500	2000	Read-only	In memory

When accessing the two logical addresses, reading at < 1, 150> and writing at <3, 400>, what are the results after address translation?
A) <1650> and operation not permitted
B) <1650> and segment out of range
C) <1650> and <2400>
D) <1650> and segment not in memory
Answer: A <1,150> 150 valid, convert to 1650. <3, 400> valid, convert to <2400> but read-only, this operation not permitted

1.9) Consider a two-level page table with the following address structure.

page number		page offset
p_1 (8 bits)	p_2 (10 bits)	d (12 bits)

What is the page numbers of the logical address 0x 061B 240A (in hexadecimal) in the outer page table (p_1) and the inner page table (p_2) in hexadecimal, respectively?
A) 0x081; 0x101
B) 0x18; 0x1B2
C) 0x081; 0x401
D) 0x201; 0x401
Answer: B
0x 061B 240A = 0b 0000 0110 0001 0111 0010 0100 0000 0110. p_2 = 0b 01 0111 0010 = 0x1B2; p_1 = 0b 0001 1000 = 0x18

1.10) Assume a system uses *two-level paging* scheme and has a TLB hit ratio of 90% (TLB hit directly results in address translation). It requires 20 nanoseconds to access the TLB, and 200 nanoseconds to access main memory. What is the effective memory access time (EAT) in nanoseconds for this system?
A) 215
B) 219
C) 225
D) 260
Answer: D 0.90x(20+200)+0.1x(20+200x3)=260

[Chapter 10: Virtual-Memory Management]
1.11) Consider a system with 32-bit virtual address space and page size of 4 KB. If a process requires 64 MB memory at the maximum and its current *working set size* is 128 KB, what is the minimum number of entries in TLB that should be allocated to this process in order to achieve reasonably good performance
A) 16
B) 32
C) 64
D) 128
Answer: B

Slide 10.55 TLB reach - TLB entries should cover the current working set. 128 KB working set with 4 KB page size, the number of TLB entries should be at least 32.

1.12) Consider the page reference from time instances 0 to 9 for processes P1 and P2:
t: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
P1: 2, 1, 3, 3, 6, 5, 3, 4, 4, 5
P2: 4, 3, 6, 6, 3, 3, 7, 6, 8, 6
Suppose that the *working set window* is 3, what is the *working set* of P1 and P2 at the time 8, respectively?
A) [3, 4] and [6, 7, 8]
B) [3, 4, 5] and [6, 8]
C) [3, 4, 5] and [6, 7, 8]
D) [3, 4] and [6, 8]
Answer: A

1.13) Suppose we have the following page accesses: 1 2 3 4 2 3 4 3 2 1 3 2 and three frames allocated. Using the FIFO replacement algorithm, what will be the final configuration of the three frames following the execution of the given reference string?
A) 4, 1, 2
B) 3, 1, 4
C) 4, 2, 3
D) 3, 4, 2
Answer: A

1.14) Suppose we have the following page accesses: 1 2 3 4 2 3 4 3 2 1 3 2 and three frames allocated. Using the LRU replacement algorithm, what is the number of page faults for the given reference string?
A) 4
B) 5
C) 6
D) 7
Answer: B final configuration is [1, 3, 2]

[Chapter 11: Mass Storage Systems]
1.15) Which of the following statements is true about disk characterization
A) A disk can provide large storage capacity
B) A disk block can be rewritten in place multiple times
C) A disk provides direct access to blocks it contains.
D) All of the above
Answer: D Slide 14.4

1.16) Consider a disk drive with 200 cylinders numbered 0 to 199. The drive is currently serving a request at cylinder 88, and the queue of pending requests are: 154, 87, 18, 199, 76, 121, 5, 185. What is the order that the requests are serviced under SSTF scheduling algorithm?
(A) 121 – 154 – 185 – 199 – 87 – 76 – 18 - 5

(B) 87 - 76 – 121 – 154 - 185 – 199 – 18 - 5
(C) 87 – 76 – 18 – 5 – 121 – 154 – 185 - 199
(D)154 – 87 – 18 – 199 – 76 – 121 – 5 - 185
Answer: B

[Chapter 13-14: File-System]
1.17) What does a *volume control block* do?
A) It can contain information needed by the system to boot an operating system from that partition
B) It is a directory structure used to organize the files
C) It contains many of the file's details, including file permissions, ownership, size, and location of the data blocks
D) It contains information such as the number of blocks in a partition, size of the blocks, and free-block and FCB count and pointers
Answer: D

1.18) A *per-process open-file table* and a *system-wide open-file table* are commonly used to keep track of open files in a system. Sippose that a process A has 10 open files and a process B has 6 open files, out of which three files are shared between the two processes. How many entries are in the per-process table of process A, the per-process table of process B, and the system-wide tables, respectively?
(A) 10, 6, 16
(B) 7, 3, 16
(C) 10, 6, 13
(D) 7, 3, 13
Answer: C

1.19) Consider a 32 GB disk using FAT-like system and each FAT table entry contains 24 bits. If a file occupies 12 entries in a FAT table, what is the size of the file?
A) 6 KB
B) 12 KB
C) 24 KB
D) 48 KB
Answer:
Each FAT table entry contains a block number pointing to one block, with 24 bits as block number, the number of blocks is 2^{24}, thus each block size is $2^{35}/2^{24} = 2^{11}$ bytes. So the file size is $12x2^{11} = 24KB$

[Chapter 17: Protection]
1.20) Which of the following statements is not true on *domain* and its implememtation?
A) A *domain* is considered as a generalization of *protection rings* without a hierarchy
B) An *access lists* correspond directly to user needs, which makes it easier to access the set of access rights for each domain
C) Each elment in an *access matrix* is the set of operations that a process executing in one domain can invoke on an object

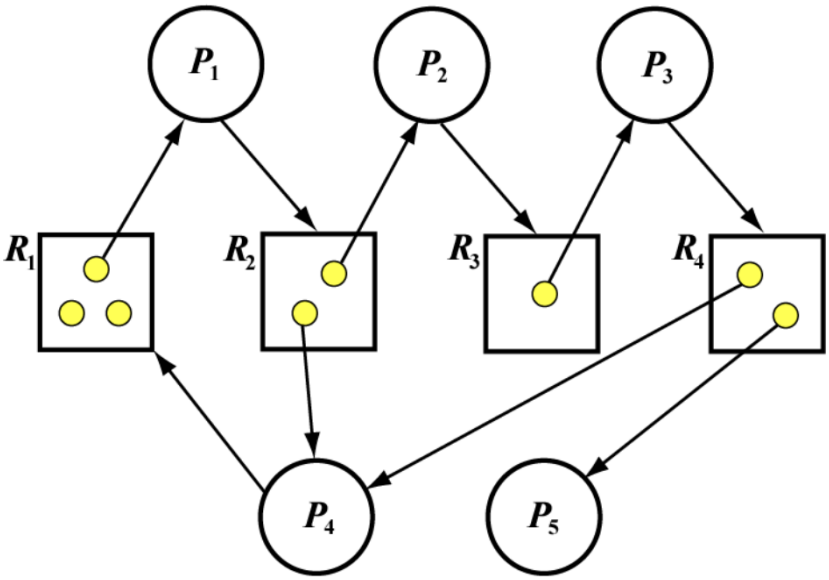
D) A *capability list* for a domain is a list of objects together with the operations allowed on those objects.

Answer: B

Slide 17.25 Determining set of access rights for each domain is difficult - every access to the object must be checked, requiring a search of the access list.

2. [15 points] Deadlock (Banker's algorithm)

2.1 (5 points) Convert the following resource allocation graph to matrix representation (i.e., Allocation, Request and Available).



	Allocation				Request				Available			
	R1	R2	R3	R4	R1	R2	R3	R4	R1	R2	R3	R4
P1	1	0	0	0	0	1	0	0	2	0	0	0
P2	0	1	0	0	0	0	1	0				
P3	0	0	1	0	0	0	0	1				
P4	0	1	0	1	1	0	0	0				
P5	0	0	0	1	0	0	0	0				

2.2 (10 points) Consider the following snapshot of a system with 5 processes and 4 resource types. The amount of each resource for A, B, C and D is 4, 15, 13 and 13 respectively.

	Allocation				**Max**			
	A	B	C	D	A	B	C	D
P1	0	0	1	2	0	0	1	2
P2	1	0	0	0	1	6	5	0
P3	1	3	5	3	2	3	5	6
P4	0	6	3	2	0	6	5	2
P5	0	0	1	3	0	6	5	5

Note: For the banker's algorithm, if there is more than one choice, always pick the process with the smaller process ID. For example, if P1 and P2 can be picked, we always pick P1 because P1 has a smaller process ID comparing with P2.

a) (2 points) What is the content of the Available Matrix denoting the number of resources available?

Answer:
A: 2; B:6; C:3; D:3

b) (4 points) Is the system safe? If yes, write a safe sequence and fill in the table below (i.e., resources available after each process finished); if not, please give the reason.

	Available				
	A	B	C	D	Process
Step 1					
Step 2					
Step 3					
Step 4					
Step 5					
Step 6					

Answer:
Yes, the safe sequence is [P1, P3, P2, P4, P5]

	Available				
	A	B	C	D	Process
Step 1	2	6	3	3	P1
Step 2	2	6	4	5	P3
Step 3	3	9	9	8	P2
Step 4	4	9	9	8	P4
Step 5	4	15	12	10	P5
Step 6	4	15	13	13	

c) (4 points) Can request (0, 3, 2, 0) by P2 be granted immediately? If yes, write the execution order and fill in the table below (i.e., resources available after each process finished); if not, please give the reason.

	Available				
	A	B	C	D	Process
Step 1					
Step 2					
Step 3					
Step 4					
Step 5					
Step 6					

Answer:
Yes, the order is [P1, P3, P2, P4, P5].
R2 (0, 3, 2, 0) <= Need (0, 6, 5, 0)
R2 (0, 3, 2, 0) <= Available (2, 6, 3, 3)

	Available				
	A	B	C	D	Process
Step 1	2	3	1	3	P1
Step 2	2	3	2	5	P3
Step 3	3	6	7	8	P2
Step 4	4	9	9	8	P4
Step 5	4	15	12	10	P5
Step 6	4	15	13	13	

3. [30 points] Memory Management and Virtual Memory

3.1 (8 points) Consider a two-level paging scheme with the following format for virtual addresses:

page number		page offset
6 bits	10 bits	12 bits

We use a two-level page table (in memory) such that the first 6 bits of an address is an index into the first level page table and the next 10 bits are an index into a second level page table. Each page table entry is 4 bytes in size.

(a) (2 points) What are the sizes of a page and logical address space?

Answer:

The page size is 2^{12}=4KB,
the logical address space size is 2^{28} = 256MB

(b) (3 points) How much space is occupied in memory by all the page tables, i.e., both first level and second level page tables?

Answer:

There are 2^6=64 entries in the first level page table, so the first page table occupies 64x4 bytes = 256 bytes or 0.25KB
There are 2^{10}=1,024 entries in each second level page table, so each second page table occupies 1,024x4 bytes = 4KB, there are 64 second level page tables, so the total size of the second page tables is 64x4KB=256KB.

The total memory occupied by all page tables is 256KB+0.25KB = 256.25KB

(c) (3 points) Conider a process with 16MB actual logical address space allocated, at least how many second level page tables does its address translation need?

Answer:

Given a file size 16MB and the page size is 4KB, there are 4K pages.
Each second-level page table contains at most 1K entries, so we need to have at least 4 second-level page tables.

3.2 (13 points) Address Translation

Consider a 36-bit virtual address space, a page size of 4KB and a single-level page table with each page table entry (PTE) size of 4 bytes.

a) (1 points) How many pages are in the virtual address space? Write your answer in term of power of 2 pages.

Since the size of a page 4K bytes (2^12),
the number of pages is 2^36 / 2^12
= 2^24

b) (1 points) Suppose we need at least 6 control bits in PTE, what is the maximum size of our physical address space? Please justify your answer.
A PTE has 32 bits (4 bytes), out of which 6 are used for control purpose. So, the remaining 26 bits are used to address the physical memory. Therefore, the maximum size of addressable physical space is 2^26 * 4 KB = 256 GB.

c) (8 points) Assume the following PTE format.

Physical Page Number	Other	User	Writeable	Valid
24 bits	5 bits	1 bit	1 bit	1 bit

Here, "Valid" means that a translation is valid, "Writeable" means that the page is writeable, "User" means that the page is accessible by the User (rather than only by the Kernel).
Suppose a page table is stored in **big-endian form** in the following page table (i.e., the **most significant byte** is the first byte in memory) and the page-table base register (PTBR) points to 0x6000. Table entries are written in **Hexadecimal**.

Address	+0	+1	+2	+3	+4	+5	+6	+7
0x6000	8C	71	92	E4	F9	BD	5A	1F
0x6008	C7	5D	66	A9	B3	F8	43	26
0x6010	59	F6	81	7A	E2	3D	C4	97
0x6018	2B	98	1E	67	D3	A0	F5	45
0x6020	6D	23	EF	58	47	9A	C1	81
0x6028	A4	0E	5F	76	3B	9D	C1	F8

Please consider the following instruction. If there is no error, provide the translated physical address (in **Hexadecimal**) based on the logical address. Otherwise, provide the specific error(s). Possible errors include **invalid, read-only, kernel-only**.

A) **Load** Logical address: 0x 0 0000 1B36
Answer: 0x F 9BD5 AB36 (2 points)

B) **Load** Logical address: 0x 0 0000 34C1
Answer: Error; Invalid (2 point)

C) **Write** Logical address: 0x 0 0000 7C43
Answer: Error; read-only (2 points)

D) **Write** Logical address 0x 0 0000 5F00
Answer: 0x E 23DC 4F00 (2 points)

d) (3 points) Now suppose the average process size is 8GB, and we want to transform our single level page table into a multi-leveled page table. Assuming that every page table is required to fit into a single page. Consider the following two-level and three-level paging, and compute the total size of the page table respectively.

Virtual Page #1	Virtual Page #2	Offset
12 bits	12 bits	12 bits

Virtual Page #1	Virtual Page #2	Virtual Page #3	Offset
8 bits	8 bits	8 bits	12 bits

Answer:
Two-level: This process accesses 2^33 / 2^12 = 2^21 pages. It needs 2^21 / 2^12 = 2^9 pages for level-two page table, and one level-1 page table. The total size of the page table is then:
//size of the level-1 table //total size of the level-2 table
1 * 2^12 * 4 + 2^9 * 2^12 * 4

= 2^14 + 2^23 bytes
= 8404992B. (~8MB is also accepted, because 2^23 is the dominate term)

Three-level: This process accesses 2^33 / 2^12 = 2^21 pages. It needs 2^21 / 2^8 = 2^13 pages for level-three page table, 2^13 / 2^8 = 2^5 pages for level-two page table, and one level-1 page table. The total size of the page table is then:

1 * 2^8 * 4 + 2^5 * 2^8 * 4 + 2^13 * 2^8 * 4
= 2^10 + 2^15 + 2^23
= 8422400B. (~8MB is also accepted because 2^23 is the dominate term)

3.3 (9 points) Page Replacement Algorithms
Consider the following sequence of page references in term of page numbers:

3, 5, 1, 1, 8, 9, 1, 6, 3, 2, 9, 9, 5, 8, 4, 7

Suppose there are **3 frames** allocated for this process, please illustrate the contents of the frames under different page replacement algorithms.
In the last row (i.e., the bottom row) of each algorithm, please write F if there is a page fault, or leave it empty if there is no page fault. Compute the number of page faults for each algorithm

a) (3 points) FIFO replacement algorithm
Answer:

3	5	1	1	8	9	1	6	3	2	9	9	5	8	4	7
3	3	3		8	8		8	3	3	3		5	5	5	7
	5	5		5	9		9	9	2	2		2	8	8	8
		1		1	1		6	6	6	9		9	9	4	4
F	F	F		F	F		F	F	F	F		F	F	F	F

Total Page fault: 13

b) (3 points) LRU replacement algorithm
Answer:

3	5	1	1	8	9	1	6	3	2	9	9	5	8	4	7
3	3	3		8	8		6	6	6	9		9	9	4	4
	5	5		5	9		9	3	3	3		5	5	5	7
		1		1	1		1	1	2	2		2	8	8	8
F	F	F		F	F		F	F	F	F		F	F	F	F

Total Page fault: 13

c) (3 points) Optimal replacement algorithm.
Note: For this OPT question, if there are more than one victim frames in the current step, always choose a victim frame with the smallest frame number.
Answer:

3	5	1	1	8	9	1	6	3	2	9	9	5	8	4	7
3	3	3		3	3		3		2			5	8	8	8
	5	5		8	9		9		9			9	9	9	9
		1		1	1		6		6			6	6	4	7
F	F	F		F	F		F		F			F	F	F	F

Total Page fault: 11
Marking scheme:

- The first half (i.e., the first 8 frames) is correct (1 point)
- The second half is correct (1 point)
- The last row (page fault row) and the total page fault is correct (1 point)

- For OPT, the exact algorithm should be implemented in PA3 this semester, but some students still make alternative answers. We still accept alternative answers this time.

4. [20 points] File System and Disk Scheduling

4.1 (2 points) File System Implementation

Suppose that you are producing a simplified file system for which the `inode_disk` contains the following data pointers:

```
#define BLOCKSIZE 1024 (bytes – clarified during the exam)
// Assume that this is one BLOCKSIZE in size
struct inode_disk {
    uint32_t direct_pointers[24];
    uint32_t indirect_pointer;
    uint32_t double_indirect_pointer;
}
struct indirect_block { // indirect blocks look like this
    uint32_t block_nums[BLOCKSIZE/sizeof(uint32_t)];
}
```

Please derive the maximum file size (in bytes) that can be supported by this file system (Hint: sizeof(uint32_t)=4 bytes)

Answer:

One block with 1024 bytes can hold 1024/sizeof(uint32_t) = 1024/4 = 256 pointers.

The maximum file size = 1024 bytes/block × [24 + 256 + (256)^2]blocks.

= 67395584 B

= 65816 KB

= 64.27 MB

4.2 (3 points) What are the advantges of using FAT over a linked allocation?

Answer:

Linked allocation performs the worst when accessing a block that is stored at the middle or near the end of a file as it needs to access all of the individual blocks of the file in a sequential manner to find the pointer to the target block starting from the first block. This may involve considerable overhead in moving disk arm and waiting for rotation. This is much easier in FAT by chasing the pointers stored within the FAT, with no or significantly less disk arm movement. In addition, most of the FAT can be cached in memory and therefore the pointers can be determined with just memory accesses instead of having to access the disk blocks.

4.3 (3 points) Consider a disk with 1TB capacity. Suppose the average seek time is 4 milliseconds (4 ms), RPM is 10,000 RPM and transfer rate is 1Gb/s or 125MB/s (Clarified during the exam: 1 G bits per second = 125 M bytes per second). Please derive the effective bandwidth or transfer rate for accessing a 100KB file stored on one cylinder and a 2MB file across two cylinders, respectively. Control overhead time is ignored. (Hint: each cylinder incurs one seek time and one rotation time).

Answer: The average latency = 0.5/(10,000/60) = 3 ms

with control overhead ignored, average access time = 4m + 3ms = 7ms

For 10KB file, the total time is 7ms + 100KB/1Gb/sec = 7ms +10x8x2^{10}/10^9 = 7.8 ms, the effective bandwidth is 100KB/7.8 = 12.82 MB/s or 102 Mb/s.
For 2MB file, the total time is 7x2 ms + 2MB/1Gb/s = 14ms + 2x8x2^{20}/10^9 = 30.78 ms the effective bandwidth is 2MB/30.78 = 65MB/s or 520 Mb/s

Note: It is fine to use 2^{30} bits per second instead of 10^9 bits per second for 1Gb/s.

4.4 (12 points) Disk Scheduling Problem
Suppose that a disk drive has 2000 cylinders, numbered 0 to 1999. The drive is currently serving a request at cylinder 600. The arm direction is moving from smaller to larger cylinder. The queue of pending requests, in FIFO order, is:

1000, 800, 30, 60, 200, 1200, 300, 1800

Starting from the current head position (600), 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? Please also write down the schedule list for each algorithm.
a) (2 points) FCFS
Answer:
The FCFS schedule is 600, 1000, 800, 30, 60, 200, 1200, 300, 1800
The total seek distance is 400+200+770+30+140+1000+900+1500 = 4940.

b) (2 points) SSTF
Answer:
The SSTF schedule is 600, 800, 1000, 1200, 1800, 300, 200, 60, 30.
The total seek distance is 1200+1770 = 2970.

c) (2 points) SCAN
Answer:
The SCAN schedule is 600, 800, 1000, 1200, 1800, (1999), 300, 200, 60, 30.
The total seek distance is 1399+1969= 3368.

d) (2 points) C-SCAN
Answer:
The C-SCAN schedule is 600, 800, 1000, 1200, 1800, (1999), (0), 30, 60, 200, 300.
The total seek distance is 1399+1999+300 = 3698.

e) (2 points) LOOK
Answer:
The LOOK schedule is 600, 800, 1000, 1200, 1800, 300, 200, 60, 30.
The total seek distance is 1200+1770 = 2970.

f) (2 points) C-LOOK
Answer:
The C-LOOK schedule is 600, 800, 1000, 1200, 1800, 30, 60, 200, 300.

The total seek distance is 1200+1770+270=3240.

5. [15 points] Synchronization

5.1 (9 points) Reader-writer Problem
In this problem, there are two semaphore related functions:

```
// sem_wait() decrements (locks) the semaphore pointed to by sem
int sem_wait(sem_t *sem);
// sem_post() increments (unlocks) the semaphore pointed to by sem.
int sem_post(sem_t *sem);
```

a) (4 points) The following is a snippet of code from read-writer problem. This code only shows the reader process.

```
// reader function
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

int readcnt = 0;

sem_t mutex;
sem_t wrt_mutex;
void Reader()
{
    do
    {
        sem_wait(&mutex);
        readcnt++;
        if (readcnt == 1)
            sem_wait(&wrt_mutex);
        sem_post(&mutex);
        printf("Reading\n");
        sem_wait(&mutex);
        readcnt--;
        if (readcnt == 0)
            sem_post(&wrt_mutex);
        sem_post(&mutex);
    } while (1);
}
```

(1) What is the purpose of mutex?
(2) What is the purpose of wrt_mutex?
(3) For the writer process (not shown here), which semaphore should it wait for? When can it write (consider the reader process)?

(1) mutex controls the access of readers to the shared variable count (1 mark)
(2) wrt_mutex controls the access to shared data (critical section) for writers, and the first reader. The last reader leaving the critical section also has to release this lock. (1 mark)
(3) Writers wait on wrt_mutex. (1 mark) It can write before the first reader gains access to the critical section, or after all current readers finish reading. (1 mark)

b) (5 points) The following is a snippet of code from read-writer problem, and the writer has priority over reader. The following properties hold:

- There can be multiple readers and writers. Readers can read simultaneously.
- When writer(s) are waiting to write or writing, the newly coming readers should wait until all writers finish.
- The writers can start writing after the currently reading processes finish.

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

sem_t w_count_mutex, r_count_mutex; // suppose the initial
value is 1
sem_t mutex_1; // the initial value is 1
sem_t mutex_2; // the initial value is 1
int readcnt = 0, writecnt = 0;
void Reader()
{
    do
    {
        BLANK1
        sem_wait(&r_count_mutex);
        readcnt++;
        if (readcnt == 1)
            sem_wait(&mutex_2);
        sem_post(&r_count_mutex);
        BLANK2
        printf("reading\n");
        sem_wait(&r_count_mutex);
        readcnt--;
        if (readcnt == 0)
            sem_post(&mutex_2);
        sem_post(&r_count_mutex);
    } while (1);
}

void Writer()
{
    do
    {
        sem_wait(&w_count_mutex);
        writecnt++;
        if (writecnt == 1)
            BLANK3
        sem_post(&w_count_mutex);
        BLANK4
        printf("writing\n");
        BLANK5
        sem_wait(&w_count_mutex);
        writecnt--;
        if (writecnt == 0)
            BLANK6
        sem_post(&w_count_mutex);
    } while (1);
}
```

(1) Please fill in the blanks using only `sem_wait` and `sem_post` functions, one line of code each.

BLANK1 (0.5 mark)	sem_wait(&mutex_1);
BLANK2 (0.5 mark)	sem_post(&mutex_1);
BLANK3 (0.5 mark)	sem_wait(&mutex_1);
BLANK4 (0.5 mark)	sem_wait(&mutex_2)
BLANK5 (0.5 mark)	sem_post (&mutex_2)
BLANK6 (0.5 mark)	sem_post(&mutex_1);

(2) What is the problem of this solution for readers? Briefly explain. (2 mark)
Ans: The readers may starve (1 mark). If writers keep coming, the writers will keep mutex_1, until there are no writers anymore. The readers cannot read during this period. (1 mark).

5.2 (6 points) Synchronization Coding Question in C

```
// sem_wait() decrements (locks) the semaphore pointed to by sem,
similar to wait() learned in the course.
Usage: int sem_wait(sem_t *sem);
// sem_post() increments (unlocks) the semaphore pointed to by sem,
similar to signal() learned in the course.
Usage: int sem_post(sem_t *sem);
/* sem_init() initializes the unnamed semaphore at the address
pointed to by sem.  The value argument specifies the initial
value for the semaphore.
In this question, we should always set pshared value as 0 when
invoking sem_init. There is a special usage for the pshared
parameter, but it is out of the scope of this question.
In other words, when invoking sem_init, we set 0 to pshared (2nd
parameter), and put appropriate values to sem (1st parameter) and
value (3rd parameter) */
int sem_init(sem_t *sem, int pshared, unsigned int value);
```

Assume function `ideal_rand()` is an ideal random number generator, and the random numbers it generates are uniformly distributed between 0 and 1.
Now, we know that the probability of the program outputting ACB is 0.0384, and the probability of it outputting ACA is 0.0096.

Based on your understanding, write code to complete the following program. You should only use the functions (i.e., sem_wait, sem_post, sem_init) listed above to fill in the missing blanks

```
// example.c
// Compile and link the program: gcc example.c -pthread -lrt
#include <stdio.h>
```

```
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t S0, S1; // 3 semaphore data structures
void *p0_thread(void *arg) {
    while (1) {
        sem_wait(&S0); // wait
        sleep(1);
        float rand = ideal_rand();
        if (rand < BLANK1) {
            printf("A"); fflush(stdout);
        }
         else {
             printf("B"); fflush(stdout);
         }
         BLANK2;
    }
}
void *p1_thread(void *arg) {
    while (1) {
        BLANK3;
        sleep(1);
        float rand = ideal_rand();
        if (rand < BLANK4) {
              printf("C"); fflush(stdout);
              sem_post(&S0);
        }
        else {
              break;
        }
    }
}

int main() {
    sem_init(&S0, 0, 1);
    sem_init(&S1, 0, 0);
    pthread_t t0, t1; // create 2 threads to run
    pthread_create(&t0, NULL, p0_thread, NULL);
    pthread_create(&t1, NULL, p1_thread, NULL);
    pthread_join(t0, NULL);
    pthread_join(t1, NULL);
    sem_destroy(&S0);  // clean up the semaphores
    sem_destroy(&S1);
    return 0;
}
```

BLANK1 (2 mark)	0.2
BLANK2 (1 mark)	sem_post(&S1)
BLANK3 (1 mark)	sem_wait(&S1)
BLANK4 (2 mark)	0.6 (or 0.4)

Further explanation of BLANK1 and BLANK4:
Assume the values of BLANK1 and BLANK4 are k and m respectively
The probability to print ACB is:

```
k * m * (1-k) * (1-m) = 0.0384
```

The probability to print ACA is:

```
k * m * k * (1 - m) = 0.0096
```

Solving these 2 equations, we have k = 0.2 and m = 0.6 (or 0.4)