COMP 3511 Operating System (Fall 2023)

Final Exam

Date: December 18, 2023
Time: 4:30 pm – 7:00 pm (2.5 hours)

Name (Write the name printed on your Student ID card)	Solution (v17 – after grading, fixed problems found during the exam)
Student ID	
ITSC email	@connect.ust.hk
Your signature: _____ fully understands the **HKUST Academic Honor Code** and confirm to follow it during the exam	

Exam format and rules:

- It is an **open-book, open-notes exam**
- The exam paper contains XX single-side pages.
- Other details are already sent to students via exam-related announcements

Q1: Multiple Choices				MCQ1-20 /20
Q2: Deadlock	Q2.1: Banker's algorithm /8		Q2.2 Dining Philosophers /7	
Q3: Memory Management & Virtual Memory	Q3.1 Working Set /5	Q3.2 Paging /6	Q3.3 Address Translation /10	Q3.4 Page Replacement /12
Q4: Disk and File System				/17
Q5: Sychronization	Q5.1 Synchronization Coding /6		Q5.2 Synchronization Barrier /9	
Total				**/100**

Please write down your answers in the boxes below:

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
D	B	A/C	C	D	C	A	B	(Cancelled)	D
Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
A	D	B	A	A	D	B	C	A	C

[Synchronization]

1.1) Which of the following statements is true on an *adaptive mutex*?

A) It is used protect data from *short-code* critical section segments on multiprocessor systems

B) It starts as a standard *spinlock* and it spins if the lock is held by a process running on another CPU

C) It blocks and sleeps (i.e., it does not spin) if the lock is held by *non-run-state* process (i.e., a process not currently running)

D) All of the mentioned.

Answer: D

1.2) Which of the following statements is **not** true for *first readers-writers problem*?

A) It ensures that no readers wait unless a writer has gained access to the shared object

B) All readers wait on semaphore *rw_mutex* if a writer is accessing the shared object

C) The semaphore *mutex* is used to ensure mutual exclusion among readers when the variable *read_count* is updated

D) Readers can enter the critical section directly if there are reader(s) accessing the shared object

Answer: B

[Deadlock]

1.3) Which of the following strategies is **not** feasible in *deadlock prevention*?

A) do not enforce mutual exclusive access to resources

B) To ensure when a process requests a resource, it does not hold any other resources

C) To allow preemption of resources hold by a process itself

D) To enforce an ordered request of all resources

Answer: A (C is ok if arguing that resources like lock and semaphores not preempted)

1.4) There are 4 different type of **single** instance resources R1, R2, R3 and R4 in a system shared by 4 processes P1, P2, P3 and P4. The resource requirements of each process are: P1 requests for R1 and R3, P2 requests for R3 and R4, P3 requests for R2 and R4, and P4 requests for R2 and R3. If the system is **deadlocked**, the number of processes in the deadlock state is at least ____.

A) 1

B) 2
C) 3
D) 4
Answer: C

1.5**)** Suppose that there are 10 resources of the same type available to three processes. At a time instant, the following table indicates the maximum number of resources declared by each process and current resource allocation. Which of the following process sequence satisfies the safety criteria?

Process	Maximum Needs	Currently Owned
P_0	10	4
P_1	3	0
P_2	7	2

A. P1, P2, P0
B. P2, P0, P1
C. P0, P2, P1
D. It is not safe
Answer: D

At time 0, work vector is [4] and need_matrix =
[6
3
5]

[Memory Management Strategies]

1.6) The logical address space has a size of 16MB size, which consists of 1024 pages. The physical memory is composed of 4096 frames. How many bits are required in the logical address and the physical address, respectively?
A) 20 bits and 22 bits respectively.
B) 22 bits and 24 bits respectively.
C) 24 bits and 26 bits respectively.
D) 26 bits and 28 bits respectively.
Answer: C

The size of logic address space is 16MB, thus the number of bits in the logical address is 24 (2^{24} bytes = 16MB). Since it contains 1024 pages, so the page or frame size is $2^{24}/1024 = 2^{14}$, or 16KB. The size of physical memory is = 4096 * 16KB = 2^{26} bytes. So the number of bits in the physical address is 26.

1.7) In a system with 32-bit logical address, given a logical address 0xFABE (in hexadecimal) with a page size of 2KB, what is the page offset?
A) 01010111110
B) 10111110
C) 101010111110
D) 11111010101
Answer: A

0XFABE in binary 1111 1**010 1011 1110**, 2KB page size, the offset has 11 bits

1.8) Assume a system uses **one-level paging** and has a TLB hit ratio of 90%. It requires 10 nanoseconds to access the TLB, and 150 nanoseconds to access main memory. What is the effective memory access time (EAT) in nanoseconds for this system?
A) 170
B) 175
C) 180
D) 185
Answer: B
Miss time = (10 + 150 + 150) = 310 and Hit time = 150 + 10 = 160
EAT = 0.9*160 + 0.1*310 = 144 + 31 = 175

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

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

What is the page numbers of the logical address 0x 2207 3300 (in hexadecimal) in the outer page table (p_1) and the inner page table (p_2), respectively?

A) 0x0AA; 0x1CC
B) 0x0AA; 0x073
C) 0x022; 0x1CC
D) 0x022; 0x073
Answer: Cancelled
(To: Grader TA, PLEASE Fix the explanation, even our 4 options are WRONG)

0x 2207 3300 = 0b 0010 0010 0000 0111 0011 0011 0000 0000. p_1 = 0b00 1000 1000 = 0x088 (we made a mistake here, so there is NO correct answer for this question)
; p_2 = 0b 00 0111 0011 = 0x 073

1.10) Consider a segment table of one process:

Segment #	Segment length	Starting address	Permission	Status
0	3000	2000	Read-only	In memory
1	2000	-	Read/Write	Not in memory
2	1000	6000	Read/Write	In memory

Given the logical address , the physical address is:
A) Segment missing
B) Return address 4000
C) Invalid permission
D) Segment out of range
Answer: D

[Virtual-Memory Management]
1.11) Which of the following results is **not** caused by a large page size?

A) large page table
B) large internal fragmentation
C) large TLB reach
D) large I/O overhead
Answer: A

1.12) Under page replacement policy, what is the main advantage of local replacement over global replacement?
A) It offers better memory utilization, as the set of pages are exclusively used for one process
B) It provides better overall system throughput
C) It does not suffer from Belady's anomaly
D) It results in more consistent performance for each process, as the set of pages for a process is not affected by the paging behavior of other processes
Answer: D

1.13) Suppose we have the following page accesses: 1 2 3 3 1 4 4 2 5 2 4 2 6 4 2 6 and a process is allocated with three frames. How many page faults does the FIFO page replacement algorithm produce?
A) 10
B) 8
C) 7
D) 5
Answer: B
Seque 1 2 3 3 1 4 4 2 5 2 4 2 6 4 2 6
Frame 1 1 1 1 1 4 4 4 4 4 4 4 6 6 6 6
Frame 0 2 2 2 2 2 2 2 5 5 5 5 5 4 4 4
Frame 0 0 3 3 3 3 3 3 2 2 2 2 2 2 2 2
Count 1 2 3 3 3 4 4 4 5 6 6 6 7 8 8 8

[Mass Storage Systems]
1.14) Consider a disk average seek time is 5 ms, RPM is 7,200 RPM, transfer rate is 1Gbps and disk block size is 4KB. The controller overhead is ignored. Suppose that a file has a size of 256KB with a *linked allocation*. What is the effective bandwith or effective transfer rate? (**Hint**: With the *linked allocation*, each block of the file is placed at a random location). Note: 1Gbps is 10^9 bps and KB is 2^{10} bytes.
A) 3.56 Mbps
B) 4.25 Mbps
C) 33.33 Mbps
D) 45.66 Mbps
Answer: A
With random placement of blocks, we only need to consider one block
RPM is 7200, RPS is 120, the average latency is half of 1/120 = 4.17 ms
Average I/O time = average seek time + average latency+ (block size/transfer rate) = 5 + 4.17 + 4KB/1Gbps = 5 + 4.17 + 0.03 = 9.2 ms
The effective bandwidth is $4KB/9.2x10^{-3}$ = 3.56 Mbps

1.15) Assume the disk arm is at cylinder 105 and is moving towards the cylinder with larger indices (left to right). Consider a queue of pending request: 6, 174, 108, 81, 23, 43, 190, 159. What is the order that the requests are serviced under LOOK scheduling algorithm?

A) 108, 159, 174, 190, 81, 43, 23, 6
B) 6, 23, 43, 81, 108, 159, 174, 190
C) 108, 159, 174, 190, 6, 23, 43, 81
D) 6, 174, 108, 81, 23, 43, 190, 159

Answer: A

[File-System Interface and Implementation]

1.16) Consider a file system with 8KB block size and 4 bytes disk address. The inode contains 12 direct block pointers, one single indirect andone double indirect and one triple indirect. Please calculate the disk capacity and the approximate maximum file size.

A) 16TB, 8TB
B) 16TB, 16TB
C) 32TB, 32TB
D) 32TB, 64TB

Answer: D

Disk capacity, disk address x block size = 2^{32}x8KB = 2^{32}x2^{13}B=2^{45}B=32TB
The number of pointers in each index block is 4KB/4B=2^{11}. The maximum file size (approximate) is 2^{11}x2^{11}x2^{11}x8KB=2^{46}B=64TB

1.17) Suppose that the operating system uses two internal tables to keep track of open files. Process A has seven files open and process B has five files open. Two 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) 7, 5, 12
B) 7, 5, 10
C) 5, 3, 10
D) 10, 10, 10

Answer: B

1.18) Consider a file system stored on a disk with block size of 2048 bytes. Suppose the disk address (block number) uses 4 byte. Please compute the total number of blocks required to allocate a file of size 107,7900 (Note: we missed the last 0, but it was clarified during the exam) bytes under (a) contiguous and (b) indexed allocation.

A) 527 and 528

B) 526 and 528

C) 527 and 529

D) 528 and 529

Answer: C

(a) For contiguous allocation, we need to calculate the number of contiguous blocks required to store the file.

Number of blocks required = ceil(file size / block size) = ceil(1,077,900 / 2048) = 527

Therefore, the file will require 527 contiguous blocks.

(b) For indexed allocation, we need to calculate the number of blocks required to store the file's index and the number of blocks required to store the file's actual data.

Assuming each index block can hold 512 addresses, we need ceil(file size / block size) index blocks to store all the addresses required to access the file's data.

Number of index blocks required = ceil(527 / 512) = 2

Therefore, we need 2 index blocks to store all the addresses required to access the file's data. The remaining space in the last index block will not be fully utilized, as it will only contain addresses for a portion of the last block of the file.

Number of data blocks required = 527 + 2 = 529

Therefore, the file will require 529 blocks in total for indexed allocation.

[Protection]

1.19) Which of the following statements on *access matrix* is incorrect?
A) The *owner* of an object can modify any access right in any entry in a row
B) The *transfer* op enables switching from one domain to another (i.e., a different row)
C) The *copy* op (denoted by "*") only applies within a column, i.e., for an object
D) The *control* op allows to modify another domain objects (i.e., access rights in another row) while in one domain
Answer: A

1.20) Which of the following statements is not true on a *capability list*?
A) A *capability list* for a domain presents a list of objects together with the operations allowed on those objects
B) It is useful for localizing information for a given process
C) It makes easier for revocation
D) A process attempting access on an object must present a capability for that access
Answer: C

2. [15 points] Deadlock

2.1 (8 points) Banker's algorithm
Consider the following snapshot of a system with 5 processes and 5 resource types:

	Allocation					Max					Available				
	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E
P0	3	0	0	2	3	5	0	1	4	7	2	3	1	4	3
P1	1	0	3	3	1	2	3	3	7	1					
P2	1	0	0	0	1	2	5	1	2	2					
P3	2	3	3	0	0	2	6	5	8	0					
P4	0	1	0	0	0	9	3	5	7	3					

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 P0 and P1 can be picked, we always pick P0 because P0 has a smaller process ID comparing with P1.
Banker's algorithm may have multiple possible solutions. To standardize the answers, for each sub-question, you can write some intermediate steps. If your final answer is wrong, TAs will consider giving you some partial credits based on your steps (i.e., you may know the steps, but only make some careless mistakes)

a) (2 points) Is the system safe? If yes, write a safe sequence, if not, write "No safe sequence".
Answer: The system is safe. The safe sequence is [P1, P0, P3, P2, P4]

b) (2 points) Can request (1, 0, 0, 1, 1) by P0 be granted immediately? Why?
Answer: The request [1,0,0,1,1] CANNOT be immediately granted to process 0.
In an early step of banker's algorithm, the work vector [1,3,1,3,2] which cannot satisfy any need of the processes, which leads to an unsafe state.

c) (2 points) Can request (1, 0, 1, 0, 1) by P0 be granted immediately? Why?
Answer: Yes, the request [1,0,1,0,1] CAN be immediately granted to process 0
A possible sequence is [1,0,3,2,4]

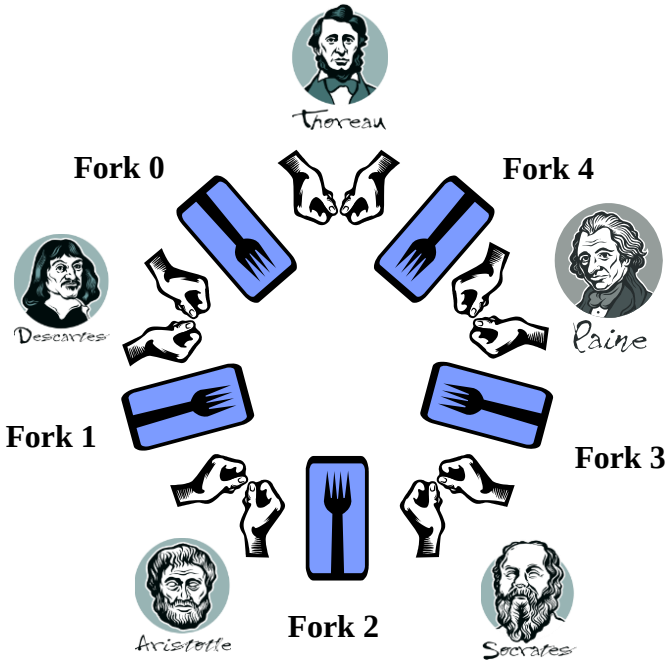
d) (2 points) Can request (1, 0, 1, 0, 1) by P4 be granted immediately? Why?
Answer: No. In the step2, when the work vector becomes [2,3,3,7,3], which cannot satisfy any need of the processes, which leads to an unsafe state.

Note: For a-d, students cannot simply write Yes/No/Yes/No without reasonable answers and explanations.

Note: Many students wrote lots of explanations in the tiny space, but the grading scheme is very simple for this question.
In general, if students can answer Yes/No/Yes/No (or similar things from a-d) with reaonsable answers and explanations, they will get full marks.
Suitable partial credits are granted to a-d if the answer is wrong, but with some correct steps.
0 marks are given if all a-d are blank (i.e., empty)

2.2 (7 points) Dining Philosophers

Five philosophers dine together at a table illustrayed below. Each philosopher has his/her own plate at the table. There is a **fork** between each plate. The dish served has to be eaten with two forks. Each philosopher alternately thinks and eats. Moreover, a philosopher can only eat his dish when he has *both a left and right fork*. Thus two forks will only be available when his two nearest neighbors are thinking, not eating. After an individual philosopher finishes eating, he will put down both forks. You are required to design a coordination algorithm such that no philosopher will starve; i.e., each can forever continue to alternate between eating and thinking, assuming that no philosopher knows when others may want to think or eat.



a) (2 points) Consider the following solution for the philosopher `i`, where `take_fork(i)` and `put_fork(i)` are implemented as **atomic functions** to take up and put down fork `i`, respectively. Unfortunately, this solution may result in a deadlock. Please illustrate this deadlock problem with a concrete example.

```
#define N 5
void philosopher(int i) {
   while (TRUE) {
      think();
      take_fork(i); /* take the left fork */
      take_fork((i+1)%N); /* take the right fork */
      eat(); /* yummy */
      put_fork(i); /* put down the left fork */
      put_fork((i+1)%N); /* put down the right fork */
   }
}
```

Answer: The deadlock occurs when each takes the left or right fork i

b) (5 points) Resolve the problem in the above solution using the *deadlock prevention* technique by invalidating circular waiting.

Answer: Number the forks and then impose an acquisition order. Between the left and right forks, a philosopher should first take the one with a smaller index, followed by the one with a larger index.

```
#define N 5
void philosopher(int i) {
   while (TRUE) {
      think();
      take_fork(min(i,(i+1)%N));
      take_fork(max(i,(i+1)%N));
      eat(); /* yummy */
      put_fork(min(i,(i+1)%N));
      put_fork(max(i,(i+1)%N));
   }
}
```

3. [

3.1 (5 points) Consider there are three processes with the following page reference strings from t0 to t9:

t: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

P1: 5, 7, 6, 5, 7, 6, 4, 3, 1, 2

P2: 6, 5, 7, 6, 1, 2, 4, 3, 4, 3

P3: 2, 5, 7, 6, 5, 7, 6, 4, 3, 5

Given that working set window is 4:

a) (3 points) What is the working set at time 9 for each of the three processes?

b) (2 points) What is the size of locality (demand frames) at time 9 for these three processes?

Answer:

working set (Process 1) = {4, 3, 1, 2}

working set (Process 2) = {4, 3}

working set (Process 3) = {6, 4, 3, 5}

Total Demand Frames = size of ({4, 3, 1, 2}, {4, 3}, {6, 4, 3, 5}) = 6

3.2 (6 points) Consider a computer system with 16 bit virtual and physical addresses. Address translation is implemented by a two-level scheme combining segmentation and paging. The page size is 256 bytes.

The virtual address format is shown as follows:

Segment Number (2 bits)	**Page Number (6 bits)**	**Offset (8 bits)**

Segmentation table is as follows, where the Base Address specifies the address of the page table associated with the segment, and Limit specifies the size of the segment:

Segment ID	**Base Address**	**Limit**
0x0	0x2000	0x2000
0x1	0x8000	0x1000
0x2	0x0000	0x1000

0x3	0xD000	0x0200

Four page tables are used, each associated with a segment as follows:

Page table with start address 0x2000:

Frame Number
0x10
0x11

Page table with start address 0x8000:

Frame Number
0x31
0x40
0x41

Page table with start address 0x0000:

Frame Number
0xB1
0xB2

Page table with start address 0xD000:

Frame Number
0x91
0x92

a) (3 points) What is the physical address corresponding to virtual address 0x8063?
0x8063 = 1000 0000 0110 0011
Segment #2 => page table address 0x0000
Page # 000000 => physical page # 0xB1
Physical address is then 0xB163

b) (3 points) What is the virtual address corresponding to physical address 0x9247?
Offset = 0x47
Physical page # 0x92 => 2nd PTE in page table w/ start address 0xD000
=> Page # 000001 and segment # 0x3 = 11
Virtual address is 0xC147 (1100 0001 0100 0111)

3.3 (10 points) Address Translation
Suppose we have 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. Assume that the PTEs are in the following format:

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

Here, "Valid" means whether a translation is valid (1 represents valid, and 0 represents invalid. The same goes for the other two bits). "Writeable" means whether the page is writeable. "User" means whether the page is accessible by the User (rather than only by the Kernel).

a) (2 points) How many bits should be used in the virtual address to represent the page offset? How many pages are in the virtual address space?
Answer: Offset: 12 bits given a page size of 4KB. (1 point) Page number: 2^24 (1 point)

b) (8 points) Assume the PTEs are stored in **big-endian form** in the following page table (i.e. the **Most Significant Byte** is the first byte in memory). Table entries are written in **Hexadecimal**.

Address	+0	+1	+2	+3	+4	+5	+6	+7
0x1000	3A	D8	7F	92	A5	0B	F1	C6
0x1008	4D	88	F5	6E	D9	23	7A	E3
0x1010	1C	B8	56	F0	8D	34	A1	79
0x1018	4E	CD	60	2A	9F	E7	51	3D
0x1020	62	AF	C2	04	9B	87	1F	E8
0x1028	73	DF	28	B5	4C	A9	15	8F
0x1030	52	3E	C7	6A	F2	0D	90	27
0x1038	B2	5F	D1	6C	A3	19	F6	48
0x1040	CE	61	DE	35	0A	97	2E	B1
0x1048	5C	E0	7D	82	1A	65	F9	74

Using the information above, translate each of the following virtual addresses to physical addresses. Assume that the page table pointer is at 0x1000. Write your answer in **Hexadecimal** in the blanks provided. If you encounter an error (e.g., can not get the answer based on the provided information), write ERROR.

1) Logical address: 0x 0 0000 7E42
Physical address: _____
Answer: 0x 9 FE75 1E42 (2 points)

2) Logical address: 0x 0 0000 A3EE
Physical address: _____
Answer: 0x 7 3DF2 83EE (2 points)

3) Logical address: 0x 0 0001 A2B8
Physical address: _____
Answer: ERROR (2 points)

4) Logical address: 0x 0 0001 1453
Physical address: _____
Answer: 0x 0 A972 E453 (2 points)

One point will be given if the answer is partically with the last 3 elements correct in hex, for example, 0x xxxxx E42.

3.4 (12 points) Consider the following page reference string:

1, 2, 1, 3, 3, 4, 4, 5, 1, 2, 5, 2, 3, 1, 4, 3,2

Assuming demand paging with **3** frames. Please illustrate each step that the following replacement algorithms work for this reference string and compute the **page faults** in each algorithm.

a) FIFO replacement
b) LRU replacement
c) Optimal replacement
d) Clock algorithm (suppose that all reference bits are 0 initially)

Answer:

FIFO replacement

Frame	1	2	1	3	3	4	4	5	1	2	5	2	3	1	4	3	2
F1	1	1	1	1	1	4	4	4	4	2	2	2	2	2	2	2	2
F2		2	2	2	2	2	2	5	5	5	5	5	3	3	3	3	3
F3				3	3	3	3	3	1	1	1	1	1	1	4	4	4
	PF	PF		PF		PF		PF	PF	PF			PF		PF		

Page faults: 9

LRU replacement

Frame	1	2	1	3	3	4	4	5	1	2	5	2	3	1	4	3	2
F1	1	1	1	1	1	1	1	5	5	5	5	5	5	1	1	1	2
F2		2	2	2	2	4	4	4	4	2	2	2	2	2	4	4	4
F3				3	3	3	3	3	1	1	1	1	3	3	3	3	3
	PF	PF		PF		PF		PF	PF	PF			PF	PF	PF		PF

Page faults:11

Optimal replacement

Frame	1	2	1	3	3	4	4	5	1	2	5	2	3	1	4	3	2
F1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4	4	4
F2		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
F3				3	3	4	4	5	5	5	5	5	3	3	3	3	3
	PF	PF		PF		PF		PF					PF		PF		

Page faults: 7

Clock algorithm

Frame	1	2	1	3	3	4	4	5	1	2	5	2	3	1	4	3	2
F1	1(0)	1(0)	1(1)	1(1)	1(1)	1(0)	1(0)	5(0)	1(0)	1(0)	1(0)	1(0)	3(0)	3(0)	3(0)	3(1)	3(0)
F2		2(0)	2(0)	2(0)	2(0)	4(0)	4(1)	4(1)	4(0)	4(0)	5(0)	5(0)	5(0)	1(0)	1(0)	1(0)	2(0)
F3				3(0)	3(1)	3(1)	3(1)	3(1)	3(0)	2(0)	2(0)	2(1)	2(1)	2(0)	4(0)	4(0)	4(0)
	PF	PF		PF		PF		PF	PF	PF	PF		PF	PF	PF		PF

Page faults: 12

4. [17 points] File System and Disk Scheduling

4.1 (5 points) Consider file **`open()`** operation., please illustrate the steps, in particular the on-disk and in-memory file system data structures involved when (a) the file is open for the first time, and (b) the file is used by another process.

Answer: (a) It accesses directory structure (on-disk) to locate the file, creates an entry in the system-wide open-file table and copies the FCB (on-disk) into the table. It then creates an entry in the per-process open-file table pointing to the corresponding entry in the system-wide open-file table (3 points). (b) It creates an entry in per-file open-file table and points to the corresponding entry in the system-wide open-file table (which contains the FCB of this file) (2 points).

4.2 (12 points) Suppose that a disk drive has 1000 cylinders, numbered 0 to 999. The drive is currently serving a request at cylinder 255. The queue of pending requests, in FIFO order, is:

255 446 23 101 631 335 54 105 516 481 75 63 996

(Note: The first 255 causes some confusions to our students. As it is HARD to clarify, we decide to accept more answers. The main problem should be on the first few rows of the table. The total seek time won't be affected.

Starting from the current head position (~~253~~ 255 Note: We clarified this during the exam) , 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? The disk arm is moving from right to left (999 →0). Note that for C-SCAN and C-LOOK, we assume the serving direction is "**From right to left**".

a) FCFS
b) SSTF
c) SCAN
d) LOOK
e) C-SCAN
f) C-LOOK

Answer: (2 points per algorithm)

Schedule	FCFS	SSTF	SCAN	LOOK	C-SCAN	C-LOOK
Starting position	255	255	255	255	255	255
	446	335	105	105	105	105
	23	446	101	101	101	101
	101	481	75	75	75	75
	631	516	63	63	63	63
	335	631	54	54	54	54
	54	996	23	23	23	23
	105	105	0	335	0	996
	516	101	335	446	999	631

	481	75	446	481	996	516
	75	63	481	516	631	481
	63	54	516	631	516	446
	996	23	631	996	481	335
			996		446	
					335	
Total Seek	3647	1714	1251	1205	1918	1866

5. [15 points] Sychronization

5.1 (6 points) Synchronization Coding Question in C
The following few C POSIX APIs are not covered in lectures and labs, but you should learn its underlying concepts in our related lectures.

```
// 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);
/* 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);
```

After completing the missing programming statements below, the program should print P1, P0, P1, …. (Note: on separate lines) indefinitely
Notes:

- P1 needs to print first. The alternating sequence cannot start with P0
- The pattern needs to be repeated indefinitely. For example, we cannot simply print P1, P0, and the program terminates/crashes
- Pthread library is covered before the midterm exam. You should know what is pthread library. In this question, we DON'T require students to write any answers related to pthread library

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; // 2 semaphore data structures
void *p0_thread(void *arg) {
    while (1) {
        BLANK1; // wait
        sleep(1);
        printf("P0\n"); fflush(stdout); // critical session
        BLANK2; // signal
    }
}
void *p1_thread(void *arg) {
    while (1) {
        BLANK3 // wait
        sleep(1);
        printf("P1\n"); fflush(stdout); // critical session
        BLANK4 // signal
    }
}
int main() {
    sem_init(&S0, 0, BLANK5);
    sem_init(&S1, 0, BLANK6);
    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;
}
```

Answer:

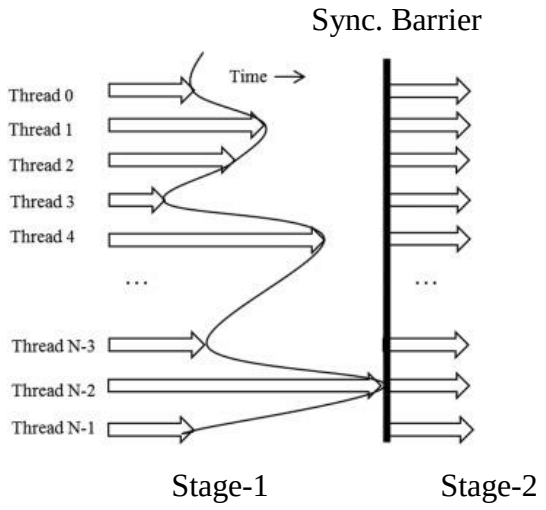
BLANK1 (1 mark)	sem_wait(&S0)
BLANK2 (1 mark)	sem_post(&S1)
BLANK3 (1 mark)	sem_wait(&S1)
BLANK4 (1 mark)	sem_post(&S0)
BLANK5 (1 mark)	0
BLANK6 (1 mark)	1

Note1: It is a runnable program. wait/signal does not work
Note2: Missed &: Overall -1 mark for this mistake
Note3: BLANK1-4 has alternative answers if S0 swapped with S1. The corresponding BLANK5 and BLANK6 values may be swapped. Alternative answers:

```
BLANK1: sem_wait(&S1); // wait
BLANK2: sem_post(&S0); // signal
BLANK3: sem_wait(&S0); // wait
BLANK4: sem_post(&S1); // signal
BLANK5: 1
BLANK6: 0
```

5.2 (9 points) Synchronization Barrier

Consider a multi-tasking program that creates N threads. Each thread performs computations in two stages. Computations in the two stages contain no critical section. Between the two stages there is a *synchronization barrier*, that is, no thread can proceed to stage-2 untill all threads complete stage-1. The following figure illustrates this barrier between two stages. Since Thread N-2 is the slowest in stage-1, the other threads that finish earlier cannot proceed to stage-2 but have to wait in front of the barrier.



You will use the following variables in developing the solution. (All the code snippets in this question are pseudo-code.)

```
N = the number of threads
count = 0; /* # of threads waiting at the barrier */
mutex = Semaphore(1); /* a semaphore w/ init value 1 */
barrier = Semaphore(0); /* a semaphore w/ init value 0*/
```

a) (3 points) What is the problem in the following solution?

```
/* The following code applies to each thread i */
stage-1(); /* do computation in stage-1 */

mutex.wait();
count++; /* the current thread arrives at the barrier */
mutex.signal();

if (count == N) barrier.signal();
barrier.wait();

stage-2(); /* proceed to stage-2 */
```

Answer: The problem of this solution is that `barrier.signal()` will be called indefinite times, between 1 and N. So there is no guarantee that all threads will proceed to stage-2.

b) (3 points) Is the problem resolved in the following solution, please explain why.

```
/* The following code applies to each thread i */
stage-1(); /* do computation in stage-1 */

mutex.wait();
count++; /* the current thread arrives at the barrier */
mutex.signal();

if (count == N) barrier.signal();
barrier.wait();
barrier.signal();

stage-2(); /* proceed to stage-2 */
```

Answer: Yes, this solution works. The semaphore barrier will be signaled only when N threads have completed stage-1, and it will be signaled at least N times. As a result, all threads can pass the barrier and proceed to stage-2.

c) (3 points) What are the possible values of the barrier after all the threads enter stage-2 in the above solution? All semaphores used are counting semaphores with non-negative values.

Answer: 1<= barrier <= N

The minimum value of the barrier is 1. This happens when the straggler (i.e., the slowest thread) is the only thread that sees count == N in the if statement and signals the barrier. At this time, the other threads have already blocked themselves and are

waiting on the barrier. The straggler's signal will unblock one waiting thread, who will immediately signal the barrier to unblock another. This leads to cascading unblock that eventually drives all threads through the barrier. In total, the barrier has been signaled N+1 times (the straggler signaled twice). Given that barrier.wait() is called N times, the final value of the barrier is 1.

The maximum value of the barrier is N. This happens when all threads load count == N in the if statement, and all of them do barrier signaling. In this case, the barrier is signaled 2N times, and waited N times. So the final value is N.

(Grading: correct values -> 1 point, partially correct interval -> 2 points, completely correct interval -> 3 points.)