2 Short Explanations [10 points]

Give brief explanations in the space provided below. **Long or unclear explanations will be penalized.**

Part A [2 points]

Explain why locks cannot be implemented simply by reading and writing a binary variable in memory.

Part B [2 points]

Explain why it is sometimes better to use spinlocks rather than block on a multiprocessor.

Part C [2 points]

A simple method for managing memory is to keep the memory partitions and holes in a linked list sorted by memory addresses. List any two changes you can make to the linked list algorithm (or data structure) that will improve the performance of memory management.

Question continues ...

Part D [2 points]

Describe two benefits of using a software-managed TLB.

Part E [2 points]

Clearly describe the difference between a page fault and a protection fault.

3 Short Explanations [24 points]

Give short explanations in the space provided below.

1. Would round-robin scheduling work in a batch system? Give two reasons why it is not used in batch systems.

   Yes it would work.

   1) Round-robin improves interactive performance, which is not a goal for batch systems.
   2) Batch systems aim to maximize throughput, while round-robin can reduce throughput compared to scheduling schemes like SJF because it context switches processes every time quantum.

2. Explain why The Shortest Seek First disk scheduling algorithm tends to favor middle tracks over the innermost or outermost tracks.

   The Shortest Seek First algorithm seeks to the next track closest to the current track. Assuming that disk requests are uniformly distributed over all tracks, consider the case when the current track is close to the outermost track. There is a higher probability of requests being on the inside rather than the outside of the current track, so the next request is likely to move the head away from the outermost track. A similar argument holds for the innermost track, so the head will favor middle tracks.

3. State three similarities and three differences between web browsers and operating systems.

   Similarities:
   1) Both run multiple applications (OS runs processes, browser runs web applications).
   2) Both need to isolate/protect different applications (OS isolates processes, browser isolates web applications).
   3) Both need to ensure that their own code is safe from application code.

   Differences:
   1) Operating systems use hardware protection to protect OS and application code while web browsers normally use software/language-level protection.
   2) Operating systems have direct access to hardware while browsers need to use OS to access hardware.
   3) Operating systems usually run local code while the main function of browsers is to run remote code and display remote data.

4. The Xax browser plugin allows running legacy application code within a web browser. Explain one benefit of running legacy application code **within** a web browser.

   1) Portability: Browsers are portable and hence code running within browser will be portable across OSs.

2) Security: Browsers run code within a tight sandbox, ensuring better security, while OSs provide a wide system call interface, that is easier to compromise.

5. A monitor can have two semantics, Hoare or MESA. Describe the differences between these semantics, and in one sentence explain how synchronization code using these two semantics is written differently.

   With Hoare semantics, when a waiting thread is signaled, it starts running immediately and the signaling thread is put to sleep (with higher priority than new incoming threads). As a result, the signaled thread is assured that the condition on which it was sleeping is true when it is woken up. With MESA, the signaling thread continues to run and the signaled thread can run only after the signaled thread exits the monitor. At this point, the signaled thread may compete with new incoming threads. Thus, the condition may not be true and the signaled thread has to test it again in a loop.

6. Explain the benefits and drawbacks of a two-level page table over a single-level page table. Give one example in which a two-level page table has no benefits over a single-level page table.

   Benefit: two-level page table takes less memory than single-level page table because normally many second-level page tables are not allocated.

   Drawback: two-level page table requires an additional memory access to access a page table entry.

   If a process were to use its entire address space, then the OS would need to allocate all the second-level page tables, in which case, a two-level page table would take more memory than a single-level page table.

7. Briefly explain what is write-ahead logging, and why it is used.

   Write-ahead logging is a technique used for crash recovery, i.e., to ensure file system consistency after a crash. It ensures that multiple disk operations needed for a file system operation are executed atomically. For each file system operatoin, all its disk operations followed by a commit record are logged to a separate area on disk. After the commit record reaches stable storage (i.e., become persistent), the disk operations are written/checkpointed to their final locations on disk. After a crash, a file system operation is considered completed only when its commit record is on disk. If this record is not on disk, then the operation was never checkpointed. If it is on disk, then the disk operations are assumed to be idempotent and checkpointed again.

8. Explain the difference between spatial locality and temporal locality. Give a specific example of accessing a data structure that illustrates temporal locality.

   Spatial locality occurs when memory accesses are close to memory accessed recently, or when processes tend to use a small fraction of their memory.

Temporal locality occurs when memory accessed recently is accessed again, or processes use the same memory over short periods of time.

Example of temporal locality: accessing an array multiple times, for example during a matrix multiply, each row of the left matrix is accessed multple times.

1. Short Questions [12 marks]

a) [3 mark] To solve the Dining Philosopher's Problem, suppose that diners sitting at a table are required to pick up one chopstick at a time, but must do so in alphabetical order by their last name (assume no duplicate last names). Does this solve the problem? Why or why not?

No. Diners can sit in alphabetical order and a deadlock can occur if everyone picks up left chopstick.

c) [3 marks] Say a thread is added to the run queue twice simultaneously (note that it is the same thread). Use an example to describe a problem that can occur.

Consider a producer in the producer-consumer problem. It needs to wait until the buffer is non-full. If the producer thread was in the run queue twice, the wait would not fully remove the thread from the run queue. The producer thread would continue running, possibly overwriting an item in the buffer.

b) [2 marks] Is it possible for a CPU scheduler with a 100-millisecond timeslice to spend over half its time in the OS context switch code? Assume that it takes the OS 1 millisecond to context switch the CPU. Justify your answer.

Yes. E.g., if all threads are thrashing.

d) [4 marks] Which of the following require special support from hardware to implement correctly? For those that do, mark a Yes, and name the hardware support needed.

Operation	Yes	Hardware support
System call	Yes	Mode switch
Context switch	Yes	Update address space register
Loading a program	No	
Creating a thread	No	
Acquiring a lock	Yes	Disable interrupt, atomic instructions
Releasing a lock	No	
Preventing deadlock	No	
Preventing starvation	No	

2. Short Questions [9 marks]

A) [3 mark] Explain why the clock algorithm loses accuracy as an approximation for LRU as physical memory size grows.

For each full rotation, the clock algorithm can only distinguish between pages that are referenced or not since the last rotation. The algorithm does not know the order in which pages are referenced within a rotation. As a result, the granularity over which the clock algorithm maintains recency is proportional to the number of physical pages, and its approximation to LRU decreases with increasing number of pages.

B) [3 marks] An architecture supports 4 kB pages and 4 MB superpages. A process uses 4 MB of virtual memory. Assume that the process's memory is properly aligned for a superpage. Give two example scenarios why many 4 kB pages might be a better way to map the region than one 4 MB superpage. Give one example scenario where one 4 MB superpage is a better option.

4KB is better:

1. Process has a constant small working set, say 2 pages.

2. Process has a changing, but small working set.

3. Process has a large working set at only one point in the program, say beginning or end.

4MB is better:

1. Process working set is close to 4MB.

2. It is an important process in the system whose pages should never be paged out.

C) [3 marks] What is the maximum file size for a file on a system with inodes containing 4 direct blocks, 2 indirect blocks, and 1 double-indirect block. The block size on the system is 1k bytes, and pointers on the system are 4 bytes.

4 + 2 * 256 + 1 * 256 * 256 blocks = 4 + 512 + 65536 = 66052 blocks = 66052 KB.

2. Multiple Choice Answers [10 marks]

Circle the correct choice. One or more answers may be valid.

1. Say a monitor uses 3 condition variables. In your monitor implementation in OS161, you would need the following number of waiting lists:

a) 1 b) 2 c) 3 [✓ d) 4] e) 5

3 for the condition variables, 1 for lock. The waiting lists are needed to wake up the threads waiting on the condition or the lock.

2. Switching threads is faster than switching processes (assume both are running in user-mode) because:

a) Interrupts do not have to be disabled when switching threads
[✓ b) Some registers may not have to be saved and restored when switching threads] e.g., page table register
c) Register values do not have to be saved or restored at all when switching threads
[✓ d) Assuming untagged TLB, TLB entries need not be invalidated when switching threads]
e) There are fewer mode switches when switching threads

no TLB invalidation because threads share same page table

3. The trapframe is used in the implementation of the `fork()` system call in OS161 for the following reasons:

[✓ a) To get the stack pointer value of the parent]
b) To use the stack of the parent for the child child has own stack
[✓ c) To get the program counter value of the parent]
[✓ d) To get the valid address space regions of the parent]
a) To get the return value of the `fork()` system call child returns its own value

4. A process invokes a system call. It may be moved to the ready state when:

a) It is waiting for a disk read to finish process will be moved to waiting state
[✓ b) It invokes the waitpid system call]
[✓ c) It invokes the getpid system call] at the end of system calls, scheduler may move current process to ready state
d) It invokes thread_sleep process will be moved to waiting state
e) It invokes thread_wakeup another process will be moved to ready state

5. An inverted page table may require more memory operations than a multi-level page table because:

a) It stores fewer page table entries
b) It requires matching on the thread id
[✓ c) The page table entries have lower locality]
d) The hash function computation is memory intensive
[✓ e) The hash function generates many collisions]

3. Short Questions [12 marks]

a) [4 mark] Clearly explain why the implementation of blocking locks in a multi-processor OS requires using spinlocks. You may give your answer in terms of two specific problems that might occur if spinlocks were not used.

The blocking lock implementation modifies the run queue which is a shared data structure and hence requires another locking method to safely update the run queue. Also, a spinlock is needed to ensure that the check between whether the lock is available or not and the call to sleep is atomic (other a lost wakeup is possible).

b) [4 marks] The `execv()` system call passes a user pointer (`argv`) as its second argument to the kernel. Describe two types of different checks that the kernel must perform on this argument. For each type of check, describe a problem that might occur if this check is not done.

- argv, argv[0], etc., lie in user address space, or else other programs/OS memory could be accessed or modified
- argv[0] terminates with 0 (argv[0] has some maximum size), or else a kernel buffer might overflow, or the memory address may lie outside user address space

c) [2 marks] When a thread exits in OS161, `mi_switch(S_ZOMB)` is called. Clearly explain the purpose of the `S_ZOMB` thread state.

When a thread exits, its state cannot be immediately destroyed because the OS executes in the context of the current thread. The S_ZOMB thread state allows the OS to destroy the state of the exited thread after a context switch when it is running in the context of another thread. The S_ZOMB thread state allows the waitpid call to retrieve the exit status of the child thread.

d) [2 marks] Compare blocking locks with yielding locks: Describe one scenario when blocking locks are better than yielding locks, and one scenario when yielding locks are better than blocking locks.

Block locks are better when the sleep will occur for a long time. In that case, the yielding lock will wake up the thread and put it back to sleep again and again, wasting processor resources. Yielding locks are better when the blocking lock holder thread is woken up frequently but then needs to sleep again because its condition is no longer true (e.g., a broadcast wakes up lots of processes but only one can enter the critical section after that, or lots of new processes keep trying to enter the critical section). In that case, yielding locks will limit the frequency of wakeups based on the number of processes in the system.

Question 2. Fun With Acronyms [10 MARKS]

For each of the following, (1) expand the acronym and briefly explain what it is, and (2) **state** whether it relates to an operating system **policy** or **mechanism**.

Part (a) [2 MARKS] MMU

Memory Management Unit

Hardware device that performs memory access permission checks and virtual to physical address translation

Mechanism

Part (b) [2 MARKS] FCFS

First Come First Served

A scheduling policy in which jobs are scheduled in the order that they arrive

Policy

Part (c) [2 MARKS] COW

Copy-on-Write

A method for sharing data/pages until the first write, after which a copy is made.

Mechanism

Part (d) [2 MARKS] ACL

Access Control List

A list associated with each object in a protection system (e.g. each file in a file system) that contains the subjects (e.g. users) and the access permissions that they have on the object (e.g. read/write/execute)

Mechanism

Part (e) [2 MARKS] LRU

Least Recently Used

A page replacement policy (or algorithm) that selects the page whose last access is farthest in the past (i.e., least recent) as the victim.

Policy

Question 3. Fill in the Blanks [15 MARKS]

Part (a) [3 MARKS] Consider a set of cooperating tasks, implemented as either a set of traditional single-threaded processes, or as a set of threads in a multi-threaded process. For each of the following operations or features, indicate whether the task will complete faster, or if the feature is easier to provide, if the tasks are implemented as single-threaded processes (**P**), kernel-level threads (**K**), or user-level threads (**U**). Assume a many-to-one mapping of user-level threads to kernel threads. If multiple options are equally good, list them all.

U_____ Creating new tasks (faster overall completion)

P_____ Preventing accidental modification of other tasks' private memory (e.g. stack) (easier to provide feature)

K,P_____ Making blocking system calls (faster overall completion)

U_____ Switching between tasks (faster overall completion)

K_____ Solving a parallel problem with fine-grained synchronization on a multiprocessor (faster overall completion)

U_____ Customizing task scheduling (easier to provide feature)

Part (b) [3 MARKS] Consider a system with a hardware-managed TLB, which provides no software interface to update TLB entries. The only software options are to flush the TLB (invalidate all entries) or invalidate a specific entry. For each of the following events, indicate whether the OS software should flush the TLB (**F**), invalidate an entry in the TLB (**I**), or take no action with respect to the TLB (**N**).

N_____ CPU interrupt

I_____ Page eviction

F_____ Context switch

N_____ File read system call

F_____ Fork system call, implemented with copy-on-write

I_____ Copy-on-write fault

This question continues on the following page

Part (c) [3 MARKS] Fill in the following table by writing if the combination of row and column headings for that cell is impossible (**I**), or writing if the combination is possible (**P**). The first cell (first row, first column) represents "TLB hit and virtual page in memory".

	Virtual page in memory	Virtual page not in memory	Invalid address exception
TLB Hit	P	I	I
TLB Miss	P	P	P

Part (d) [6 MARKS] In a Unix file system, indicate whether the following are stored in these locations: inode (**I**), directory entry (**E**), data block (**B**), single indirect block (**S**), double indirect block (**D**), triple indirect block (**T**), or in none of these data structures (**N**). If a value can be stored in multiple locations, write all of them.

I	Access control information
I, S	Pointer to data block
N	File descriptor
B	Symbolic link name
E, B	Directory name
B	Executable header
I,T	Pointer to double indirect block
E, B	Inode number
E, B	Hard link name
N	File position

1. Multiple Choice – Circle the best answer [10 marks; 1 each]

1. Hardware support for operating systems includes:
a) Mode bit
b) Context switch
c) Interrupts
d) (a) and (b) only
e) (b) and (c) only
f) (a) and (c) only
[✓ g)] (a), (b) and (c)
h) None of the above.

Context switch because it may need to switch MMU registers.

2. Executing a privileged instruction from a user-level process...
a) Has no effect (it is a no-op)
[✓ b)] Causes a trap to an OS function
c) Causes the immediate termination of the executing process by hardware
d) (a) or (b) are possible, depending on the instruction
e) (b) or (c) are possible, depending on the instruction
f) (a) or (c) are possible, depending on the instruction
g) (a), (b) or (c) are possible, depending on the instruction
h) None of the above.

3. Privileged instructions include:
a) Setting the Program Counter (PC) register
b) Setting the Stack Pointer (SP) register
[✓ c)] Setting MMU register
d) (a) and (b) only
e) (b) and (c) only
[✓ f)] (c) only
g) (a) and (c) only
h) (a), (b) and (c)

Either c) or f) is fine.

4. Possible transitions (that is, possible next states) for a process in the Ready state are:
a) New
b) Ready
[✓ c)] Running
d) Blocked
e) Exit
f) (c) and (d) only
g) (c) and (e) only
h) (c), (d) and (e) only

5. Possible transitions (i.e., possible next states) for a process in the Running state are:

a) New
b) Ready
c) Running
d) Blocked
e) Exit
f) (b) and (c) only
g) (b) and (d) only
[✓ h)] (b), (d), and (e)

6. In OS/161, the operating system identifies the system call requested by a process by:

a) The exception code stored in the processor status word.
b) A number corresponding to the system call, stored on the user process stack.
c) A string holding the name of the system call, stored on the user process stack.
[✓ d)] A number corresponding to the system call, saved in the trapframe.
e) A string holding the name of the system call, saved in the trapframe.
f) None of the above.

7. Kernel-level threads are better than user-level threads because:

[✓ a)] A thread waiting for IO operations will not block other threads.
b) Creating a new thread is faster.
c) Switching between threads is faster.
d) Thread synchronization operations are faster.
e) All of the above.
f) They're not. User-level threads are always better than kernel-level threads.

8. Which of these solutions can be used to prevent deadlocks:

a) When a process requests a resource that is already held, force the process holding the resource to release it.
b) Only allow a process to request a resource when no other process has acquired the resource.
c) Require each process to grab all desired resources at once.
d) (a) and (b) only.
[✓ e)] (a) and (c) only.
f) (b) and (c) only.
g) (a), (b), and (c).
h) None of the above.

9. Which of the following scheduling algorithms can lead to starvation:

a) First Come First Served
b) Shortest Job First
c) Round Robin
d) Priority
e) All of the above.
f) None of the above.
g) (a) and (c) only
[✓ h)] (b) and (d) only

10. The scheduler's goals for an interactive system typically include:

a) Minimize response time.
b) Prevent starvation.
c) Give each thread a fair share of the CPU.
d) Favour threads that wait for user input events.
[✓ e)] All of the above.
f) None of the above.
g) (a) and (c) only
h) (b) and (d) only

Question 2. Short Answers [16 MARKS]

Part (a) [4 MARKS] **Protection:** A hardware manufacturer has defined a new feature called User Page Protection (UPP). When the UPP bit is set in the processor status word control register, any attempt to access user-space memory while running in privileged mode causes a page fault. Give one reason why an operating system would use this feature. Describe two OS functions that would need to disable UPP protection.

A OS can use this feature to stop a buggy OS from randomly writing to user memory. OS functions that 1) copy data from user-space to kernel memory, or 2) copy to user-space from kernel memory, will need to disable UPP.

Part (b) [4 MARKS] **Scheduling:** Give two reasons why a multiprocessor CPU scheduler might assign processes to run on specific processors rather than using a global run queue that serves all the processors.

1. A local run queue will ensure that processes running on a processor continue to run on that processor, improving cache locality, and thus the performance of the application.
2. Locking a global run queue can become a scaling bottleneck on a system with a large number of processors.

Part (c) [4 MARKS] **TLB:** Assume the following code snippet:

```
int i;
int p[1024];

for (i = 0;i < 1024;i++) {
    p[i] = 0;
```

How many TLB misses will take place when this code is run for the first time? Provide a justification for each TLB miss. State any assumptions that you make. Assume a 1 KB page size.

Total = 6 misses.

1. 1 miss for the code region.
2. Data region has 1024 * 4 + 4 = 4100 bytes. With 1KB page size, 5 pages are required, so 5 misses for the data region.

The stack is not accessed, so there are no misses for the stack region.

Part (d) [4 MARKS] **File Systems:** A program issues the following system calls on a Unix file system:

```
int fd;
fd = open("/Foo", 0); /* Assume that the file exists and is more than 512
                       * bytes. The second argument specifies read-only. */
     read(fd, buf, 512);
```

Show the order in which the blocks are read (and the types of these blocks) to complete the system calls shown above. State the total number of block reads that are performed. You may assume an empty file buffer cache.

Total = 5 blocks.

1. Read superblock (normally this block is read when file system is mounted, but the question asks you to assume that the file buffer cache is empty, so this block would need to be read, unless the contents of the block are cached elsewhere).
2. Read inode block containing inode of “/”.
3. Read one (or more) directory data block(s) of “/” to find the “Foo” directory entry.
4. Read inode block containing inode of “Foo”. This block may be the same as the inode block containing the inode of “/”, in which case this read is not needed.
5. Read first data block of “Foo”.

Question 1. Short Answers [20 MARKS]

Part (a) [4 MARKS] **Threads:** Consider the user-level threads system that you implemented in the OS labs. Which of the following operations require support from the operating system (e.g., require issuing system calls) in their implementation? For those that do, write Yes, and provide a reason. Do **not** write Yes, if an operation does not directly issue system calls, but simply uses another operation that does.

Operation	**Yes**	**System Call Explanation**
Thread switching	no:	getcontext and setcontext are not system calls
Creating a thread	yes:	need to allocate stack dynamically, requiring sbrk
Disabling interrupts	yes:	disables Unix signals, requires system calls
Enabling interrupts	yes:	enables Unix signals, requires system calls
Thread exits itself	no:	disable/enables interrupts, switches thread
Thread preemption	yes:	requires signal support from OS
Thread sleep	no:	disable/enables interrupts, switches thread
Releasing a mutex lock	no:	disable/enables interrupts

1/2 mark for each yes, 1/2 mark for correct explanation, no explanation needed for 'no' answers

Part (b) [6 MARKS] **File Systems:** A program issues the following system call on a Unix file system:

```
ret = mkdir("/a/b", 755);
```

This system call creates directory "b" with Unix permission 755. Assume that the directory "a" exists and directory "b" is created successfully as a result of this call. Assume that the super block has already been read at boot time, but otherwise the buffer cache is empty, and all directories use one data block. Also, assume that the file system does not use journaling.

Show the types of blocks that are read and written when the system call is performed. Add a brief explanation (e.g., read inode of "a") for why the block is read or written beside each block type.

Read	Block Type	Explanation	**Write**	Block Type	Explanation
1	inode block	read inode of /	1	block bitmap	allocate block for b
2	dir block	find inode of a	2	inode bitmap	allocate inode for b
3	inode block	read inode of a	3	dir data block	write data for b
4	dir block	check if b exists in a	4	inode block	write inode for b
5	block bitmap	find empty block for b	5	dir data block	update data for a*
6	inode bitmap	find empty inode for b	6	inode block	update inode of a**
7	inode block	read inode of b *	7		
8			8		
9			9		

* this is to ensure the rest of the inodes in inode block are not overwritten in step 4 on the right side.

*create b dir in a
**e.g., update time stamp

This question continues on the following page.

Part (c) [4 MARKS] **Scheduling:** If a thread is CPU-bound, would it make sense to give it higher priority for disk I/O than an I/O bound thread?

Yes, for the same reason that I/O-bound threads should get higher priority for the CPU. If a CPU-bound thread must wait for I/O behind I/O-bound threads, we could end up in a situation where all of the threads are waiting for I/O and the CPU is idle. Better to finish the I/O for the CPU-bound threads as quickly as possible so they can get keep the CPU busy; this increases our chances of keeping all of the system components busy at all times.

Part (d) [6 MARKS] **Virtual Memory:** Answer the following for a demand paging system. Assume that the physical memory size is fixed.

a) Can doubling the page size reduce the number of page faults? If so, describe how. If not, describe why.

Doubling the page size behaves similar to prefetching pages. If a program accesses contiguous pages (e.g., pairs of pages), then increasing the page size will reduce page faults.

b) Can halving the page size reduce the number of page faults? If so, describe how. If not, describe why.

Paging systems cause internal fragmentation. Reducing the page size reduces internal fragmentation, potentially making more physical memory available (e.g., if a program was using a small part of a page, the smaller page size would still suffice, but the rest of the memory would be available to others). This will lead to fewer page faults and demand paging requests.

c) Suppose a friend told you: If a system has lots of runnable threads, it can use them to hide the cost of page faults. Explain whether your friend is right, wrong, or both.

Both. If the working sets of all the runnable threads fit in memory, then while one thread is waiting for a page fault another thread can execute, effectively hiding the cost of page faults. However, if the working sets do not fit in memory, then the system will quickly end up in a state where all threads are waiting for page faults; as soon as a thread resumes after a page fault, it will touch another page that is not in memory and generate another page fault to wait for. The more runnable threads there are, the worst things will get.

Question 2. Fill in the Blanks [15 MARKS]

Part (a) [3 MARKS] **TLB:** Consider a system with a software-managed tagged TLB. The tag uses 6 bits. The OS supports running a maximum of 1024 processes at a time. The processor provides the following options, in increasing cost, for programming the TLB: 1) invalidate a specific TLB entry, 2) invalidate the entries with a given tag, 3) flush the TLB (invalidate all entries). For each of the following events, indicate whether the OS software should take no action with respect to the TLB (**N**), invalidate an entry in the TLB (**I**), invalidate all entries associated with a tag (**T**) or flush the TLB (**F**). Use the most efficient option available. You may use the space on the right to explain your answer (optional).

T	Context switch	note that multiple processes can map to the same tag, so we will need to invalidate all entries if another process with same tag ran the last time
I	Page eviction	update mapping to invalid page
N	Return from a recursive function call	no change to address space
T	Process exit	invalidate all entries associated with this process
I	Copy-on-write fault	update mapping to writeable page
N	Fork system call, without copy-on-write	no change to address space of current process

Part (b) [4 MARKS] **OS Concepts:** Hardware support for operating systems includes privileged instructions (**P**), MMU support for virtual memory (**M**), interrupts (**I**), and the trap instruction (**T**). Indicate how the OS takes advantage of (one or more of) this hardware to ensure or support the following:

M	a program doesn't modify the memory of another	
M, P	a program doesn't access hardware directly	The P is for i/o operations
I	a program doesn't run forever	
M	a program doesn't jump to an arbitrary OS address	
M, P	a program doesn't modify its virtual memory mapping	The P is for MMU operations
T	a program updates a block on disk	Need system call
T	a program sends a message to another process	Need system call
T	a program waits to receive a message from another process	Need system call

This question continues on the following page.

Part (c) [5 MARKS] **Disk scheduling:** Your OS *only* allows reading and writing whole files (similar to the web server implementation in your lab that read entire files). When files are accessed, the OS issues disk requests for all the blocks of the file. These requests can be scheduled using various disk scheduling algorithms.

We have discussed the First-Come-First-Served (FCFS), Shortest-Seek-First (SSF) and the Elevator scheduling algorithms in class. In addition, let's consider two other scheduling algorithms. The Shortest-File-First (SFF) is a non-preemptive algorithm that schedules all block requests for a file at a time, prioritizing the shortest file first. The Round-Robin (RR) algorithm is a preemptive algorithm that schedules requests one block at a time for all files that are being accessed concurrently.

These disk scheduling algorithms have various desirable properties shown below.

Exploit Locality: whether the algorithm exploits request locality.

Bounded Waiting: whether the algorithm bounds waiting time for all block requests it has received.

Fairness: whether the algorithm provides fairness, meaning that each block request is treated equally, regardless of what position on the disk the request is referencing.

Fill in the cells of the table below, using a check mark to indicate that the algorithm has the desired property. If the algorithm does not have the desired property, leave the table cell blank.

	Exploits locality	**Bounded waiting**	**Fairness**
FCFS		X	X
SSF	X		
Elevator	X	X	
SFF	X*		
RR		X	X

* assumes that small files are placed mostly sequentially

Part (d) [3 MARKS] **Page Replacement:** You are familiar with the FIFO, Clock and LRU page replacement algorithms. The MRU page replacement algorithm replaces the *most* recently accessed page. Simply state **True** or **False** for the following statements regarding these four page replacement algorithms. You may use the space below each statement to explain your answer (optional).

	True or False
Clock and FIFO are easier to implement that MRU.	True
A program accesses an array that is slightly larger than physical memory size repeatedly. MRU is the best choice for this program. MRU page will be accessed furthest in the future.	True
A program accesses a large array just once. LRU is the best choice for this program. MRU is better because the data should be replaced right away since it will not be used in the future.	False
A program accesses the elements of a large array randomly. All the four page replacement algorithms will work equally well for this program.	True
A program makes a deeply recursive function call. FIFO is the best choice for this program. FIFO and LRU will make the same page replacement decisions, but FIFO is more efficient.	True
All the four page replacement algorithm benefit from hardware support for tracking page accesses. FIFO doesn't need reference bits.	False

Question 1. All False [5 MARKS]

Your friend makes the following statements below. You know they are not correct. Clearly explain to your friend **why** they are incorrect. We have provided an example below. Single sentence answers are preferred.

0) On a uni-processor, when a process makes a system call, the system call must return to the process before another process can make a system call.

Answer: A system call can block in the kernel, after which another process can run and issue a system call before the first process's system call returns.

A) When a process invokes a system call, a context switch occurs.

When a process invokes a system call, the OS may return control to the same process, so a context switch will not occur. Some students mentioned that when a process invokes a system call occurs, then a mode switch occurs. This is correct but this explanation doesn't contradict the statement above.

B) A divide-by-zero error in a program invokes the trap instruction.

A divide-by-zero error causes a h/w exception. (a program has to invoke a trap instruction explicitly).

C) A program never executes code after the Unix execve() system call.

A program can execute code located after the execve() system call if the system call returns an error.

D) A Unix wait() system call will always cause a process to block.

A wait() system call will not block if the child process doesn't exist (has exited or wasn't created or on any error).

E) A setcontext() call will always enable signals.

A setcontext call will set the signal state to the state saved by the previous corresponding getcontext call, so it may not enable signals.

Question 1. True / False [8 MARKS]

TRUE [✓ FALSE] A thread of a process can read but not modify the local variables of another thread in the same process.

A thread has access to the entire address space of a process and can read or modify any part of it.

TRUE [✓ FALSE] The atomic test-and-set instruction requires that interrupts are disabled when the instruction executes.

The programmer doesn't have to disable interrupts when issuing atomic instructions

TRUE [✓ FALSE] The shorter the time slice, the more the round-robin scheduler gives results that are similar to a FIFO scheduler.

The *longer* the time slice, the more the round-robin scheduler is similar to a FIFO scheduler.

[✓ TRUE] FALSE In a paging system, a context switch requires modifying the page table base register.

True when h/w manages TLB. Even with software-managed TLB, the OS needs to change the page table base address (the page table that is used).

TRUE [✓ FALSE] A MMU would not be useful if the machine has more physical memory than the sum total of the virtual address space size of all running programs.

A MMU is useful because it allows virtualizing addresses. For example, program addresses can start at 0, which simplifies compiling, loading and debugging programs.

[✓ TRUE] FALSE The page fault frequency algorithm helps decide how pages should be allocated across processes but not within processes.

TRUE [✓ FALSE] A program compiled with dynamically linked libraries has a smaller working set than the same program compiled with statically linked libraries.

The working set is based on the pages used by a process. This doesn't change, whether dynamic or static libraries are being used. Dynamic libraries enable sharing reducing the amount of physical memory frames that need to be allocated.

[✓ TRUE] FALSE Compared to a regular file system, the log-structured file system improves the performance of file writes, but reduces the performance of file reads.

LFS improves write performance because it mostly writes blocks sequentially. It reduces read performance because file data can get spread over the disk, as parts of it are overwritten.