CS111 Midterm Exam

[redacted]

TOTAL POINTS

99 / 110

QUESTION 1

Interface vs Implementation 10 pts

1.1 Consequences of incompatible change 3 / 3

✓ **- 0 Correct**

- 1 Application might stop working after the OS upgrade.

- 2 Application might stop working after the OS upgrade.

1.2 Dealing w/incompatible change 2 / 3

- 0 Correct

✓ **- 1 They might also have to distribute different versions of their software to run on different versions of OS.**

- 2 Independent software vendor should modify the program to work with new interface, rebuild it and ship new version to affected customers. They might also have to distribute different versions of their software to run on different versions of OS.

- 3 Click here to replace this description.

1.3 How specifications would help 3 / 4

- 0 Correct

✓ **- 1 Clear interface specification makes the risk more obvious when an interface changes due to OS update.**

- 2 Interface specifications are important as they make the new changes to the API's less likely. Clear interface specification makes the risk more obvious when an interface changes due to OS update.

- 4 Interface specifications are important as they make the new changes to the API's less likely. Clear interface specification makes the risk more obvious when an interface changes due to OS update.

- 3 Interface specifications are important as they make the new changes to the API's less likely. Clear interface specification makes the risk more obvious when an interface changes due to OS update.

QUESTION 2

Convoys 10 pts

2.1 What is a convoy 2 / 2

✓ **- 0 Correct**

- 1 A resource convoy is a persistent queue of processes waiting to get access to a popular resource, which eliminates parallelism and increases delays and reduces system throughput.

- 2 A resource convoy is a persistent queue of processes waiting to get access to a popular resource, which eliminates parallelism and increases delays and reduces system throughput.

2.2 How it forms 2 / 2

✓ **- 0 Correct**

- 1 The key to convoy formation is that the processes are no longer able to immediately allocate the resource, but are always forced to block.

- 2 The key to convoy formation is that the processes are no longer able to immediately allocate the resource, but are always forced to block. It may be precipitated by process becoming blocked or preempted while holding the resource.

2.3 How to eliminate 5 / 6

- 0 Correct

✓ **- 1 Click here to replace this description.**

- 2 Implement read/write locks to reduce mutual exclusions

- 3 Click here to replace this description.

- **4** check the comment
- **5** Click here to replace this description.
- **6** Click here to replace this description.

QUESTION 3

Preemption and Blocking 10 pts

3.1 Events leading to Preemption 3 / 5

- **0** Correct
- ✓ **2 First event wrong.**
- **2** Second event wrong.
- **4** Both events wrong.
- **2** Both events are (almost) the same.
- **4** Both events wrong.
- **1** First event half correct.
- **1** Second event half correct.

3.2 Events leading to blocking 5 / 5

- ✓ **0 Correct**
- **2** First event wrong
- **2** Second event wrong
- **4** Both events wrong
- **2** Both events are (almost) the same/similar
- **1** First event half correct
- **1** Second event half correct

QUESTION 4

Mutua Exclusion 10 pts

4.1 Evaluation criteria 4 / 4

- ✓ **0 Correct**
- **1** First criteria wrong
- **1** Second criteria wrong
- **1** Third criteria wrong
- **4** No answer

4.2 Evaluate Spin-Locks 2 / 2

- ✓ **0 Correct**
- **1** Partial correct
- **2** Wrong

4.3 Evaluate Interrupt Disables 2 / 2

- ✓ **0 Good**
- **1** Partial correct
- **2** Wrong

4.4 Evaluate Mutexes 1 / 2

- **0** Good
- ✓ **1 Partial correct**
- **2** Wrong

QUESTION 5

Dynamically Loadable Libraries 10 pts

5.1 Benefit vs shared libraries 4 / 4

- ✓ **0 Correct**

5.2 When it would be needed 1 / 3

- **0** Correct
- **2 Point adjustment**

5.3 Additional mechanism required 2 / 3

- **0** Correct
- **1 Point adjustment**

QUESTION 6

Shared Memory vs messages 10 pts

6.1 Advantage of shared memory 2 / 2

- ✓ **0 Correct**
- **1** Shared memory communication happens without any system calls, using ordinary instructions to copy data at memory speed.
- **1** Have to point out advantage is performance
- **2** Incorrect answer

6.2 How it gains this advantage 2 / 2

- ✓ **0 Correct**
- **1** Shared memory use ordinary user-mode instructions, OS not involved
- **1** Should be more specific
- **2** Not explaining why fast

6.3 Two advantages of messages 6 / 6

- ✓ **0 Correct**
- **2** Advantage #1
- **1** Reason #1: why shared memory IPC cannot achieve
- **2** Advantage #2
- **1** Reason #2: why shared memory IPC cannot achieve

QUESTION 7

Variable Partition Free Lists 10 pts

7.1 required fields 4 / 4

✓ **- 0 Correct**

- 4 Address and size of chunk, pointer to next chunk in list.

7.2 operations to optimize 3 / 3

✓ **- 0 Correct**

- 1 search and coalesce

- 2 search and coalesce

7.3 additional diagnostic info 3 / 3

✓ **- 0 Correct**

- 1 feature to add

- 1 how to maintain

- 1 how it is used to detect/prevent

QUESTION 8

8 semaphores to distribute requests 10 / 10

✓ **- 0 Correct**

QUESTION 9

Page faults 10 pts

9.1 hardware fault handling 3 / 3

✓ **- 0 Correct**

- 1 TLB miss generates trap, gets PC/PS from trap vector, pushes prev PC/PS onto supv stack, transfer to 1st level handler, save regs, call 2nd level handler

- 2 TLB miss generates trap, gets PC/PS from trap vector, pushes prev PC/PS onto supv stack, transfer to 1st level handler, save regs, call 2nd level handler

- 3 invalid page generates trap, gets PC/PS from trap vector, pushes prev PC/PS onto supv stack, transfer to 1st level handler, save regs, call 2nd level handler

9.2 software page fault handling 4 / 4

✓ **- 0 Correct**

- 1 is referenced page legal, allocate new page frame (perhaps evicting another page), schedule read of desired page, block process, when it completes, update page table (&perhaps TLBs).

- 2 is referenced page legal, allocate new page frame (perhaps evicting another page), schedule read of desired page, block process, when it completes, update page table (&perhaps TLBs).

- 3 is referenced page legal, allocate new page frame (perhaps evicting another page), schedule read of desired page, block process, when it completes, update page table (&perhaps TLBs).

- 4 is referenced page legal, allocate new page frame (perhaps evicting another page), schedule read of desired page, block process, when it completes, update page table (&perhaps TLBs).

9.3 return and resumption 3 / 3

✓ **- 0 Correct**

- 1 back-up failed instruction, restore saved registers, return-from-trap to saved PS/PC

- 2 back-up failed instruction, restore saved registers, return-from-trap to saved PS/PC

- 3 back-up failed instruction, restore saved registers, return-from-trap to saved PS/PC

QUESTION 10

Condition Variables 10 pts

10.1 Why a mutex 2 / 2

✓ **- 0 Correct**

- 1 protect critical sections involving condition variable updates, signals, and waits. The worst is a CV being signaled after a process checks it, but before the process sleeps.

- 2 protect critical sections involving condition variable updates, signals, and waits. The worst is a CV being signaled after a process checks it, but before the process sleeps. Condition variables do not guarantee mutual exclusion or even availability of a resource.

10.2 Use example 2 / 4

- 0 Correct

✓ **- 2 No signal**

- 2 No wait

- 4 No

- 2 no locking

10.3 What OS does w/Mutex in wait 2 / 2

✓ **- 0 Correct**

- 1 pthread_cond_wait releases mutex while process is blocked and reacquires it when it resumes ... to enable others to get mutex and call pthread_cond_signal

- 2 pthread_cond_wait releases mutex while process is blocked and reacquires it when it resumes ... to enable others to get mutex and call pthread_cond_signal

10.4 Doable in user-mode? 2 / 2

✓ **- 0 Correct**

- 2 No

- 1 vague/unconvincing justification

- 2 doesn't address this race

QUESTION 11

Memory Allocation techniques 10 pts

11.1 Benefits of heap allocation 3 / 3

✓ **- 0 Correct**

- 1 The huge advantage is storage that remains usable after the allocating block exits.

- 2 The huge advantage is storage that remains usable after the allocating block exits.

- 3 No

11.2 Benefits of malloc vs sbrk 4 / 4

✓ **- 0 Correct**

- 2 vague, hardly a key capability as described

- 3 fewer parameters is not a key capability

- 4 no difference in this respect

- 4 sbrk can only allocate/deallocate contiguous storage. malloc manages numerous discontiguous chunks.

- 4 We can call sbrk at any time and increase/decrease our data segment size by any amount.

- 4 ???

11.3 Benefits of mmap vs sbrk 3 / 3

✓ **- 0 Correct**

- 2 And this is a better malloc arena beacause ...

- 3 mmap can be used to create multiple, discontiguous heaps ... which might be handy for per-core or slab-allocation.

- 3 how is this a better way to augment the MALLOC ARENA?

- 3 ???

[redacted]

All questions are of equal value. Most questions have multiple parts. You must answer every part of every question. Read each question CAREFULLY; Make sure you understand EXACTLY what question is being asked and what type of answer is expected, and make sure that your answer clearly and directly responds to the asked question.

Many students lose many points for answering questions other than the one I asked. Misunderstanding a question may be evidence that you have not mastered the underlying concepts. If you are unsure about what a question is asking for, raise your hand and ask. Spend more time thinking and less time writing. Short and clear answers get more credit than long, rambling or vague ones. Write carefully. I do not grade for penmanship, spelling or grammar, but if I cannot read or understand your answer, I can't give you credit for it.

1.

2.

3.

4.

5.

6.

7.

8.

9.

10.

SubTotal:

XC

Total:

1: (a) Consider an after-market application, developed and shipped separately from the OS. What could happen if the OS vendor relased a new OS version with a non upwards compatable API (and associated ABI) -change?

Non upwards compatable means it will not support previous interfaces. Thus systems running the new OS will not support the after-market applicaten as it is based on no longer supported interface.

(b) What would the application developer have to do to deal with this?

The developer would have to rewrite code to be compatible with the new API/ABI changes, changing the implementation to comply w/ the updated interface.

(c) Explain how/why interface specifications, designed and written independently from the current implementation, might have affected this situation.

If they are written seperatly from considering current implementation, they do not consider how the interface is currently being used and thus may break the previous contract that was in place when the implementation was set. If they had considered current implementation, may have used polymorphism to support previous interface.

2: (a) What is a resource contention convoy?

It is when there is demand for a resource, but something such as context switching / fairness policy causes there to be a backup in a queue waiting for the resource. This then causes a dramatic rise in response time as this resource is waited on.

(b) Under what circumstances is one likely to form?

Convoy may form when there is demand for a resource from many competing threads/processes but due to a fairness policy, a requesting thread/process always needs to wait in a FIFO queue vs. a policy of progress that allows available resources to be instantly grabbed w/ the cost of context switch.

(c) Suggest two distinct approaches to eliminating (or SIGNIFICANTLY improving) the problem.

1) Implement policy of progress over fairness, thus allowing greatest utilizaten and increasing threughput

2) Put a limit on queue length to ensure convoy doesn't form, keeping response time manageable.

3: (a) List two different types of events that might cause a running process to be preempted.

1) If there is an interrupt.

2) If a process goes past its alloted time slice.

(b) List two different types of events that might cause a running process to become blocked.

1) If it does an I/O request.

2) If it is waiting on an asynchronous completion of some condition determined by condition variable/semephore.

4: (a) Identify three key criteria in terms of which mutual exclusion mechanisms should be evaluated.

a) fairness

b) performance

c) Correctness / Mutual Exclusion.

(b) Evaluate spin-locks against these criteria.

Spin locks:

a) They are not fair, potential starvation, unbonded wait times

b) Very weak performance on uniprocessor, may actually prevent necessary process from finishing use of locked object. Also wastes CPU cycles

c) They do correctly provide Mutual exclusion

(c) Evaluate interrupt disables against these criteria.

a) They are fair as there is no convey formation.

b) They will affect performance negatively overtime as interrupts may be critical to system performance.

c) They accurately prevent preemption thus allowing mutual exclusion on uniprocessor, NOT correct on multiprocessor as a thread on another processor may be interrupted.

(d) Evaluate mutexes against these criteria.

a) They are not necessarily fair as may be unbounded wait times waiting to get mutex if many competing processes

b) Performance may be poor if there are long wait times as a process waits, especially if interrupt while a process holds the mutex.

c) They are correct if implemented properly.

5: (a) Describe a major capability (not merely memory savings) of DLLs that cannot be achieved w/mere shared libraries.

Shared libraries may waste address space by including library functions that are never needed, while DLL's only map in necessary functions/methods when needed.

(b) Describe a specific situation where and why this capability would be NECESSARY.

For a program that is running on an extremely limited address space due to small physical memory size or lots of processes running concurrently, this is necessary as space is at a premium and a large shared library could occupy way too much of the address space, especially if the heap is doing lots of memory allocation as well.

(c) Briefly describe a significant mechanism that is not required to support shared libraries, but is required to support DLLs ... and very briefly explain why this additional mechanism is necessary.

Procedure Linkage Table -

Needed to map in what methods are called at runtime, but shared library does not need this because it relies on stub modules at linkage time to provide method names but DLL's do not have stubs, instead solely dynamic binding.

6: (a) What is the primary advantage of shared memory IPC over message communication?

Extremely fast

(b) Why does it have this advantage?

Because this memory is highly accessible to CPU and can be accessed by multiple processes through direct mapping into address spaces.

(c) List TWO advantages of message IPC over shared memory, and why each cannot reasonably be achieved with shared memory IPC.

a) Isolation/Privacy: in shared memory each process can access the same data so it is difficult to ensure that one has private access while messages leads to isolation until explict sharing requested.

b) Safety/Simplicity: Shared memory requires locks to ensure that data is not lost/overwritten which is more complicated to implement while messages are distinct to each process and can be implemented w/o worrying over data overwrite.

7: (a) list two pieces of information that a variable partition free-list must keep track of that might not be needed with fixed partition memory allocation.

1) size of partition
2) previous/next partition for coalescing

(b) list two operations that a variable-partition free list has to be designed to enable/optimize (zero points for "allocate and free").

1) Splitting chunks to fit requested partition size
2) Coalescing to prevent external fragmentation

(c) list an additional piece of information we might want to maintain in the free list descriptors to detect or prevent common errors, and briefly explain how the information would be maintained, and how it would be used to detect or prevent a problem.

We might want to maintain a buffer zone at the beginning and end of each free list partition to ensure that if it is written on then we know there was an out of bounds reference by another thread/process. Implemented by allocating a few bytes at beginning/end of each chunk as specific buffer zone when it written on throws an exception and notes the writing process id. This can help us diagnose free list.

8: Consider a server front-end that receives requests from the network, creates data structures to describe each request, and then queues them for a dozen server-back-end threads that do the real work. Sketch out server and worker-thread algorithms that use semaphores to distribute/await incoming requests, and protect the critical sections in queue updates.

- Need to ensure server only adds/puts requests when the queue buffer space is not overfilled. - empty is variable used for this
- Need to ensure worker threads only gets requests when they are available w/o violating critical section - filled is used for this

3 - semaphores used are mutex, empty, filled

Server: Single server

While
- Call wait on empty semaphore lock
- Call wait on mutex semaphore lock
- put(struct request)
- Call post on mutex semaphore lock
- Call post on unfilled semaphore lock

Worker: Multiple workers

While
- Call wait on filled semaphore lock
- Call wait on mutex semaphore lock
- get(request = get())
- Call post on mutex semaphore lock
- Call post on empty semaphore lock

- initialize mutex semaphore counter to 1
- initialize filled/empty semaphore counters to 0/buffer size respectively
- put/get each deal w/ accessing queue as put has struct request to add to queue and get returns struct request from queue

9: Briefly list the sequence of (hint: 8-10) operations that happens, in a demand-paging system, from the page-fault (for a page not yet in main memory) through the (final, successful) resumption of execution.
(a) describe the hardware and low level fault handling.

1) Page fault occurs. Trap instruction treds into page fault handler through trap vector table.
2) [illegible]
- Load PC/PS from vector table of 1st level handler which saves PC/PS, of current process and general registers instruction on stack and then looks at locals up 2nd level trap handler to handle the following instructions

(b) describe the software lookup, selection, I/O, updates.

- Lookup in process page table where the page is in disk
- Have memory mayement unit allocate space for page in memory, possibly needing to select to swap out to disk another page that hasn't been used with global LRU / working set LRU algorithm.
- Make I/O request to accomplish selected swapout (if necessary) and to swap in needed page to memory.
- Update page table w/ new physical page number in memory and update valid bit of page in page table

(c) describe the return/resumption process.

- Return to 1st level trap handler and
- Reload saved PC/PS, and registers with privileged instruction, unblock process
- Resume at same PC instruction as when page fault occured

10 c) which pthread mutex system call

10: (a) Why is each Linux Condition Variable ~~pared~~ paired with a mutex? Briefly describe the race condition that is being managed.

To ensure that the condition does not change in the time that it was last checked and then when it executes the wait instruction to sleep - this is the race condition. The mutex ensures that from the time it checks the condition, nothing can change it until it goes to sleep with wait and releases the mutex. This prevents an infinite sleep caused by a post/wake signal (happening after the condition is checked and before it is put to sleep.

(b) Write snipppets of signal and wait code, illustruatrating correct use of the condition variable to await a condition.

```
pthread_lock(mutex);
while (condition == 0)
    pthread_cond_wait(needed, mutex);
.... execute code
pthread_unlock(mutex);
```

(c) What will the operating system do with the mutex, during which system call(s)?

pthread_cond_wait() - it releases the mutex after putting the calling process to sleep. It then reacquires the lock upon a complementary post() command that awakens puts the process as ready before it moves into checking the while condition again as running.

(d) Could we do this for ourselves? If so, how? If not, why not?

We cannot because we could not release the mutex once we are already sleeping, nor could we reacquire the mutex once awakened before we start running again. This needs to be done for us by the OS.

XC: (a) Heap allocation is much more complex than stack allocation. What key capability do we gain by using heap allocation functions like malloc(3) rather than stack allocation?

We gain the ability to allocate as the programmer during runtime and have allocated memory exist beyond the local scope as occurs w/ stack.

(b) Heap allocation is much more complex than direct data segment extension and contraction with sbrk(2). Ignoring the higher cost of system calls (vs subroutine calls), what key capability do we gain by using heap allocation functions like malloc(3) rather than sbrk(2)?

We gain the ability to carve off specific chunks of memory vs. sbrk which extends the end of the segment. We can then use accompanying mfree(3) to reclaim this memory during runtime, thus recycling heap memory at a relatively affordable cost.

(c) It was briefly mentioned that mmap(2) could be used as an alternative to sbrk(2) to increase the usable data size in a process' virtual address space. What practical benefit/ability might we gain by using mmap(2) rather than sbrk(2) to augment the malloc arena?

Sbrk() allows us to extend the end of a segment but if the heap space will begin to overwrite another segment in the virtual AS it will fail, but mmap() allows us to map to a new space in physical memory that can provide a larger overall malloc arena.