CS111 Midterm Exam

Jeffrey Yufu Qiu

TOTAL POINTS

79 / 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 1 / 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 2 / 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.

Round robin scheduling can still cause resource convoys.
Multi-level queues may result in starvation of low priority processes.

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 1 / 2

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

4.4 Evaluate Mutexes 2 / 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 3 / 3

- **0 Correct**

5.3 Additional mechanism required 3 / 3

- **0 Correct**

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 2 / 10

- 0 Correct
- 8 **Point adjustment**

Check Posted Solution.

QUESTION 9

Page faults 10 pts

9.1 hardware fault handling 2 / 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 4 / 4

- 0 **Correct**
- 2 No signal

- 2 No wait
- 4 No
- 2 no locking

10.3 What OS does w/Mutex in wait 0 / 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? 0 / 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 1 / 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 0 / 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 1 / 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 ???

CS 111 Midterm 05/04/2017

Name Jeffrey Qiu Seat Row F

Student ID # 804 808 179 Exam # 97 Seat Col 6

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:

a desktop app

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?

The application could completely fail to function because the new OS may no longer support the interface of the old API, which the app was designed around

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

The app developer would need to rewrite his software such that it would comply w/ the interface of the API of the new OS He doesn't worry about ABI bc the new API is bound to hardware w/ updated ABI.

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

If the interface specs were to be completely indept then the app developer would not need to worry about upwards compatability bc the interface of the API stayed the same. Even if OS is updated it's okay bc the update will only update implementation behind the interface to ABI and subsequently hardware

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

A resource contention convoy is when many jobs / threads are attempting to acquire a resource such as a lock. Then a long job holds the resource and many shorter (and longer) jobs get held up in an increasingly longer queue.

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

If you have a poor scheduling algorithm such as FIFO then when a really long job comes in it will make all the subsequent jobs wait for it to finish.

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

- Use a scheduling algorithm that will routinely check queued jobs and preempt one if it has used up a certain amount of time. (time slice).
- Use history of jobs by means of good accounting
- Keep a queue of scheduled jobs to achieve fairness of distributing resource among requesters.

interrupts, signals, or exceptions, (traps)

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

Timer (Hardware) Interrupt.

Illegal address access or reference will result in a trap & later signal

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

If a running process makes an I/O call then it will get blocked and late moved to a waiting state.

If a process calls a Sleep() function in order to save on performance while waiting on a resource

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

Correctness: Does it actually achieve mutual exclusion. Are there logical flaws?

Fairness: Does each job or thread requesting to acquire the lock get a fair shot at it and does the mechanism avoid starvation?

Performance: Is there reasonably low overhead w/ acquiring and releasing locks?

(b) Evaluate spin-locks against these criteria.

When spin locks are implemented w/ test-and-set and compare-and-swap they are correct and mutually exclusive

However, they are neither fair nor performing well b/c they always spin when waiting for lock wasting CPU cycles. Also it's random lock acquisition, so a thread could potentially starve.

(c) Evaluate interrupt disables against these criteria.

Interrupt disables are not correct on multiprocessors systems b/c while one core disables interrupts the other can still run and therefore there is no true mutual exclusion when entering a critical section

Performance is low b/c all other jobs & threads blocked. No fairness either, b/c there is no waiting queue

(d) Evaluate mutexes against these criteria.

Mutexes operate correctly and perform fairly because they have a waiting queue to decide which waiting ~~process~~ or thread gets to next run

Performance really depends on the implementation and granularity of the lock. However, mutexes are better than spin locks b/c requesters yield

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

DLLs can be loaded at runtime w/ an explicit call to the function dlopen(), allowing for a federation network and deferred library binding.

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

Consider a situation where we have different implementation of a similar interface (i.e. FAT32 and other filesystems) We want to utilize a certain library that we can only know during runtime b/c of this deferred binding.

(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.

The runtime loader that will link the DLL requires some extra symbols to declare that there exists an external declaration in some kind of format. The format will be a general driver like module that the DLL will then need to satisfy the interface of. Imagine browser plugins and such.

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

Shared memory is incredibly fast on the scale of microseconds.

(b) Why does it have this advantage?

B/c shared memory literally looks at the same physical memory and effectively is as fast as electricity can update our HW memory

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

Message IPC is a well defined protocol between two processes that provides security and isolation that shared does not. On distributed systems we can use msgs but not shared memory. Shared memory restricts the processes to the same local machine.

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.

It must include the size of the chunks of memory it is keeping track of. And it could also have a pointer to the beginning of the next element in free list.

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

You need coalesce() to combat external fragmentation and split() to combat internal 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.

To protect our memory we could include two "canaries" that exist before and after each block of memory. This magic number would be used to authenticate accesses by making sure the data wasn't tampered with

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.

BTW for server process > thread

```
Server() {
    Request req[100], temp;
    while(temp = listen() != -1){
        req.push_back(temp);
    }
    for(int i=0; i< req.size ; i++){
        pthread_create(worker, NULL, req[i]);
    }
    for(int j=0 ; j< req.size ; j++) {
        pthread_join(threadId[j]);
    }
    return 0
}
```

```
Worker(Req_req)
    pthread_mutex_lock();
    pthread_cond_wait
    Handle(req)
    update(req[i]);
    pthread_mutex_unlock();
    pthread_mutex_signal();
}
```

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.

First a page fault is raised after we find that the page we want is not in memory. So, the user process that requested the "non existent" address will raise an exception trap that will tell the OS to have the first level trap handler preempt and save the user process' registers and PC before turning on the kernel mode bit and index into the trap vector to let the OS know how to service the exception.

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

The OS assumes control to find that a requested page is neither in memory nor the TLB. So, it looks up the PTE in the page table, translates the VPN into a PPN and retrieves the page from secondary storage and brings it back to memory, turns on the present bit and stores it in the TLB. If a page needs to be evicted in the TLB or memory then they can choose a page based on Global LRU or Working Set (stealing an unfrequently page of another process)

(c) describe the return/resumption process.

After completing the service the OS then issues a return from trap instruction from the 2nd level handler to 1st level, and restores the registers and user process state by popping it off the k-stack before setting privilege bit off and back to user mode. Upon return we rerun the same instruction that raised the page fault which will now no longer raise the page fault bc the page is in the Page Table and subsequently moved into the TLB.

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

Within each mutex we now have a new data structures such as the queue (and counter) that allows for fairness by knowing which thread runs next. As such we need to protect ourselves with a CV from the race condition of mutex request that will not only access the lock but also the queue inside the mutex.

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

```
consumer() {
    pthread_mutex_lock(empty)
    pthread_cond_wait()
while(pthread_mutex_lock(empty));
    get()
    pthread_mutex_unlock(empty)
    pthread_cond_signal();
    exit()
}
```

```
producer() {
    pthread_mutex_lock(full)
    pthread_cond_wait()
while(full);
    put()
    pthread_mutex_unlock(full)
    pthread_cond_signal()
    exit()
}
```

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

P() or pthr_mutex_lock will decrement

V() or pthread_mutex_unlock will increment.

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

You can but it would be impractical because you'd need access to some kernel data structures to manage the mutex which is incredibly costly operation.

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?

w/ heap alloc like malloc(3) we get to define as a user exactly how much memory we want allocated

malloc also automatically manages the free list alongside its buddy function free().

(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)?

Malloc() is a much safer function with protection that is built around the system calls like brk() and Sbrk().

brk and sbrk simply expand the available memory by pushing forward the "break". We would need to check if we are stepping on somebody else's memory and malloc() updates and searches through the free list accordingly, splitting and coalescing which brk does not.

(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?

mmap() accepts parameters that allows us more helpful custom features / settings we can decide about the chunk of memory we're setting aside giving us a more convenient way to decide options such as protection and permissions of the memory we're allocating.