Principles of Computer System Design

Homework - 3 Part-A

a) Which of the following adjustments will allow files larger than the current 1-gigabyte limit to be stored?
   a. Increase just the file size field in the inode from a 32-bit to a 64-bit value.
      False. Just increasing the file size won't help. The maximum file size is fixed to about 1GB by the inode number and the indirect blocks given.
   b. Increase just the number of bytes per block from 512 to 2048 bytes.
      True. The triple indirect block will give up to (block size) times (block size/4)^3 bytes. It will result to 256 GB for 2048-byte blocks. The maximum file size will be 4 GB for 32-bit size field in the inode.
   c. Reformat the disk to increase the number of inodes allocated in the inode table.
      False. The increment of inode numbers will increase the number of files and will have no effect making the size of files larger.
   d. Replace one of the direct block numbers in each inode with an additional triple-indirect block number.
      True. The additional triple indirect block number replacement will increase the file size to almost twice.

b) Which of the following adjustments (without any of the modifications in the previous question), will allow files larger than the current approximately 1- gigabyte limit to be stored?
   a. Increasing the size of a block number from 4 bytes to 5 bytes.
      False. The max file size will get reduced as it will store less number of block addresses.
   b. Decreasing the size of a block number from 4 bytes to 3 bytes.
      True. The number of addresses that can be stored in indirect block is increased and hence, will increase the max file size.
   c. Decreasing the size of a block number from 4 bytes to 2 bytes.
      False. Reducing the size of block will increase the number of blocks.

4. EZ Park

4.1 Which of these statements is true about the problems with Ben's design?
   a. There is a race condition in accesses to available[], which may violate one of the correctness specifications when two find _spot ( ) threads run.
      True. The find_spot() threads can be called together by two client cars and concurrency will lead to a race condition and can reserve for a common parking spot. This will violate the correct behavior rules which allows at most one car at a time.
   b. There is a race condition in accesses to available[], which may violate correctness specification A when one find _spot ( ) thread and one relinquish _spot ( ) thread runs.
      False. To access 'i' several clients might get into race conditions. But, the specifications are still followed.
   c. There is a race condition in accesses to numcars, which may violate one of the correctness specifications when more than one thread updates numcars.
      True. This might happen in race condition. Suppose, both the find_spot() and relinquish_spot() threads run at the same time and update variable numcars.

d. There is no race condition as long as the average time between client requests to find a spot is larger than the average processing delay for a request.
False. The clients will keep sending requests concurrently. This will create race condition problems and also the correctness specifications may get violated. So, though the average time between the requests is more than the average processing delay, the race condition may occur due to different reasons.

4.2 Does Alyssa's code solve the problem? Why or why not?
Ans. The code uses locks and entire available[i] (not like before) which helps the threads from encountering any race conditions. And, it will stick to the correctness specifications given.

4.3 Ben can't see any good reason for the release (avail_lock) that Alyssa placed after line 7, so he removes it. Does the program still meet its specifications? Why or why not?
Ans. The program code without release(avail_lock) will work but somewhere the acquire lock will miss matching release. The thread can come across a condition when the available[i] condition is false and doesn't enters the loop. Basically, it doesn't release the lock and the next time it goes to acquire lock, it is unable for car to get a parking spot even if spot is available.

4.4 Does that program meet the specifications?
Ans. As explained in the previous question, it will not release lock in a condition where available[i] is false. Since now, release has been added to the else condition every time lock will release and removing the freeze error, unable to acquire lock for next 'i'.The code will work fine.

4.5 Does Ben's slimmed-down program meet the specifications?
Ans. No. The code is designed in such a way that the race conditions are not faced. The code can work for at least one client can park at one time which is not correct according to the correctness specification's third point where client should get spot.

4.6 If a client invokes the find _spot ( ) RPC when the parking lot is full, how quickly will it get a response, assuming that multiple cars may be making requests?
a. The client will not get a response until at least one car relinquishes a spot.
True. According to the question, no spot is available. So, until and unless any spot is relinquished, the client would not get any response from the server.
b. The client may never get a response even when other cars relinquish their spots.
True. The client may not get the spot as clients keep on asking for spots.

4.7 When Ben adds the timer to his client, he finds some surprises. Which of the following statements is true of Ben's implementation?
a. The server may be running multiple active threads on behalf of the same client car at any given time.
yes. The server has set timer for its client. The server runs multiple thread on behalf of the same client if the time period finishes for the first time and the client retries.

b. The server may assign the same spot to two cars making requests.
No. According to the question, the client requesting for the find_spot() will have a specific amount of time given. So, same spot cannot be allotted to two clients. Whereas, if the timer is also added for relinguish_spot(), then answer to the question might change.

c. numcars may be smaller than the actual number of cars parked in the parking lot.
No. The numcars cannot be different than actual number of cars, if timer is applied only to one thread. The numcars may be smaller if timer is added to relinquish_spot() thread.

d. numcars may be larger than the actual number of cars parked in the parking lot.
True. Numcars may be incremented for every spot allocated, may it be multiple slots assigned to a single client.

4.8 Alyssa thinks that the operating system running Ben's server may be spending a fair amount of time switching between threads when many RPC requests are being processed concurrently. Which of these statements about the work required to perform the switch is correct? Notation: pc 5 program counter; sp 5 stack pointer; pmar 5 page-map address register. Assume that the operating system behaves according to the description in Chapter 5.

a. On any thread switch, the operating system saves the values of the pmar, pc, sp, and several registers.
Incorrect. Since, the address space is same with the threads, the PMAR is unnecessarily saved.

b. On any thread switch, the operating system saves the values of the pc, sp, and several registers.
Correct. The values of counters and pointers are required for the threads to switch.

c. On any thread switch between two relinquish _spot ( ) threads, the operating system saves only the value of the pc, since relinquish _spot ( ) has no return value.
Incorrect. The relinquish_spot() threads uses pointers while executing and also manipulate registers.

d. The number of instructions required to switch from one thread to another is proportional to the number of bytes currently on the thread's stack.
Incorrect. The number of bytes on thread's stack in not dependent on the number of instructions required to switch.

9 Ben's Kernel

9.1 Describe how the supervisor obtains the value of n, which is the identifier for the svc that the calling program has invoked.
When there is a switch from user space to kernel space, the processor switches the use from upmar to kpmar. The information about the n is loaded from page maps that are for high level kernel programs. This has physical address information about the program or function that has invoked svc call.

9.2 How can the current address space be switched?

a. By the kernel writing the kpmar register.
Yes**.** Kpmar register changes the mode to kernel from user.

b. By the kernel writing the upmar register.

No. The upmar register doesn't change the address space.

c. By the processor changing the user-mode bit.
   Yes. The value of this bit gives information about which register is used kpmar or upmar.
d. By the application writing the kpmar or upmar registers
   Yes. The values in the registers can be changed in Kernel mode. User has no access to manipulate them. So, for changing these values, the address space has to be changed.
e. By doyield saving and restoring upmar.
   No. we can't change address space using doyfield in kernel. Upmar register can be updated and stored by doyfield.

9.3 Ben runs the system for a while, watching it print several results, and then halts the processor to examine its state. He finds that it is in the kernel, where it is just about to execute the rti instruction. In which procedure(s) could the user-level thread resume when the kernel executes that rti instruction?

a. In the procedure kernel.
   No. The procedure kernel will not resume user thread. It can't call itself.
b. In the procedure main.
   No.
c. In the procedure yield.
   Yes. The supervisor call 0 will return in the yield to the instruction after svc 0 which might be return instruction.
d. In the procedure doyield .
   No. Doyield doesn't conduct supervisor call and is in the kernel.

9.4 In Ben's design, what mechanisms play a role in enforcing modularity?

a. Separate address spaces because wild writes from one application cannot modify the data of the other application.
   Yes. Separate address spaces will keep the address of different applications and thus, it can prevent any unwanted manipulation among one another.
b. User-mode bit because it disallows user programs to write to upmar and kpmar.
   Yes. The value of user-mode bit will not allow any changes either in the address spaces or other programs.
c. The kernel, because it forces threads to give up the processor.
   No. The scheduler in the processor thread will not serve any thread in between and is not preemptive.
d. The application, because it has few lines of code.
   No. less number of lines has nothing to do with modularity. Though, it can reduce complexity.

9.5 Ben again runs the system for a while, watching it print several results, and then he halts the processor to examine its state. Once again, he finds that it is in the kernel, where it is just about to execute the rti instruction. In which procedure(s) could the user-level thread resume after the kernel executes the rti instruction?

a. In the procedure dointerrupt.

No.

b. In the procedure supervisorcall
   No. Supervisor call is in kernel. Can't call itself.
c. In the procedure main.
   Yes. Clock interrupt might interrupt while the main() is executed.
d. In the procedure yield.
   Yes. The supervisor call 0 will return in the yield to the instruction after svc 0 which might be return instruction.
e. In the procedure doyield.
   No. The doyield is in kernel where the svc0 will not be executed.

9.6 In Ben's second design, what mechanisms play a role in enforcing modularity?

a. Separate address spaces because wild writes from one application cannot modify the data of the other application.
   Yes. Separate address spaces will keep applications separate and doesn't allow one application to write in some other application's address space.
b. User-mode bit because it disallows user programs to write to upmar and kpmar.
   Yes. The value of user mode- bit will not allow user programs to change without permissions.
c. The timer chip because it, in conjunction with the kernel, forces threads to give up the processor.
   Yes. Preemption technique will temporarily serve a thread and prevent other threads from using processor.
d. The application because it has few lines of code.
   No. The reduced number of lines will not help enforcing modularity, it just helps to reduce the complexity.

9.7 What values can the applications print (don't worry about overflows)?

a. Some odd number.
   Yes. After 100, the thread gets interrupted and thread 2 doubles t, t is 1 at first. When the initial thread will continue after interrupt, it will end up adding 1 and 2. This will give final value as 3.
b. Some even number other than a power of two.
   Yes. If t is even say 2, then thread 1 gets interrupted and other thread will double the value of t. And, when the first thread will come back it will add 2 to 4 and the last value will be 6.
c. Some power of two.
   Yes. Thread 1 is not interrupted by any other thread.
d. 1
   No. No thread will print the value without making it twice.

9.8 Now, what values can the applications print (don't worry about overflows)?

a. Some odd number.

No. Code region from 100-112 will run atomically i.e without interrupt. A thread will always double the value of variable t and will start at 1 which produces only the power of 2.

b. Some even number other than a power of two.
   No. The output variable will be of power of two as explained in previous answer.
c. Some power of two.
   Yes. Thread 1 is not interrupted by any other thread. And, the values can be of power two.
d. 1
   No. No thread will print the value without making it twice.

9.9 Can a second thread enter the region from virtual addresses 100 through 112 while the first thread is in it (i.e., the first thread's upc contains a value in the range 100 through 112)?

a. Yes, because while the first thread is in the region, an interrupt may cause the processor to switch to the second thread and the second thread might enter the region.
   Yes. The thread lies in 100-112 region and gets interrupted. The program counter will point 100 and second thread runs and enter the program region of 100-112.
b. Yes, because the processor doesn't execute the first three lines of code in dointerrupt atomically.
   No. The dointerrupt will be executed in kernel mode and the interrupts will not be entertained by the kernel and are ignored. Dointerrupt will not serve any thread in between(atomically).
c. Yes, because the processor doesn't execute doyield atomically.
   No. The kernel will complete the doyield atomically.
d. Yes, because main calls yield.
   No. The main can call yield only after the first program is complete(100-112).

9.10 What are some possible outcomes if a thread executes this restartable atomic region and variables a, b, and x are not shared?

a. a = 2 and b = 1
   yes. The value of a and b gets interchanged. Assuming, no interrupt is coming in between.
b. a = 1 and b = 2
   No. The value of a is 2 always at the end of the execution. As, the value of a and b are being swapped.
c. a = 2 and b = 2
   Yes. If the code is interrupted before 108 and after 104, we can find the value of a as 2. And, this new value is assigned to x on restart at 100. Now, at 108 the b is assigned the value 2.
d. a = 1 and b = 1
   No. a is always set to 2 at the end of the code. And, there values are being swapped in the code.