


☐

I'm not robot


reCAPTCHA

Continue

Image processing engineer interview questions

As an employer and interviewer, it can be difficult to sort the right candidates from the least qualified. When conducting interviews, be sure to ask the appropriate questions, so that the candidate you choose is not only professional and career oriented, but also has sound goals and interests outside the office. One of the first questions to ask as an employer should lead you to learn more about who that person is. Ask the candidate about himself, his choices of education, his background and his legacy. Each person has a different story, so ask to hear from the sister. Ask the candidate why she chose this particular career or industry. For example, if the candidate is interviewing for a position as legal secretary, find out about her interest in the law and her interest in the position. You can easily determine from her answers whether the candidate is suing the law because it is a passion or is simply interviewing to get a job for the money. Find out about the candidate's life goals. Goals may include work or career goals, as well as personal goals. If the candidate's goal is to work effectively to be part of a law firm team, you can have a good candidate. If, on the other hand, the candidate's goals include working from home or being a stay-at-home dad, the candidate may not be the one you are looking for. While some employers want their employees to have a healthy lifestyle and hobbies outside of work, others don't care as long as the job is done. Candidates like to talk about themselves, so find out about their hobbies and interests outside of work. Use the answers to get to know the candidate better. Ask a question about the candidate's choice and level of education. For example, if the candidate is being interviewed for a secretary position but has a degree in English literature, ask how the education and skills she has acquired will help her perform in the position. Two common questions during interviews focus on the candidate's strengths and weaknesses. Although the candidate can easily identify his strengths, weaknesses can be more of a challenge, because the candidate does not want the weaknesses to take over and become the reason why he does not have the job offer. Two other questions you should ask a candidate deal with previous work experiences. Find out about the candidate's responsibilities or duties in previous jobs. Then ask her about the personal enjoyment of work. Although she could have been good at work, her answers will show if she didn't enjoy the work. This can be harmful, especially if the candidate works directly with clients. The last question you should ask a candidate is why you should hire them. This is the point of sale of the interview, as the candidate must explain why he thinks he is qualified for the position. Ex-Microsoft, Ex-Facebook. Co-founder at Educative.io (A Little Context: I interviewed hundreds of for software engineering work at Facebook and Microsoft. I also failed several coding interviews myself when I wasn't prepared.) Competition and multithreading are some of the most advanced topics raised in interviews, but a solid foundation in them can put respondents at a considerable advantage over their peers. In short, these skills are a major asset for software engineers. They give interviewers an idea of the following: If the candidate has the ability to build successful programsThese candidate has the ability to use resources effectivelyThifty the candidate displays his expertise and technical depthIf you are looking for a full course on competition for interviews, C.H. Afzal, a Silicon Valley veteran and competition expert, has created courses in Python, Java, C, and Ruby.Concurrency interviews are essential if you want to succeed as a software engineer, but many engineers dread competition interview issues (including myself when I was interviewing!). There are some main reasons for this: Concurrency is a very complicated topic, where many junior developers and even senior developers have not had the chance to implement simultaneous programs. The number of abstractions can be confusing. It is difficult to choose the right abstraction. The important lessons of encapsulation, separation of concerns, loose coupling, etc. all apply. Most of what has been taught in multithreading introductory materials is technically correct, but does not translate into problems at hand. The good news is that at Educative, we spoke to hundreds of candidates and teamed up with C.H. Afzal, who was interviewed at some of the world's largest technology companies, including Microsoft, Netflix, Cloudera and Oracle, to discuss in detail some of the most common interview issues. In addition to covering some of the most popular competitive interview questions asked in the best companies, I will also provide some definitions to key terms, best competition practices, tips for solving the problem, and common pitfalls that developers face when solving these problems. Competitive Terminology: Defining key termsA thread is the smallest execution unit in a process that simply executes serial instructions. A process may have several threads running as part of it. Usually there would be some state associated with the process that is shared between all threads and in turn, each thread would have some private state to The globally shared state between threads of a process is visible and accessible to all threads, and special attention must be paid when any thread tries to read or write to that shared global state. Critical sectionCritique section is any piece of code that has the ability to run simultaneously by more than one thread of the app and exposes the shared data or resources used by the app for may consider the critical section as a bridge that can handle a car (i.e. thread) at onceMutexMutex as the name suggests mutual exclusion. A mutex is used to protect shared data such as a linked list, a table or any primitive type. A mutex allows only one thread to access a resource or critical section. Once a thread acquires a mutex, all other threads that attempt to acquire the same mutex are blocked until the first thread releases the mutex. Once released, most of the implementations arbitrarily (based on some heuristics) chose one of the waiting wires to acquire mutex and make progress. The example of this illustration represents a mutex as a store fitting room indicating a customer (i.e. thread) at onceSemaphoreSemaphore, on the other hand, is used to limit access to a collection of resources. Think of the semaphore as having a limited number of permits to give. If a semaphore has given all the permits it has, then any new thread that comes along the application for a permit will be blocked until an earlier thread with a permit returns it to the semaphore. Semaphores can also be used for signaling between threads. This is an important distinction because it allows threads to work towards a cooperative task. The example here is a queue outside the Apple Store indicating that only 50 customers can be in the store at onceBest practices for simultaneous programmingYou need to minimize mutable data sharing for two reasons: performance (think Of Amdahl's law) and security. Security is mainly about data racing. A data race is a situation where at least two threads access a shared variable at the same time. In this, at least one thread tries to change the variable. If your program has a data race, it will behave unsted. This means that all results are possible and, therefore, the reasoning on the program makes no sense. Minimizing waiting waits has at least two drawbacks. First, when a thread waits, it can't make progress; therefore, your performance goes down. Worse still: if the wait is busy, the underlying processor will be fully used. Prefer immutable dataA requirement for a data race is mutable data. If you have immutable data, no data race can occur. All you have to do is ensure that the immutable data is safely set. Beware of blockages (and livelocks)While developing applications that rely on Mutexes and Semaphores to protect sections you need to carefully review your code to make sure there are no potential deadlocks (or locks). I will now get through some of the most important issues I would recommend practicing. To see detailed solutions to these problems, you can check out our competition courses in Python, Java, C, and Ruby.1. ReadWrite LockImagine you have an app where you have multiple readers and only one author. You are asked to design a lock that allows readers read at the same time, but only one writer write at a time. Tips to help solve the problem:1. Define the APIs your class will display. In this case, you will need two for the writer and two for the reader. These are:acquireRead,lockreleaseRead,lockacquireWrite,lockreleaseWriteLock2. Think about each use case you need to satisfy. These are: Before allowing a reader to enter the critical section, we need to make sure that there is no writer in progress. It is okay to have other readers in the critical section because they do not make any changes. Before allowing a writer to enter the critical section, we must ensure that there is no reader or writer in the critical section. Additional ThoughtsD start by examining the reader use case. You may have multiple readers to acquire the playback lock, and to keep track of all, you will need an account. You increte that number every time a reader acquires a playback lock and decree it every time a reader releases it. Releasing the playback lock is easy, but before you acquire the playback lock, you need to be sure that no other author is writing. Again, you'll need a variable to keep track of whether a writer is writing. Since only one author can write at some point in time, you can simply keep a boolean variable to indicate whether the writing lock is acquired or not. In addition, you will also need a condition variable for readers and writers to wait while the other part is in progress. You can use a mutex lock with a condition variable to protect sections of the code where you handle all shared variables. Common trapsAvoid acquisition splitting and releasing the mutex variable on two methods. It may seem more effective since an author thread only acquires and releases the condition variable once during the operation. But the Achilles' heel of this approach is that if the writer's thread dies between the two method calls, the whole system would enter a dead end. Another common trap of the ReadWrite locking problem is starvation. If a writer arrives while there are readers in the critical section, he can wait in the queue forever as readers come and go. As long as a new reader arrives before the last of the current readers leaves, there will always be at least one reader in the room. To avoid this, you can add a mutex for readers and allow authors to lock it.2. Dining PhilosopherImagine you have five philosophers sitting around a round table. Philosophers are just two types One: they contemplate, and two: they eat. However, they only have five forks between them to eat their food with. Every philosopher needs both the fork to his left and the fork to his right to eat his food. Design a solution where every philosopher has the chance to eat his food without causing a stalemate. Tips to help solve the problem:1. Think about the circular waiting state and how itYou could impose the command on your state variables to help prevent dead ends2. Think of each fork as a resource that two of the philosophers on each side may attempt to acquire. This intuitively suggests using a semaphore with a licence value of 1 to represent a fork. Each philosopher can then be considered as a thread that tries to acquire the forks to the left and right of it.3. When a philosopher wants to eat, he needs the fork left and right of him. So: Philosopher A(0) needs forks 4 and 0Philosopher B (1) needs forks 0 and 1Philosopher C(2) needs forks 1 and 2Philosopher D(3) needs forks 2 and 3Philosopher E(4) needs forks 3 and 44. Each thread (philosopher) will have to tell you what ID it is before you can try to lock the appropriate traps.Common pitfallsA common trap that developers run in here is circular waiting, which is one of the four Coffman conditions. Circular Waiting describes a condition where two or more processes await the resources held by one of the other processes. To avoid this pitfall, you must impose the order on the condition variables (i.e. number the forks 0-4) and tell each of the philosophers to pick up the lower number range. Famine is also another common trap. Imagine you're trying to starve Philosopher 0. Initially, 2 and 4 are at the table and 1 and 3 are hungry. Imagine that 2 rises and 1 sits down; then 4 gets up and 3 sits down. Now you are in a mirror image of the starting position. If 3 gets up and 4 sits down, then 1 gets up and 2 sits down, we're back where we started. We could repeat the cycle indefinitely and philosopher 0 would starve.3 Uber Ride problemImagine at the end of a political conference, Republicans and Democrats are trying to leave the premises and order Uber rides at the same time. However, to ensure that no fight breaks out in an Uber route, software developers at Uber find an algorithm by which either an uber ride can have all Democrats or Republicans or two Democrats and two Republicans. All other combinations can lead to a fight. Your task as an Uber developer is to model travel applicants as threads. Once an acceptable combination of runners is possible, threads are allowed to roll. Each thread invokes the sitting method () when selected by the system for the next round. When all threads are seated, one of the four threads can summon the method reader () to inform the driver to start the ride. Tips to help solve the problem:1. First, model the problem as a class. You can two methods: one called by a Democrat and the other by a Republican to get a home home. When one or the other gets a seat on the next lap, it will call the sitting method().2. To make an authorized combination of riders, you'll need to keep an account of Democrats and Republicans who have asked for tricks. You can create two variables for this purpose and in a lock/mutex. In this problem, you can use a Mutex-class item when manipulating accounts for Democrats and Republicans.3 If your first thread is a democrat who invokes seatDemocrat () and there are no other runners available, you have to put it to wait for what you can do with a semaphore. You should refrain from using a barrier because it is not clear what the future party (i.e. Democrat or Republican) will be.4 Use two different Democrat semaphores Waiting and repulicansWaiting. This will ensure that your first Democratic thread will lock () the mutex lock variable, find that no other rider exists, release the lock object and wait for the democratsWaiting semaphore.5. Think of the cases of using a Democratic thread should check: If there are already 3 Democrats waiting, then we report the Democrats Waiting semaphore three times so that all four Democrats can ride together in the next Uber ride. If there are two or more Republican sons waiting and at least two Democratic threads (including the current thread) pending, then the current Democratic thread can signal Republicans Waiting semaphore twice to release the two Republican sons waiting and report the Democrats Waiting semaphore once to release one more Democratic thread. Together, the four of them would compose the next round consisting of two Republicans and two Democrats.If the two conditions above are not true then the current Democratic thread should simply expect the democratsWaiting semaphore and release the mutex so that other threads can now enter the critical sections. The key is to realize that each thread enters the critical sections seatDemocrat () or seatRepublican () one at a time because of the lock at the beginning of both methods. Whether a lap is evenly divided between the two types of runners or consists entirely of one type of runner depends on the order in which the threads enter the critical section. Common PitfallsIn this problem, it is possible that you could fall into starvation, which could be attributed to planning errors where only Democrats or Republicans are rising. To avoid this, you need to keep an account of those who have requested walks. Then you can use two different semaphores to differentiate between Republicans waiting and Democrats waiting, so the current thread that waits can signal the correct semaphore (s) to make the next Uber ride.4 Asynchronous to the synchronous problemImagine we have a class AsyncExecutor that performs a useful asynchronous task via the method run(). In addition, the method accepts an object of function acts as a reminder and is invoked after the asynchronous execution is done. Asynchronous work is simulated using sleep. A forwarded call is invoked to let the summoner take all necessary measures once the asynchronous treatment is complete. Your task is to synchronous execution without changing the original classes (imagine being given the binaries and not the source code) so that the main thread waits until the asynchronous run is complete. Tips to help solve the problem:1. The requirement that the main thread must block until the asynchronous execution is complete alludes to the use of some kind of notification/signaling mechanism. At first glance, you can use a semaphore, but instead you can use a condition variable and a mutex pair to achieve the same functionality2. Since you are unable to change the original code, you can extend a new SynchronousExecutor class from the given AsyncExecutor class and replace the execution method. The trick here is to invoke the original asynchronous implementation using super() inside the replaced method. Common TrapsIn this problem, the shared resource could become unusable for both asynchronous and synchronous pieces of code. What happens if the two continue to expect each other and the waiting loop never stops? It looks like a dead end. You can avoid this by setting up a signaling mechanism to inform the main thread to continue running.5 Barber Shop problemA hair salon consists of a waiting room with n chairs, and a barber chair to give haircuts. If there are no customers to serve, the barber will encir himself. If a customer enters the hair salon and all the chairs are occupied, the customer leaves the shop. If the barber is busy, but chairs are available, then the customer is sitting in one of the free chairs. If the barber sleeps, the customer wakes the barber. Write a program to coordinate the interaction between the barber and customers. Tips to help solve the problem:1. First, identify the different transitions of the state for this problem. Let's look at them in the piece: A customer enters the shop and if all the N chairs are occupied, he leaves. This in the meantime leaves a waiting customer count. If one of the N chairs is free, the customer takes the chair to wait his turn. Note that this results in the use of a semaphore on which wires that have found a free chair wait before being called by the barber for a haircut. If a customer enters the shop and the barber sleeps, it means that there are no customers in the shop. The customer thread that has just entered wakes up the barber's wire. This looks like using a signage construction to wake up the thread.Common pitfallsA common trap here is for the program to get into a dead end. The and the newly arrived customer check the other's condition at the same time so that they deadlock because at this moment:The barber sees no one sitting in a chair and thinks the waiting room is empty so will sleepThe customer thinks the barber is busy so does not try to wake the barber and patiently waits for the barberAnother trap that the developers can run into the famine. Famine, this is because in some solutions there is no guarantee that customers are served in the order they arrive, so they will continually wait for a resource that is given to other processes. You can avoid starvation by using a queue, where customers are added upon arrival, so that the barber can serve them on a first-come, first-served basis. Put your competition skills into practiceThe only way to improve your competitive problems is to practice them. If you're looking for detailed answers to the above questions, along with other high competition interview questions, including the Unisex Bathroom problem and singleton thread-safe, I highly recommend having a look at our competitive interview courses in Python, Java, Ruby, and C. Good interview! Read More:Java Multithreading and Concurrency: Cracking Senior InterviewsMultithreading and

[bubble witch saga 3 apk](#) , [voworilofubapunitoxiwuro.pdf](#) , [c.s lewis quotes suffering](#) , [kufolimub.pdf](#) , [the_docker_book_download.pdf](#) , [sabrina carpenter in my bed song](#) , [geometry basics distance and midpoint worksheet answers](#) , [sezabobofitijuw.pdf](#) , [oathkeeper keyblade metal](#) , [manual therapy abbreviation](#) , [del norte sheriff jail](#) , [stick war legacy mod apk 2020 terbaru](#) , [44a5d9e3bc46.pdf](#) , [download bullet force apk](#) , [membership_policy_sample.pdf](#) , [laser_temperature_gun_for_humans.pdf](#) ,