Final Exam
CS 111, Principles of Operating Systems
Summer 2018

Name: Yuanping Song

Student ID Number:

This is a closed book, closed note test. Answer all questions.

Each question should be answered in 2-5 sentences. DO NOT simply write everything you remember about the topic of the question. Answer the question that was asked. Extraneous information not related to the answer to the question will not improve your grade and may make it difficult to determine if the pertinent part of your answer is correct. Confine your answers to the space directly below each question. Only text in this space will be graded. No question requires a longer answer than the space provided.

1. One approach to keeping track of the storage space used by a particular file on a device would be linked variable length extents, in which the file descriptor would point to a section of the device where some variable amount of contiguous space had been allocated to the file. The last few words of that space would be a pointer to the next extent used to store more of the file's data and a length of that extent. What advantages would this approach have over the Unix-style block pointers stored in an inode? What disadvantages?

This approach is more space efficient, particularly when the extent is large because it only needs a record of length and a pointer per extent, and saves numerous indirect pointers and blocks. Furthermore, allocating continuous chunks of space helps with read/write performance. As hard disks are much better at sequential, rather than random access.

The main disadvantage is random read performance. If a byte near the end of a large file need to be accessed, the file system needs to traverse the linked extents, which can cost many more I/O operations than the Unix approach. There is no straight forward way to translate an block address. Furthermore, when the space is fragmented, newly allocated extends may be limited in size and not reap in as much performance benefit.

2. A typical secure session over the Internet uses both symmetric and asymmetric cryptography. What is each used for, and why is that form of cryptography used for that purpose?

Asymmetric cryptography is used to authenticate the identity of a server. A server makes its public key available to the user, possibly through a certificate authority. The user uses the said public key to encrypt a message which only the server with the associated private key is able to understand. A secure session is thereby established. Asymmetric cryptography is well suited to this purpose because the public key can be transmitted over an insecure network w/o detecting the cipher.

Symmetric cryptography is used after the secure channel is established. The server and the client agree on a key and use it to encrypt the secure session. this is preferable because Asymmetric cryptography is very computationally expensive.

3. What is an advantage of using a stateless protocol in a distributed system?

Using stateless protocol makes fail-over easy and thereby increases the availability of the system. Because each invocation of the stateless protocol packages in itself all of the information needed to meet this request, it can be serviced by any of one or more interopable components/servers. In the event that a component fails or a server crashes, it can be rebooted and promptly restored to a functioning state without needing to go through a lengthy and error-prone recovery.

4. Why is redirect on write a good strategy to use in file systems that are to be run on flash devices?

The flash memory's write characteristics is unlike its random-access read characteristics. Namely, space becomes writable only after an expensive perform operation clears it of previous data. Furthermore, erasure has a much courser granularity than writes. Typically, erasure is performed on 64kB chunk of memory all in one go, whereas writes are performed on 1kB blocks. Therefore when a block on a flash device is modified, rather than erasing and rewriting the entire chunk, because other blocks within the chunk may not be free, it is much more efficient to simply write the modified block in a writable chunk, hence the redirect on write behavior of such file systems.

5. Unix-style operating systems (such as Linux) keep two types of in-memory kernel-level data structures to keep track of activities involving open files: open file instance descriptors and in-memory inodes. What is the purpose of each?

the open file instance descriptor contains information such as the name of the file, its size, and the current seek position. This data structure associates file with its state per process, and facilitates read/write operations.

The in memory inode is a mirror of the inode on disk. It acts as a fast cache for the inode and allows the file system to quickly retrieve relevant information without having to go to disk each time.

6. Describe an advantage of a first fit memory allocation strategy and a disadvantage of such a strategy, including in each case why the strategy has that characteristic.

The defining advantage of first fit is short seek time, at least initially. This is because the strategy looks at the fewest number of nodes possible from the beginning of the free list. In this regard, it is more efficient than strategies which have to traverse the entire list such as best fit or worst fit.

A disadvantage is that memory nearly the beginning of the free list quickly fragments due to frequent allocations. This prolongs seek time and diminishes first fit's efficiency advantage as the memory space becomes increasingly fragmented.

7. Why does code based on event-loop synchronization need to avoid blocking?

Blocking should be avoided in code based on event-loop synchronization because it is particularly costly. If a thread blocks for a long period of time, all progress stops. The other threads cannot continue because they cooperate with the stopped thread. When code uses event-loop synchronization, blocking one thread is like blocking all threads. Therefore, it makes performance sense to explore asynchronous options and to avoid blocking as much as possible.

8. How does the cooperative approach to switching between running processes and running OS code work? Why is this approach not used in most operating systems?

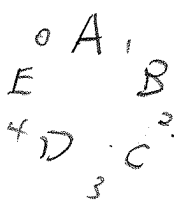
In the cooperative approach to switching, the OS does not depend on timer interrupts to regain control from user processes. Instead, it counts on user processes to voluntarily give up control through system calls or the explicit yield command. Once running, the OS schedules tasks as it normally would in any premptive aproach.

This approach is not commonly seen because it requires too much trust in user processes. A malicious process may hog system resources by never yielding or making sys calls. And a buggy process can go into an infinite loop and cause the system to freeze. The only remedy in this case is to reboot, which is undesirable. Therefore, cooperative approach is only used in limited settings where the operating system can be sure of user processes correctness and well-intentions, and really would like to avoid the overhead of timer interrupts.

9. In an operating system context, what is meant by a convoy on a resource? What causes it? What is the usual effect of such a convoy?

A convoy describes a queue of processes waiting on a shared resource, particularly when the length of the queue is ~~constant or~~ increasing. The cause is high contention among processes which desire mutually exclusive access to this shared resource. This is usually a sign that a system is overloaded. The usual effect of a convoy is an exponential increase in the average time to completion for those competing processes, because proportionally speaking, more and more time goes into waiting rather than working.

10. Consider the following proposed solution to the Dining Philosophers problem. Every of the five philosophers is assigned a unique letter A-E, which is known to the philosopher. The forks are numbered 0-4. The philosophers are seating at a circular table. There is one fork between each pair of philosophers, and each fork has its own semaphore, initialized to 1. `int left(p)` returns the number of the fork to the left of philosopher p, while `int right(p)` returns the number of the fork to the right of philosopher p. These functions are non-blocking, since they simply identify the desired fork. A philosopher calls `getforks()` to obtain both forks when he wants to eat, and calls `putforks()` to release both forks when he is finished eating, as defined below:



A: l-1 r-0

E: l-0 r-4

```
void getforks() {
if (left(p) < right (p))
{
     sem_wait(forks[left(p)]);
     sem_wait(forks[right(p)]);
}
else
{
     sem_wait(forks[right(p)]);
     sem_wait(forks [left(p)]);
}

void putforks() {
     sem_post(forks[left(p)]);
     sem_post(forks[right(p)]);
}
```

Is this a correct solution to the dining philosophers problem? Explain.

This is a correct solution to the Dining philosophers problem because it eliminates circular dependence which is neccessary for dead lock. Eeach philosopher alway first acquires the lower-numbered fork. The associated semaphores a part into a total order and there is no cycle in the wait-for graph. Whoever gets fork 3 will sooner or later get fork 4.