操作系统作业

姓名，学号

1. Explain the difference between internal and external fragmentation.

   Answer:

   a. Internal fragmentation is the area in a region or a page that is not used by the job occupying that region or page. This space is unavailable for use by the system until that job is finished and the page or region is released.

   b. External fragmentation is unused space between allocated regions of memory. Typically external fragmentation results in memory regions that are too small to satisfy a memory request, but if we were to combine all the regions of external fragmentation, we would have enough memory to satisfy a memory request.

2. Most systems allow a program to allocate more memory to its address space during execution. Allocation of data in the heap segments of programs is an example of such allocated memory. What is required to support dynamic memory allocation in the following schemes?

   a. Contiguous memory allocation
   b. Pure segmentation
   c. Pure paging

   Answer:

   a. contiguous-memory allocation: might require relocation of the entire program since there is not enough space for the program to grow its allocated memory space.

   b. pure segmentation: might also require relocation of the segment that needs to be extended since there is not enough space for the segment to grow its allocated memory space.

   c. pure paging: incremental allocation of new pages is possible in this scheme without requiring relocation of the program's address space.

3. Compare the memory organization schemes of contiguous memory allocation, pure segmentation, and pure paging with respect to the following issues:

   a. External fragmentation
   b. Internal fragmentation
   c. Ability to share code across processes

   Answer:

   The contiguous memory allocation scheme suffers from external fragmentation as address spaces are allocated contiguously and holes develop as old processes die and new processes are initiated. It also does not allow processes to share code, since a process's virtual memory segment is not broken into noncontiguous finegrained segments. Pure segmentation also suffers fromexternal fragmentation as a segment of a process is laid out contiguously in physical memory and fragmentation would occur as segments of dead processes are replaced by segments of new processes. Segmentation, however, enables processes to share code; for instance, two different processes could share a code segment but have distinct data segments. Pure paging does not suffer from external fragmentation, but instead suffers frominternal fragmentation. Processes are allocated in page granularity and if a page is not completely utilized, it results in internal fragmentation and a corresponding wastage of space. Paging also enables processes to share code at the granularity of pages.

4. Assuming a 1-KB page size, what are the page numbers and offsets forthe following address references (provided as decimal numbers):
   a. 3085
   b. 42095
   c. 215201
   d. 650000
   e. 2000001

   Answer:

   a. page = 3; offset = 13
   b. page = 41; offset = 111
   c. page = 210; offset = 161
   d. page = 634; offset = 784
   e. page = 1953; offset = 129

5. Consider a logical address space of 256 pages with a 4-KB page size, mapped onto a physical memory of 64 frames.
   a. How many bits are required in the logical address?
   b. How many bits are required in the physical address?

   Answer:

   a. 12 + 8 = 20 bits.
   b. 12 + 6 = 18 bits.

6. Consider a paging system with the page table stored in memory.
   a. If a memory reference takes 50 nanoseconds, how long does a paged memory reference take?
   b. If we add TLBs, and 75 percent of all page-table references are found in the TLBs, what is the effective memory reference time? (Assume that finding a page-table entry in the TLBs takes 2 nanoseconds, if the entry is present.)

   Answer:

   a. 100 nanoseconds: 50 nanoseconds to access the page table and 50 nanoseconds to access the word in memory.
   b. Effective access time = 0.75 × (50 nanoseconds) + 0.25 × (100 nanoseconds) = 62.5 nanoseconds.

7. Assume a program has just referenced an address in virtual memory. Describe a scenario how each of the following can occur: (If a scenario cannot occur, explain why.)

- TLB miss with no page fault
- TLB miss and page fault
- TLB hit and no page fault
- TLB hit and page fault

Answer:

- TLB miss with no page fault: page has been brought into memory, but has been removed from the TLB
- TLB miss and page fault: page fault has occurred
- TLB hit and no page fault: page is in memory and in the TLB. Most likely a recent reference
- TLB hit and page fault cannot occur. The TLB is a cache of the page table. If an entry is not in the page table, it will not be in the TLB.

8. What is the copy-on-write feature, and under what circumstances is it beneficial? What hardware support is required to implement this feature?

Answer:

When two processes are accessing the same set of program values (for instance, the code segment of the source binary), then it is useful to map the corresponding pages into the virtual address spaces of the two programs in a write-protected manner. When a write does indeed take place, then a copy must be made to allow the two programs to individually access the different copies without interfering with each other. The hardware support required to implement is simply the following: on each memory access, the page table needs to be consulted to check whether the page is write protected. If it is indeed write protected, a trap would occur and the operating system could resolve the issue.

9. Assume we have a demand-paged memory. The page table is held in registers. It takes 8 milliseconds to service a page fault if an empty page is available or the replaced page is not modified, and 20 milliseconds if the replaced page is modified. Memory access time is 100 nanoseconds. Assume that the page to be replaced is modified 70 percent of the time. What is the maximum acceptable page-fault rate for an effective access time of no more than 200 nanoseconds?

Answer:

$$
\begin{aligned}
0.2\ \mu\text{sec} &= (1-P)\times 0.1\ \mu\text{sec} + (0.3P)\times 8\ \text{millisec} + (0.7P)\times 20\ \text{millisec} \\
0.1 &= -0.1P + 2400\ P + 14000\ P \\
0.1 &\simeq 16{,}400\ P \\
P &\simeq 0.000006
\end{aligned}
$$

10. Consider the following page reference string: 7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0 , 1. Assuming demand paging with three frames, how many page faults would occur for the following replacement algorithms?

    - LRU replacement
    - FIFO replacement
    - Optimal replacement

    Answer:

    - 18
    - 17
    - 13

11. Explain why SSDs often use a FCFS disk scheduling algorithm.

    Answer:

    Because SSDs do not have moving parts and therefore performance is insensitive to issues such as seek time and rotational latency. Therefore, a simple FCFS policy will suffice.

12. Suppose that a disk drive has 5,000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 2150, and the previous request was at cylinder 1805. The queue of pending requests, in FIFO order, is: 2069, 1212, 2296, 2800, 544, 1618, 356, 1523, 4965, 3681 Starting from the current head position, what is the total distance (in cylinders) that the disk armmoves to satisfy all the pending requests for each of the following disk-scheduling algorithms?

    a. FCFS
    b. SSTF
    c. SCAN
    d. LOOK
    e. C-SCAN
    f. C-LOOK

    Answer:

    a. The FCFS schedule is 2150, 2069, 1212, 2296, 2800, 544, 1618, 356, 1523, 4965, 3681. The total seek distance is 13,011.

b. The SSTF schedule is 2150, 2069, 2296, 2800, 3681, 4965, 1618, 1523, 1212, 544, 356. The total seek distance is 7586.

c. The SCAN schedule is 2150, 2296, 2800, 3681, 4965, 2069, 1618, 1523, 1212, 544, 356. The total seek distance is 7492.

d. The LOOK schedule is 2150, 2296, 2800, 3681, 4965, 2069, 1618, 1523, 1212, 544, 356. The total seek distance is 7424.

e. The C-SCAN schedule is 2150, 2296, 2800, 3681, 4965, 356, 544, 1212, 1523, 1618, 2069. The total seek distance is 9917.

f. The C-LOOK schedule is 2150, 2296, 2800, 3681, 4965, 356, 544, 1212, 1523, 1618, 2069. The total seek distance is 9137.

13. Requests are not usually uniformly distributed. For example, we can expect a cylinder containing the file-system metadata to be accessed more frequently than a cylinder containing only files. Suppose you know that 50 percent of the requests are for a small, fixed number of cylinders.

a. Would any of the scheduling algorithms discussed in this chapter be particularly good for this case? Explain your answer.

b. Propose a disk-scheduling algorithmthat gives even better performance by taking advantage of this “hot spot” on the disk.

Answer:

a. SSTF would take greatest advantage of the situation. FCFS could cause unnecessary head movement if references to the “highdemand” cylinders were interspersed with references to cylinders far away.

b. Here are some ideas. Place the hot data near the middle of the disk. Modify SSTF to prevent starvation. Add the policy that if the disk becomes idle for more than, say, 50 ms, the operating system generates an *anticipatory seek* to the hot region, since the next request is more likely to be there.

14. Consider a RAID Level 5 organization comprising five disks, with the parity for sets of four blocks on four disks stored on the fifth disk. How many blocks are accessed in order to perform the following?

a. A write of one block of data

b. A write of seven continuous blocks of data

Answer:

a. A write of one block of data requires the following: read of the parity block, read of the old data stored in the target block, computation of the new parity based on the differences between the new and old contents of the target block, and write of the parity block and the target block.

b. Assume that the seven contiguous blocks begin at a four-block boundary. A write of seven contiguous blocks of data could be performed by writing the seven contiguous blocks, writing the parity block of the first four blocks, reading the eight block, computing the parity for the next set of four blocks and writing the corresponding parity block onto disk.

15. The open-file table is used to maintain information about files that are currently open. Should the operating system maintain a separate table for each user or just maintain one table that contains references to files that are being accessed by all users at the current time? If the same file is being accessed by two different programs or users, should there be separate entries in the open file table?

Answer:

By keeping a central open-file table, the operating system can perform the following operation that would be infeasible otherwise. Consider a file that is currently being accessed by one or more processes. If the file is deleted, then it should not be removed from the disk until all processes accessing the file have closed it. This check can be performed only if there is centralized accounting of number of processes accessing the file. On the other hand, if two processes are accessing the file, then two separate states need to be maintained to keep track of the current location of which parts of the file are being accessed by the two processes. This requires the operating system to maintain separate entries for the two processes.

16. If the operating system were to know that a certain application is going to access the file data in a sequential manner, how could it exploit this information to improve performance?

Answer:

When a block is accessed, the file system could prefetch the subsequent blocks in anticipation of future requests to these blocks. This prefetching optimizationwould reduce thewaiting time experienced by the process for future requests.

17. What are the advantages of the variation of linked allocation that uses a FAT to chain together the blocks of a file?

Answer:

The advantage is that while accessing a block that is stored at themiddle of a file, its location can be determined by chasing the pointers stored in the FAT as opposed to accessing all of the individual blocks of the file in a sequentialmanner to find the pointer to the target block. Typically, most of the FAT can be cached inmemory and therefore the pointers can be determined with just memory accesses instead of having to access the disk blocks.

18. Consider a system where free space is kept in a free-space list.

   a. Suppose that the pointer to the free-space list is lost. Can the system reconstruct the free-space list? Explain your answer.

   b. Consider a file system similar to the one used by UNIX with indexed allocation. How many disk I/O operations might be required to read the contents of a small local file at */a/b/c*? Assume that none of the disk blocks is currently being cached and assume that all inodes are in memory.

   c. Suggest a scheme to ensure that the pointer is never lost as a result of memory failure.

Answer:

a. In order to reconstruct the free list, it would be necessary to perform “garbage collection.” This would entail searching the entire directory structure to determine which pages are already allocated to jobs. Those remaining unallocated pages could be relinked as the free-space list.

b. Reading the contents of the small local file */a/b/c* involves 4 separate disk operations: (1) Reading in the disk block containing the root directory /, (2) & (3) reading in the disk block containing the directories *b* and *c*, and reading in the disk block containing the file *c*.
c. The free-space list pointer could be stored on the disk, perhaps in several places

19. Consider a file system that uses inodes to represent files. Disk blocks are 8-KB in size and a pointer to a disk block requires 4 bytes. This file system has 12 direct disk blocks, plus single, double, and triple indirect disk blocks. What is the maximum size of a file that can be stored in this file system?

Answer:

(12 * 8 /KB/) + (2048 * 8 /KB) + (2048 * 2048 * 8 /KB/) + (2048 * 2048 * 2048 * 8 /KB) = 64 terabytes

20. Explain why logging metadata updates ensures recovery of a file system after a file system crash.

Answer:

For a file system to be recoverable after a crash, it must be consistent or must be able to be made consistent. Therefore, we have to prove that logging metadata updates keeps the file system in a consistent or able-to-be-consistent state. For a file system to become inconsistent, the metadata must be written incompletely or in the wrong order to the file system data structures.Withmetadata logging, the writes aremade to a sequential log. The complete transaction is written there before it is moved to the file system structures. If the system crashes during file system data updates, the updates can be completed based on the information in the log. Thus, logging ensures that file system changes are made completely (either before or after a crash). The order of the changes is guaranteed to be correct because of the sequential writes to the log. If a change was made incompletely to the log, it is discarded, with no changes made to the file system structures. Therefore, the structures are either consistent or can be trivially made consistent via metadata logging replay.