1 Problem – Concurrency (10 Points)

1. (10 Points) One process is running on a single-processor system. The process has two threads that run concurrently. The code for the two threads are as follows:

Thread A	Thread B
for i = 1 to 5 do x = x + 1;	for j = 1 to 5 do x = x + 1;

Suppose *x* is a global variable shared by two threads, and the computer uses a load-store architecture, which means to add 1 to a memory location, you have to load it into a register, add 1 to the register, and then store the register back. Suppose that *x* is initialized to be 0, list **all the possible values** of *x* after both threads have completed. To get full credit, *briefly* specify how can *x* end up with each of the values that you think is possible.

Answer:

The loop body contains three instructions:

(1) R <= x (load)
(2) R <= R + 1 (add)
(3) x <= R (store)

Without loss of generality, suppose the last store instruction is from thread A, then the load instruction of the last loop of thread A actually determines the final value of x. If n is loaded by that instruction, then n+1 is stored as the final value of x. Now we say that the following values could be loaded by that instruction:

1: if thread B loads 0, then thread A executes the first four loops, then B stores 1, overwrites whatever is in x. A loads 1 at this time, B continues to finish, then A stores 2 into x as the final value.
2: B loads 0, stores 1, loads 1, A does first 4 loops, B stores 2, A loads 2 at 5^{th} loop.
3: B does 2 loops and loads 2, A does first 4 loops, B stores 3, A loads 3 at 5^{th} loop.
4: B does 3 loops, loads 3 in 4^{th} loop, A does first 4 loops, B stores 4, A loads 4.
5: B does 4 loops, loads 4 in 5^{th} loop, A does first 4 loops, B stores 5, A loads 5
6: A does the first loop and stores 1. B does 4 loops and loads 5 in 5^{th} loop, A does next 3 loops, B stores 6, A loads 6 in 5^{th} loop.
7: A does first two loops, …
8: A does first three loops,…
9: A does first four loops, …

Therefore, the final value of x could be any one from 2 through 10.

2 Problem – Memory Management (10 Points)

1. (4 Points) What is Copy on Write and why it can improve system performance?

Answer:

In Unix systems, when a process does a *fork* system call, no page copy is done for the new process. Both processes can access the same pages. However, if one process attempts to write some data, then a copy is made so that each process has its own private copy of the data. This is called copy on write. It can improve the system performance because if no process ever writes data to a page, it is never copied.

2. (3 Points) In a computer system, on average it takes 20 nsec to execute an instruction if there are no page faults. Suppose it has been measured that in every 1 million instructions executed, one will cause a page fault, and it takes 40 ms (1 ms = 10^6 nsec) to handle a page fault (in addition to executing the faulting instruction itself). What is the effective instruction execution time?

Answer:

40 ms / 1 million = 40 nsec.

Effective instruction execution time is 20 + 40 = 60 nsec.

(Page fault handling will execute some number of instructions. We ignore this because the number is supposed to be small compared with 1 million. The 40ms is mainly spent waiting for I/O)

3. (3 Points) Suppose we have $100 to improve the performance of the computer system described in the previous problem. We have two options: we can buy a new CPU which is twice as fast (i.e., average instruction execution time is 10 nsec when no page faults) and replace the old one, or we can buy an additional memory module which will halve the page fault rate (i.e., 1 fault every 2 million instructions). Which improvement should we adopt in order to maximize the performance?

Answer:

New CPU: effective instruction execution time is 10 nsec + 40 nsec = 50 nsec;

More Memory: 20 nsec + 20 nsec = 40 nsec;

So should buy the additional memory module.

3 Problem –I/O systems (10 Points)

1. (5 Points) Briefly describe the steps taken to print a string on a printer that is controlled by interrupted I/O. Suppose the printer can accept one character at a time.

Answer:

1. The user issues the system call and enters kernel mode; the data is copied to kernel.
2. The kernel sends a character to the printer and does a context switch to other processes. The user process that does the system call is blocked.
3. When the printer prints the character, it generates an interrupt.
4. The CPU runs the interrupt handler. If there are more data to print, then it sends the next character to the printer. Otherwise it unblocks the user process. Then it returns to the process that has been interrupted.

2. (6 Points) Briefly describe how a system call is carried out and especially how the user program passes parameters to the kernel and how the kernel passes the return value back to the user program.

Answer:

1. The user process pushes the parameters on the user stack and issues the SVC instruction
2. The SVC instruction causes a trap and the system switches from user mode to supervisor mode.
3. The trap handler decides which system routine is called and branch to that system routine.
4. The system routine copies the parameters from the user stack to the kernel space and executes
5. When the system routine finishes, it copies the return value to the user stack and return to the trap handler
6. The system switches back to user mode and returns to the user program (the instruction following the SVC instruction).

(The copy operation is done in step 4, when the specified system routine begins to execute. The reason is, first, the copy can only be done in the supervisor mode, because in the user mode the user program cannot access the kernel memory; second, only the called routine knows what parameters should be passed and therefore knows what data to copy from the user stack to the kernel)

4 Problem – File Systems (10 Points)

1. (3 Points) Some file system provides a rename system call, which changes the name of a file. Is this system call still useful, given the fact that we can always copy a file to a new file with the new name, and then delete the old file?

Answer:

The rename system call is still useful, because it doesn't have to do a file copy, which improves the system performance.

2. (4 Points) Unix has a command "tail", which prints the tail of a file. For example, "tail –n 10 file1" prints the last 10 lines of file1 on the screen. This command often has a better performance than just open the file with an editor and jump to the end if the file is very large. Explain why this is the case. Is there any system in which such a command wouldn't have any performance advantages?

Answer:

Unix file system is indexed. When we execute the tail command, it can just read the last few data blocks without touching other data (probably by seeking to the end of the file, and the file system does this by extracting the pointers to the last few data blocks from some index block). If we use an editor, it will often read the whole file. When the file is very large, the difference could be significant.

If the file data is managed by linked list instead of index, then the tail command will not have any performance advantages.

3. (3 Points) Continuous allocation is often not used in general file systems. But in many multimedia file systems it is considered a good choice, why?

Answer:

Continuous allocation is often not used in general file systems because it will cause fragmentation on the disk as files are created and deleted. Also it's not flexible because it's sometimes difficult to append to a file. These problems don't exist in multimedia file systems, because once the multimedia file is written onto the disk, it doesn't need to be modified. Moreover, since continuous allocation reduces the intra-file disk seek when reading the data, it is desirable in a multimedia file system

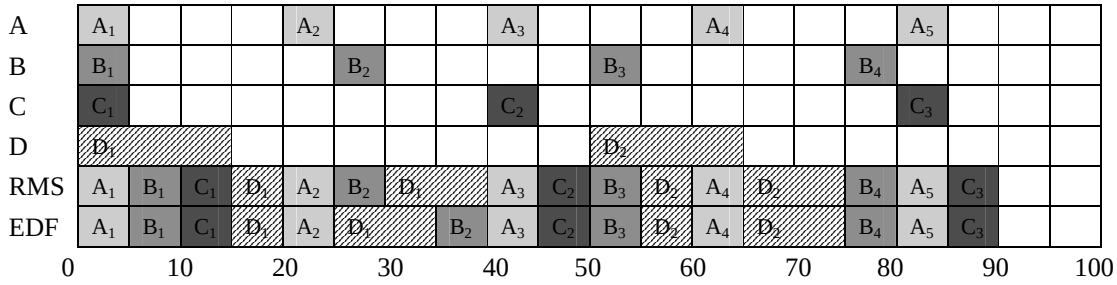
5 Problem – Multimedia Systems (20 Points)

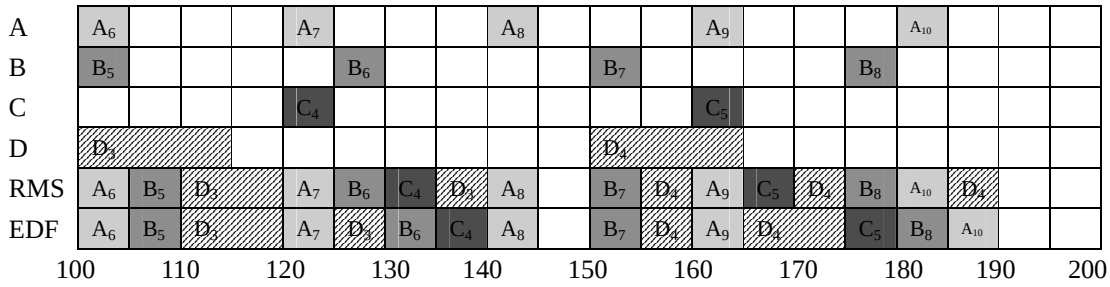
Consider a real-time system with 4 processes as shown in the following table.

Process	Processing time	Period
A	5ms	20ms
B	5ms	25ms
C	5ms	40ms
D	15ms	50ms

- (16 Points) Using Rate Monotonic Scheduling algorithm and Earliest Deadline First algorithm, draw the scheduling of the processes for the first 200ms. Suppose one task from each of the processes arrived at time 0. Also for EDF, if two tasks have the same deadline, then the one that *arrived* earlier has a higher priority.

Answer:





- (4 Points) According to the textbook, RMS is guaranteed to work for a 4-process system if the CPU utilization is <= 0.757. The above system has a CPU utilization of 0.875. Does RMS fail for the above system according to your scheduling? Is your result in conflict with the textbook?

Answer:

RMS doesn't fail. The result is not in conflict with the textbook, because the text books only says that if the CPU utilization is low, then the RMS is guaranteed to work. But even if the utilization is high, RMS may still work for some periods and run time configurations (but not for all configurations).

6 Problem – Distributed Systems (20 Points)

1. (8 Points) The Test and Set Lock (TSL) instruction is very useful in single-processor computer systems because it can be used for process synchronization. For example, before a process enters critical section, it can call the following procedure to make sure that only one process can be in the critical section at a time. (Suppose the *TestAndSetLock* does exactly what TSL does, i.e., it returns the value of *lock* and at the same time sets it to 1. *lock* can be thought of as a global variable shared by multiple processes, 0 means the lock is free, 1 means it is held by someone else).

```
EnterCriticalSection(lock){
        while(TestAndSetLock(lock));
}
```

   On a multi-processor computer system, the above code can still be used, but it has a serious performance problem (which doesn't exist in single-processor systems). What is it? Rewrite the *EnterCriticalSection* procedure to eliminate the problem as much as possible.

Answer:

When more than one processor is executing the TSL instruction, only one will succeed, but the others will continuously write the shared variable (lock), which will cause a lot of unnecessary cache invalidation and take up a lot of bus bandwidth. It can be re-written as follows:

```
EnterCriticalSection(lock){
        while(1){
                while(lock);
                if(!TestAndSetLock(lock)) break;
        }
}
```

(Busy waiting is not the problem we mean, because it is present in single-processor systems.)

2. (5 Points) What is false sharing in distributed shared memory (DSM) system and how can it be reduced?

Answer:

In distributed shared memory system, if the page size is large, it is possible that two processors frequently access two different variables, but the two variables happen to be on the same page. Therefore the page is transferred back and forth between the two processors, even if they are not sharing any data. This is called false sharing. Reduce the page size can make false sharing less likely.

3. (7 Points) In a distributed file system such as AFS, more than one client can open the same file for modification at the same time. When the file server sees a file modified by different clients, it tries to resolve the conflict (to determine if there's really a conflict, e.g., two clients write to the same portion of a file, or it is not a real conflict, i.e., the two writes can be merged, and merge the writes if possible). For regular files this is often very difficult. However, it is often surprisingly easy to resolve the conflict for directory files. Explain why this is the case.

Answer:

Two clients will write to a directory file only when they create, delete or rename a file in that directory. Since the directory file is structured, it is possible to merge different writes. For example, if both clients create a file with different names, this can be merged into the final directory file without generating a conflict.

7 Problem – Security (20 Points)

1. (5 Points) What are access control list and capability list? How do they provide protection when a process wants to access a file?

Answer:

Access control list is a list associated with each object (file); it specifies which domain (process) has what access rights to it. Capability list is a list associated with each process; it specifies the list of objects (and operations) that this process has access to. When a process wants to access a file, the access control list of the file is examined to see if this process has the necessary access right; or the capability list of the process is examined to see if the file and the necessary access right is in the capability list.

2. (5 Points) One student found that some virus has infected his personal computer. He immediately took the following steps for recovery:
    1. Reboot the infected system.
    2. Back up all files to an external medium.
    3. Run *fdisk* to format the disk.
    4. Reinstall the operating system from the original CD-ROM.
    5. Reload the files from the external medium.

    Name two serious errors in these steps.

Answer:

First, Should run anti-virus software to check the files before copying them to external medium.

Second, Shouldn't run *fdisk* of the original system because it might have been infected.

3. (5 Points) What is the use of digital signature and how does it work in a public key system?.

Answer:

Digital signature is used to show that a document really comes from someone who signed the document. It also makes it impossible for the sender to repudiate a message that he has signed.

The sender first computes a hash function of the document (MD5 or SHA, etc.), then "decrypts" the digest with his private key. This signature is sent with the document itself. When the receiver receives the document, he calculates the digest; he also "encrypts" the signature and gets the digest calculated by the sender. If the two digests match, then the document is authentic. Otherwise it has been tampered (or accidentally changed). The sender cannot repudiate it because only he has his own private key, no one else can use his private key to sign the document.

4. (5 Points) Explain in a public key system, how does two clients establish a secure communication channel with each other? Suppose there's a trusted server and both clients have already got the trusted server's public key, and each client has a secret (symmetric) key.

Answer:

Both clients can send their secret keys to the trusted server, encrypted with the server's public key. Then both client has a secure communication channel with the server. If client 1 wants to communicate with client 2, then client 1 can ask the server to send client 1's secret key to client 2, encrypted in client 2's secret key. Client 2 decrypts the message using his own secret key and get client 1's secret key. Client 2 can then communicate with client 1 using client 1's secret key.