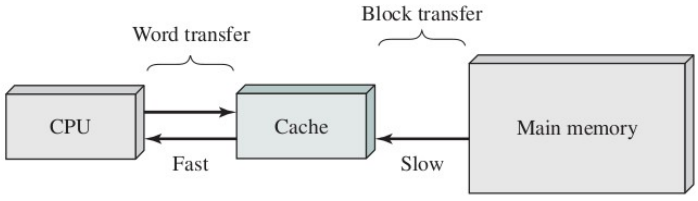
UNIVERSITY OF BIRMINGHAM

M.Sc. CS/AI & CS
Computer Systems
Operating Systems and Architecture [Solutions]

Exercise #1: Assuming that a CPU is operating at 1 MHz, and is able to execute one instruction / clock cycle by taking benefit of pipelining. The DMA module on the same system is transferring characters (one byte at a time) to the main memory from an external device transmitting at 9600 bits per second. By approximately how much the processor will be slowed down due to the DMA activity?

Let us ignore data read/write operations and assume the processor only fetches instructions. Then the processor needs access to main memory once every microsecond. The DMA module is transferring characters at a rate of 1200 characters per second, or one every 833 µs. The DMA therefore "steals" every 833rd cycle. This slows down the processor approximately 1 / 833 * 100% = 0.12%

Exercise #2: Suppose that the processor has access to two levels of memory. Level 1 contains 1000 bytes and has an access time of 0.1µs; level 2 contains 100,000 bytes and has an access time of 1µs. Assume that if a byte to be accessed is in level 1, then the processor accesses it directly. If it is in level 2, then the byte is first transferred to level 1 and then accessed by the processor. For simplicity, we ignore the time required for the processor to determine whether the byte is in level 1 or level 2. Such a two level memory can be shown as in figure below:



(a) Single cache

We can define hit-ratio H, as the fraction of all memory accesses that are found in the faster memory e.g. the cache memory. T1 is the access time to level 1 (cache) and T2 is the access time to level 2 (main memory). Now suppose 95% of the memory accesses are found in the cache (H=0.95). Then what would be the average time to access a byte from such a two-level memory sub-system? Hint, the graph below shows the average access time to a two-level memory as a function of the hit-ratio H.

The following figure shows the average access time to a two-level memory as a function of the hit-ratio H.

H x T1 + (1 – H) x (T1 + T2)

0.95 x 0.1µs + 0.05 x (0.1µs + 1µs) = 0.095 + 0.055

= 0.15µs

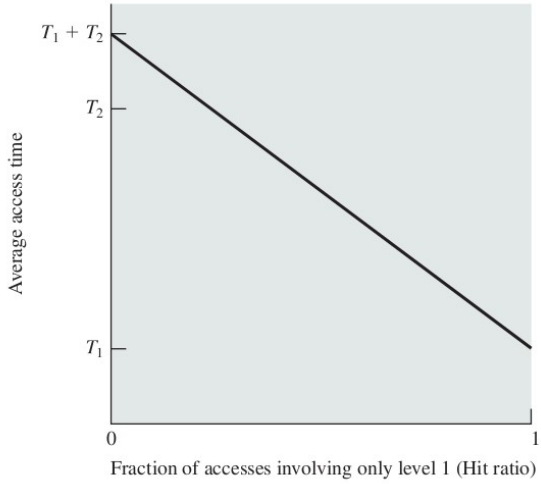


Figure 1.15 Performance of a Simple Two-Level Memory

Exercise #3: A computer system has a cache, main memory, and a disk used for virtual memory. If a referenced word is in the cache, 20ns are required to access it. If it is in the main memory but not in cache, 60ns are needed to load it into cache (this includes the time to originally check the cache), and then the reference is started again. If the word is not in main memory, 12ms are required to fetch the word from disk, followed by 60ns to copy it to the cache, and then the reference is started again. The cache hit-ratio is 0.9 and the main memory hit-ratio is 0.6. What is the average time in ns required to access a referenced word on this system?

Location of referenced word	Probability	Total time for access in ns
In cache	0.9	20
Not in cache, but in main memory	(0.1)(0.6) = 0.06	60 + 20 = 80
Not in cache or main memory	(0.1)(0.4) = 0.04	12ms + 60 + 20 = 12,000,080

So the average access time would be:

$$\text{Avg} = (0.9)(20) + (0.06)(80) + (0.04)(12000080) = 480026 \text{ ns}$$

Exercise #4: What is a program? and what is a process? Discuss as many differences between them as you can imagine.

Program: A program is a **group of instructions** to carry out a specified task. When we execute a program that was just compiled, the OS creates a process to execute the program. Execution of the program starts via GUI mouse clicks, command line entry of its name, etc. A program is a passive entity as it resides in the secondary memory, such as the contents of a file stored on disk. One program can have several processes.

Process: A **process** is a **program in** execution. The term process refers to program code that has been loaded into a computer's memory so that it can be executed by the CPU. A process can be described as an instance of a program running on a computer or as an entity that can be assigned to and executed on a processor. A program becomes a process when loaded into memory and thus is an active entity.

Difference between Program and Process:

	Program	**Process**
1.	Program contains a set of instructions designed to complete a specific task.	Process is an instance of a program.
2.	Program is a passive entity as it resides in the secondary memory.	Process is a active entity as it is created during execution and loaded into the main memory.
3.	Program exists at a single place (such as a disk) and continues to exist until it is deleted.	Process exists for a limited span of time as it gets terminated after the completion of task.

4.	Program does not have any resource requirement, it only requires memory space for storing the instructions.	Process has a high resource requirement, it needs resources like CPU, memory address, I/O during its lifetime.
5.	Program does not have any control block or context.	Process has its own control block called PCB as well as its context.

Exercise #5: List down three reasons for a process to move from running state to blocked state?

A process is put into the blocked state if it requests something for which it must wait. A request to the OS is usually in the form of a system call. A process may enter blocked state when:

1. it issues a request for IO (read / write) to a file on secondary storage (disk / usb etc) or a network socket / connection.
2. it requests for a resource (such as a memory, a semaphore, a mutex etc) which is currently not available and being used by another process 信号量、互斥体
3. it tries to access a protected section of code that is currently locked by some other process / thread.

Exercise #6: Multi-programming (or multi-tasking) enables more than a single process to apparently execute simultaneously. How is this achieved on a uniprocoessor where we have only one CPU?

Multi-programming is achieved on a uniprocessor by the concept of scheduling. Every process can be given a certain amount of time, called a timeslice and when the timeslice of a process is finished, the CPU scheduler switches to a different process. The ultimate goal is to keep the system responsive while really maximising the processor's ability to process.

Exercise #7: Calculate number of times hello is printed:

```
int main() {
      fork();
      fork();
      fork();
      printf("hello\n");
      return 0;
}
```

The number of times ‘hello’ is printed is equal to number of processes created. Total Number of Processes = 2^n, where n is number of fork system calls. So here n = 3, 2^3 = 8

Exercise #8: Predict the Output of the following program:

```
int main() {
    int x = 1;
    if (fork() == 0)
        printf("Child has x = %d\n", ++x);
    else
        printf("Parent has x = %d\n", --x);
    return 0;
}
```

Output:

```
Parent has x = 0
Child has x = 2
     (or)
Child has x = 2
Parent has x = 0
```

Here, global variable change in one process does not affected two other processes because data/state of two processes are different. And also parent and child run simultaneously so two outputs are possible.