ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Operating Systems and Networks	Spring 2016
Assignment 5	Due 19 April 2016

Part I: File Systems

1.1 Is the "open" system call in UNIX absolutely essential? What would be the consequences (i.e., how would you need to change other system calls, what impact on performance do you expect) of not having it?

1.2 It has been suggested that the first part of each file be kept in the same disk block as its inode. What good would this do?

1.3 An Operating System only supports a single directory, but allows that directory to have arbitrarily many files with arbitrarily long file names. Can something approximating a hierarchical file system be simulated? How?

1.4 Systems that support sequential files always have an operation to rewind files. Do systems that support random access files need this too?

1.5 Contiguous allocation of files leads to disk fragmentation if files are deleted. Is this internal fragmentation or external fragmentation? What if the disk is accessed in blocks and we demand that each block contains at most data of one file?

1.6 One way to use contiguous allocation of disk space and not suffer from holes is to compact the disk every time a file is removed. Since all files are contiguous, copying a file requires a seek and rotational delay to read the file, followed by the transfer at full speed. Writing the file back requires the same work. Assuming a seek time of 5 msec, a rotational delay of 4 msec, a transfer rate of 8MB/sec and an average file size of 8KB, how long does it take to read a file into main memory then write it back to the disk at a new location? Using these numbers, how long would it take to compact half of a 16GB disk?

1.7 Consider an inode structure with 12 direct addresses, one indirect address, one double indirect address and one triple indirect address. Each block is 4KB in size. Assuming block addresses are 32 bit values, what is the maximum file size?

1.8 Free disk space can be kept track of using free list or a bit map. Disk addresses require D bits. For a disk with B blocks, F of which are free, state the condition under which the free list uses less space than the bit map. For D having the value 16 bits, express your answer as a percentage of the disk space that must be free.

1.9 The open() syscall returns a filehandle, which allows us to read() and write() to that file. In order to delete a file we use the unlink() syscall, which takes the pathname of the file to delete as its parameter. What happens if we create/open a file, and delete it right after, can we still use the file descriptors returned from open()? Write a short program to try it out. How can a user access the contents of the "deleted" file without modifying your program?

1.10 Implement your own version of the `ls` utility. Of course you do not need to implement all the options `ls` provides, emulating the behaviour of `ls -l --color=never` is sufficient. Hint: start by reading the man pages for opendir(), readdir() and fstat().

Part II: I/O Systems

2.1 Why might a system use interrupt-driven I/O to manage a single serial port, but polling I/O to manage a front-end processor, such as a terminal concentrator?

2.2 Polling for an I/O completion can waste a large number of CPU cycles if the processor iterates a busy-waiting loop many times before the I/O completes. But if the I/O device is ready for service, polling can be much more efficient than is catching and dispatching an interrupt. Describe a hybrid strategy that combines polling, sleeping and interrupts for I/O device service. For each of these three strategies (pure polling, pure interrupts, hybrid), describe a situation in which that strategy is more efficient than is either of the others.

2.3 How does DMA increase system concurrency? How does it complicate hardware design?