CO211 - Operating Systems Tutorial Sheets

Tutorial 1 - Introduction

1. The issue of resource allocation shows up in different forms in different types of operating systems. List the most important resources that must be managed by an operating system in the following settings;

   (a) Supercomputer

   Since this is most likely used for computation, processor time as well as memory should be carefully managed.

   (b) Workstations connected to servers via a network

   Network access and bandwidth.

   (c) Smartphone

   Energy, since it is a portable device, as well as access to hardware such as the camera, GPS, as well as connectivity (Bluetooth, mobile network, etc).

2. What is the kernel of an operating system?

   The kernel of the operating system is part of the operating system that remains in memory, executing in the privileged part of the CPU.

3. Why is the separation into a user mode and a kernel mode considered good operating system design? Give an example in which the execution of a user process switches from user mode to kernel mode, and then back to user mode again.

   Any bugs executing in user space should not cause the entire system to crash, since the kernel allows for recovery. If all programs were to run in kernel mode, a failure would bring down the entire system. An example of this switch would be writing to disk (or any system call in general).

4. Which of the following instructions should only be allowed in kernel mode, and why?

   (a) Disable all interrupts   kernel only

   If something in user were to disable interrupts, the kernel would have no way of regaining control,

   (b) Read the time of day clock   user

   All user processes should be able to access the time if needed.

   (c) Change the memory map   kernel only

   Managing memory should be restricted to the kernel.

   (d) Set the time of day   kernel only

   Processes running in user space should not be able to change the time, as it can cause issues for other processes.

5. A portable operating system is one that can be ported from one system architecture to another with little modification. Explain why it is infeasible to build an operating system that is portable without any modification. Describe two general parts that you can find in an operating system that has been designed to be highly portable.

   Since the operating system must interact with the hardware, it's not feasible to build an OS that can interact with every hardware configuration. Device drivers (**platform specific**) allow for the operating system to interact with hardware, and this can be provided by the hardware manufacturer. Another part would be an API the OS provides to programs (**platform independent**), allowing them to interact with hardware via this abstraction.

Tutorial 2 - Processes + Threads

1. If a multithreaded process forks, a problem occurs if the child gets copies of all the parent's threads. Suppose that one of the original threads was waiting for keyboard input. Now two threads are waiting for keyboard input, one in each process. Does this problem ever occur in single-threaded processes?

   No, this does not happen in single-threaded processes as the entire process would be blocked when waiting for input, and therefore cannot fork.

2. What is the biggest advantage of implementing threads in user space? What is the biggest disadvantage?

   The biggest advantage is that it allows the thread scheduling to be managed by the programmer, and also avoids the overhead of context switching. On the other hand, the main disadvantage is that if any of the user space threads were to block (to wait for input), it would context switch to another thread.

3. If in a multithreaded web server the only way to read from a file is the normal blocking `read()` system call, do you think user-level threads or kernel-level threads are being used?

   Kernel-level threads are being used, as the entire web server would block if it was done with user space threads.

4. Why would a thread ever voluntarily give up the CPU by calling `thread_yield()`? After all, since there is no periodic clock interrupts, it may never get the CPU back.

   It is done to lower the priority of the thread in the scheduler, allowing another thread to run. This allows threads to cooperate.

5. The register set is a per-thread rather than a per-process item. Why? After all, the machine has only one set of registers.

   When a thread is stopped, it has its own contents in a register which must be saved (and then restored transparently to the thread).

6. In a system with threads, is there one stack per thread or one stack per process when user-level threads are used? What about when kernel threads are used? Explain.

   Since each thread can call procedures, it must have a stack per thread in order to store local variables and calls. The same can be said for kernel threads.

7. In this problem you are to compare reading a file using a single-threaded file server and a multithreaded server, running on a single CPU-machine. It takes 15 ms to get a request for work, dispatch it, and do the rest of the necessary processing, assuming that the data needed are in the block cache. If a disk operation is needed, as is the case one-third of the time, an additional 75 ms is required, during which time the thread sleeps. For this problem, assume that thread switching time is negligible. How many requests/sec can the server handle if it is single-threaded? If it is multithreaded?

   Let the average time for an operation be $15 + \frac{1}{3} \cdot 75 = 40$ ms. Therefore, with a single thread, it can process 25 requests/s.

   We want to calculate the probability that all threads are waiting for I/O. The average blocking time per thread is 25 ms, as seen above. This means that the probability that all threads are blocked is $1 - (\frac{25}{40})^n = 1 - (\frac{5}{8})^n$, As such, we can calculate the number of requests per second as;

$$\left(1 - \left(\frac{5}{8}\right)^n\right) \cdot \frac{1000}{15}$$

   Note $\frac{1000}{15}$ requests/sec is at full efficiency.

8. Would an algorithm that performs several independent CPU-intensive calculations concurrently (e.g. matrix multiplication) be more efficient if it used threads, or if it did not use threads? Why is this a hard question to answer?

It would be more efficient as long as the overhead for threads is negligible compared to the performance increase, and the CPU is actually able to perform the computation in parallel (multiple cores), otherwise the overhead of threads will cause it to be less efficient. This is difficult to answer as it depends on the problem itself, how it is divided, as well as the system specifications.

9. IPC mechanisms

    (a) What happens when a signal is received by a process?

    Other than `SIGKILL` and `SIGSTOP`, the receiving process is able to choose what it does with the signal. It can either ignore it, or manually handle it. Otherwise, it generally terminates the process.

    (b) When two processes communicate through a pipe, the kernel allocates a buffer (of size 65536 bytes in Linux) for the pipe. What happens when the process at the write-end of the pipe attempts to sen additional bytes on a full pipe?

    It cannot write, and will block until the process at the read-end reads from it, thus freeing up space.

    (c) What happens when the process at the write-end of the pipe attempts to send additional bytes and the process at the read-end has already closed the file descriptor associated with the read-end of the pipe?

    The writing process will have an error returned to it.

    (d) The process at the write-end of the pipe wants to transmit a linked list data structure (with one integer field, an a "next" pointer) over a pipe. How can it do this?

    Since they do not share address spaces, it must be serialised in some form by the sending process, which can then be converted back into a linked list by the receiving process.

    (e) When would it be better for two processes to communicate via shared memory instead of pipes? What about the other way around?

    It would be better for two processes to communicate via shared memory as it is faster due to the lack of kernel intervention - it also allows for bi-directional communication. On the other hand, pipes are handled by the kernel, thus synchronisation does not have to be implemented by the programmer.

Tutorial 3 - Scheduling

1. State which of the following are true and which false.

    (a) Interactive systems generally use non-preemptive processor scheduling.

    False. An interactive system uses preemptive scheduling - if there was a background process that was taking up too much time, the user interface would appear unresponsive. A preemptive scheduler allows this process to be preempted, giving control to the process running the user interface.

    (b) Turnaround times are more predictable in preemptive than in non-preemptive systems.

    False. In a non-preemptive system, a process will run to completion (or until it blocks).

    (c) One weakness of priority scheduling is that the system will faithfully honour the priorities, but the priorities themselves may not be meaningful.

    True. The actual priority of a job can depend on what other processes are running at the same time.

2. (a) Give an example showing why FCFS is not an appropriate scheduling scheme for interactive users.

   Assume that there are two tasks on a system. The first task, is a large computation that happens in the background, and takes a notable amount of time (and is purely CPU bound, hence no blocking). The second task is a browser. Since the first task would be run first, the browser will be unresponsive to the end user, since the first task occupies the resources until it finishes.

   (b) Using the previous example, show why round-robin is a better scheme for interactive users.

   Round-robin switches between the two tasks, thus ensuring both tasks get the processor periodically. As such, the browser will be able to respond (since the large task can be interrupted), and the computation can continue after the browser uses up its time quantum.

3. Five jobs are waiting to be run. Their expected run times are $9, 6, 3, 5, X$. In what order should they be run to minimise average turnaround time?

   It should be run with the shortest job first, and the longest job last (X should be placed in the list $3, 5, 6, 9$ accordingly).

4. Five batch jobs, A through E, arrive at a computer centre at essentially the same time. Their estimated running times, and priorities (lower value is higher priority) are as follows;

job	running time (minutes)	priority
A	15	6
B	9	3
C	3	7
D	6	9
E	12	4

   For each of the following scheduling algorithms, determine the turnaround time for each job, and the average turnaround time for all jobs. Ignore process switching overhead and assume all jobs are completely CPU bound.

   (a) non-preemptive priority scheduling

job	turnaround time (minutes)
B	9
E	$9 + 12 = 21$
A	$21 + 15 = 36$
C	$36 + 3 = 39$
D	$39 + 6 = 45$

   Hence the average turnaround time is 30 minutes.

   (b) FCFS (in order A, B, C, D, E)

job	turnaround time (minutes)
A	15
B	$15 + 9 = 24$
C	$24 + 3 = 27$
D	$27 + 6 = 33$
E	$33 + 12 = 45$

   Hence the average turnaround time is 28.8 minutes.

   (c) shortest job first

job	turnaround time (minutes)
C	3
D	$3 + 6 = 9$
B	$9 + 9 = 18$
E	$18 + 12 = 30$
A	$30 + 15 = 45$

Hence the average turnaround time is 21 minutes.

(d) round robin with a time quantum of 1 minute

The order is as follows; note that a teal quantum is the final quantum for that job.

ABCDEABCDEABCDEABDEABDEABDEABEABEABEAEAEAEAAA

job	turnaround time (minutes)
A	45
B	35
C	13
D	26
E	42

Hence the average turnaround time is 32.2 minutes.

Tutorial 4 - Synchronisation

1. Explain why the following statement is false; "when several threads access shared information in main memory, mutual exclusion must be enforced to prevent the production of indeterminate results."

   If multiple threads are **reading** the same shared information, then mutual exclusion is not needed. Mutual exclusion is needed if the shared information can be modified.

2. Discuss the pros and cons of busy waiting.

   Busy waiting is useful when the wait times are short, as we therefore do not need to invoke the kernel for context switching. Additionally, they are used within the kernel, since it cannot use a blocking abstraction. On the other hand, busy waiting wastes CPU time as it is doing nothing, but occupying the processor.

3. One requirement in the implementation of the semaphore operations `up` and `down` is that each of these operations must be executed atomically; once started, each operation runs to completion without interruption. Give an example of a simple situation in which, if these operations are not executed atomically, mutual exclusion may not be properly enforced.

   Assume we are using a binary semaphore, (where the counter is initialised to 1). Two threads are attempting to access a critical region, and therefore both `down` the semaphore. If this wasn't atomic, it's possible that both threads fall into the branch where the counter is greater than 0, and then both attempt to decrement it. This means that neither threads are blocked, and both continue to execute, thus both being in the critical region.

4. Can two threads in the same process synchronise using a kernel semaphore if the threads are implemented by the kernel? What if they are implemented in user space? Assume no threads in other processes have access to the semaphore.

   Yes, as the kernel handles the synchronisation by providing the semaphores. On the other hand, user space threads cannot use semaphores, since blocking would block all threads, and therefore user space threads must have another method of synchronisation implemented.

5. Does the strict alternation solution work the same way when process scheduling is preemptive?

   Yes, it was designed with preemption. If the scheduling isn't preemptive, and the process that's waiting runs first, it may run indefinitely, since it hasn't completed execution.

6. Give a sketch of how a uni-processor operating system that can disable interrupts could implement semaphores.

Any operations involving a semaphore should disable interrupts right at the start. This ensures atomicity of the semaphores.

When a semaphore is initialised with `init(s, i)`, the `counter` should be set to `i`, and an empty queue is associated with the semaphore. When a semaphore is downed with `down(s)`, it should first check if the counter is greater than 0 - if it is, it decrements the counter, otherwise it adds the calling thread to the queue and blocks the thread (this is atomic). Similarly, if a thread calls `up(s)`, it should disable interrupts, then check the semaphore's queue. If the queue isn't empty, a process in the queue is resumed, otherwise the counter is incremented.

7. Consider the following three threads;

```
T1:                          T2:                          T3:
 1   a = 1;                    1   b = 1;                    1   a = 2;
 2   b = 2;
```

(a) Show all possible thread interleavings.

a = 1;	a = 1;	a = 1;	a = 2;	a = 1;	a = 1;	a = 1;	a = 2;	b = 1;	b = 1;	b = 1;	a = 2;
b = 2;	b = 2;	a = 2;	a = 1;	b = 1;	b = 1;	a = 2;	a = 1;	a = 1;	a = 1;	a = 2;	b = 1;
b = 1;	a = 2;	b = 2;	b = 2;	b = 2;	a = 2;	b = 1;	b = 1;	b = 2;	a = 2;	a = 1;	a = 1;
a = 2;	b = 1;	b = 1;	b = 1;	a = 2;	b = 2;	b = 2;	b = 2;	b = 2;	a = 2;	b = 2;	b = 2;
(2, 1)	(2, 1)	(2, 1)	(1, 1)	(2, 2)	(2, 2)	(2, 2)	(1, 2)	(2, 2)	(2, 2)	(1, 2)	(1, 2)

(b) If all thread interleavings are as likely to occur, what is the probability to have `a = 1` and `b = 1` after all threads complete execution?
$$\frac{1}{12}$$

(c) What about `a = 2` and `b = 2`?
$$\frac{5}{12}$$

8. Synchronisation within monitors uses condition variables and two special operations, `wait` and `signal`. A more general form of synchronisation would be to have a single primitive, `waituntil`, that had an arbitrary boolean predicate as a parameter. Thus, one could say;

```
waituntil x < 0 or x + z < n
```

The `signal` primitive would be no longer needed. This scheme is clearly more general than that of Hoare, but it is not used, why?

This is more expensive, the entire predicate must be re-evaluated every time any of the variables in the predicate change. On the other hand, processes can only be awakened with a `signal` primitive, when using a Hoare monitor.

Tutorial 5 - Deadlocks

1. Suppose that there is a resource deadlock in a system. Give an example to show that the set of processes deadlocked can include processes that are not in the circular chain in the corresponding resource allocation graph.

Consider a cycle in the resource allocation graph, such that process A is holding resource R, and wants to acquire resource S, and similarly B is holding S and is attempting to acquire R. If a process C tried to acquire either R or S, it would be deadlocked, but not in the cycle.

Note that an arrow from a square (resource) to a circle (process) denotes the process holding the resource, and an arrow from a circle to a square denotes the process waiting for (attempting to acquire) a resource.
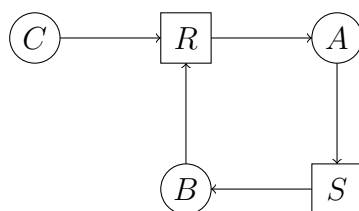
2. Consider a system that uses the banker's algorithm to avoid deadlocks. At some time a process P requests a resource R, but is denied even though R is currently available. Does it mean that if the system allocated R to P, the system would deadlock?

No, it is possible for P to complete execution, and free all the resources. However, we no longer have the guarantee that a deadlock will not occur, hence it is **possible** the system will deadlock.

3. Two processes A and B each need three records, 1, 2, and 3 in a database. If A asks for them in the order 1, 2, 3, and B asks for them in the same order, deadlock is not possible. However, if B asks for them in the order 3, 2, 1, then deadlock is possible. With three resources, there are $3! = 6$ possible combinations each process can request resources. What fraction of all combinations is guaranteed to be deadlock free?

Since the records are arbitrary, we can reason about it with just one configuration. Assuming A asks for them in ascending order, if B were to also request 1 first, one of the two processes would obtain the lock, blocking the other, thus a deadlock will not occur. Therefore, a third of the possible combinations will not result in deadlock.

4. Can a single-processor system have no processes ready and no process running? Is this a deadlocked system?

Yes, all the processes could be blocked (waiting for I/O) - however typically there is an idle process that runs in this case. This is not a deadlocked system, thus the lack of a running process cannot be used to determine deadlock.

Tutorial 6 - Memory Management

1. Describe the difference between swapping and paging in the context of virtual memory management.

While both move data from main memory to the disk, swapping moves the entire process's address space into the swap device, whereas paging does this with individual pages from the address space.

2. What is the advantage of a paged virtual memory system with a

(a) small page size

There is less unused space, hence less fragmentation.

(b) large page size

Less overhead, hence translation is faster.

3. What is an associative memory, how does it work, and how is it implemented?

Associative memory is used to cache frequently translated memory addresses. This can be thought of as a table which holds the page number and frame number - instead of checking the page table (in main memory) directly, it can first check if the page number is in this cache, and only accesses the page table when it cannot be found. It is typically implemented in hardware for high performance.

4. Describe how a context switch affects the virtual memory system. What must the OS do to ensure that memory reference made by the newly-running process will be properly translated?

   On a context switch, the page table needs to be changed (the base register in the MMU should be changed to point to the page table in memory), as well as the cache being flushed.

5. A system implements a paged virtual address space for each process using a one-level page table. The maximum size of an address space is 16 MB. The page table for the running process includes the following entries;

page	frame
0	4
1	8
2	16
3	17
4	9

   The page size is 1024 bytes, and the maximum physical memory size of the machine is 2 MB.

   (a) How many bits are required for each page table entry?

   Since each page is 1 KB, there is a total of $2 \cdot 2^{10} = 2^{11}$ frames on physical memory - this can be represented in 11 bits. By thinking of the page table as an array, each entry needs 11 bits (as well as any additional bits for metadata).

   (b) What is the maximum number of entries in a page table?

   Similarly, there can be a total of $16 \cdot 2^{10} = 2^{14}$ pages in the virtual address space.

   (c) How many bits are there in a virtual address?

   In a virtual address, there are the 14 bits specifying the page, and then an additional 10, which indicate the offset, thus a total of 24 bits.

   (d) To which physical address will the virtual address 1524 translate to?

   Because $1023 < 1524 < 2047$, it lies in page 1. Therefore, it maps to $8 \cdot 1024 + (1524 - 1024) = 8592$.

   (e) Which virtual address will translate to physical address 10020?

   Working backwards, it lies in frame 9, hence it is in page 4. Therefore, the virtual address that translates to this is $4 \cdot 1024 + (10020 - 9 \cdot 1024) = 4900$.

6. A pure paging system uses a three level page table. Virtual addresses are decomposed into four fields, (a, b, c, d) with d being the offset. In terms of these constants, what is the maximum number of pages in a virtual address space?

$$2^{a+b+c} \text{ pages}$$

7. Calculate the access times for a four-level paging system assuming the following TLB hit ratios. of 80% and 98%. Assume that time for memory access is 100ns, and for TLB access 20ns.

   (a) 80%
   $$\underbrace{0.8 \cdot (20 + 100)}_{\text{TLB hit}} + \underbrace{(1 - 0.8) \cdot (20 + 4 \cdot 100 + 100)}_{\text{TLB miss}} = 200\text{ns}$$

   (b) 98%
   $$\underbrace{0.98 \cdot (20 + 100)}_{\text{TLB hit}} + \underbrace{(1 - 0.98) \cdot (20 + 4 \cdot 100 + 100)}_{\text{TLB miss}} = 128\text{ns}$$

How does this compare to a single-level paging system?

With single level paging, it takes 140ns, and 122ns respectively.

8. Suppose that pages in a virtual address space a referenced in the following order:

$$1\ 2\ 1\ 3\ 2\ 1\ 4\ 3\ 1\ 1\ 2\ 4\ 1\ 5\ 6\ 2\ 1$$

There are three empty frames available. Assume that paging decisions are made on demand (when page faults occur). Show the contents of the frames after each memory reference, and the number of page faults;

(a) the LRU replacement policy   11 faults

frame	reference																
	1	2	1	3	2	1	4	3	1	1	2	4	1	5	6	2	1
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2
1		2	2	2	2	2	2	3	3	3	3	4	4	4	6	6	6
2				3	3	3	4	4	4	4	2	2	2	5	5	5	1
fault	×	×		×			×	×			×	×		×	×	×	×

(b) the clock policy   9 faults

frame	reference																
	1	2	1	3	2	1	4	3	1	1	2	4	1	5	6	2	1
0	1	1	1	1	1	1	4	4	4	4	4	4	4	5	5	5	5
1		2	2	2	2	2	2	2	1	1	1	1	1	1	6	6	6
2				3	3	3	3	3	3	3	2	2	2	2	2	2	1
fault	×	×		×	×		×		×		×			×	×		

Tutorial 7 - Device Management

1. In which of the four I/O software layers (user-level I/O software, device-independent OS software, device drivers, and interrupts handlers) is each of the following done?

(a) computing the track, sector and head for a disk read   device drivers

This requires familiarity with the actual disk layout. Modern hard disks will often perform the mapping entirely in hardware by the disk controller.

(b) maintaining a cache of recently used blocks   device-independent OS software

A block cache is useful across block I/O devices, thus it can be shared if implemented in this layer.

(c) writing commands to the drive registers   device drivers

This can also be done in the interrupt handler (if it quick and time-critical).

(d) checking to see if the user is permitted to use the device   device-independent OS software

Access control is applicable to many devices, hence it can be reused if implemented in this layer.

(e) converting binary integers to ASCII for printing   user-level I/O software

This can be done cheaply in user space as it doesn't require any privileged access.

2. What is the difference between

(a) a device driver and a device controller?

A device driver is software in the OS kernel which communicates with the device controller (hardware) on the external device.

(b) a block-oriented device and a character-oriented device?

A block-oriented device deals with data in fixed size blocks, which is preferable when reading or writing a large amount of data (over making multiple individual requests) - this can also use a cache. On the other hand, a character-oriented device deals with data a character at a time.

3. What is memory-mapped I/O, and why is it sometimes used?

Memory mapped I/O allows a device to be addressed as a memory location. This can be used when reading from / writing to a large file, instead of loading the contents into a buffer. To implement this, the memory controller must intercept requests to address ranges representing I/O devices and put them on the PCI bus instead of the memory bus.

4. An alternative to using interrupts for I/O is polling, are there any circumstances where this is better choice?

This is a simpler mechanism (hence easier to implement) compared to interrupt-driven I/O. As such, this is advantageous in embedded systems, where the overhead of a context switch is unnecessary (if the operation finishes quickly).

5. Explain what direct memory access is, and why it is used. Although DMA does not use the CPU, the maximum transfer rate is still limited. Consider reading a block from disk. Name three factors that might ultimately limit the rate of transfer.

DMA allows the device to write directly into memory, and then raise an interrupt once the operation is complete, thus reducing the number of interrupts it requires. The following factors might limit the rate of transfer;

- speed of the disk (how fast it can actually spin)
- speed of the link (bus) between devices
- access bandwidth supported by the memory

6. What is spooling? Why is a printer spooling system better than direct user access to printers?

Spooling is used to create queue of jobs to be sent to a device. For example, a process can send a job to the spooling daemon for the printer, and know that it will eventually be printed (hence only the daemon has direct access to the printer). This prevents the process from blocking if there are other jobs in the queue, or interfering with other jobs which are also attempting to write into the device.

7. An operating system has to support I/O devices with very diverse properties. Complete the following table;

device	data rate	type (**C**haracter / **B**lock)	operation (**R**ead, **W**rite, **S**eek)
clock			
keyboard	10 B/s	C	R
mouse	100 B/s	C	R
56k modem	7 KB/s	C	R, W
laser printer	100 KB/s	C	W
scanner	400 KB/s	C	R
52x CD-ROM	8 MB/s	B	R, S
FastEthernet	12.5 MB/s	C	R, W
EIDE (ATA-2) disk	16.7 MB/s	B	R, W, S
ISA bus	16.7 MB/s	C	R, W
Fire Wire (IEEE 1394)	50 MB/s	C	R, W
USB 2.0	60 MB/s	C	R, W
XGA monitor	60 MB/s	C	R, W
Gigabit Ethernet	125 MB/s	C	R, W
Serial ATA disk	300 MB/s	B	R, W, S
SCSI Ultrawide4 disk	320 MB/s	B	R, W, S
PCI bus	528 MB/s	C	R, W

8. Explain how one can provide an asynchronous I/O API on top of a blocking I/O system call interface. You have to implement a web server that should handle thousands of concurrent incoming connections. What would be the advantages of using a non-blocking I/O interface for this?

   The main process can create a new thread, which will be blocked waiting for I/O, and the main process continues executing. The new thread can then signal (once it unblocks by receiving I/O) the main process, which can then handle the signal with a callback.

   If this was to use blocking I/O, there would be a significant context switching overhead (due to a thread being required for each connection). Instead, the web server can be implemented using an event-driven design, where an event loop polls active file descriptors with outstanding I/O operations.

9. Write a C program that implements the copy (`cp`) command. Your program should be invoked as

   <div align="center">

   `mycp <source file> <destination file>`

   </div>

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>

int main(int argc, char argv[]) {
  int inFile, outFile;
  char line[512];
  int bytes;

  // check if either are -1
  inFile = open(argv[1], O_RDONLY);
  outFile = open(argv[2], O_WRONLY | O_CREAT);

  while ((bytes = read(inFile, line, sizeof(line))) > 0) {
    write(outFile, line, bytes);
  }

  close(inFile);
  close(outFile);
}
```

Tutorial 8 - Disk Management

1. A disk controller with enough memory can perform read-ahead, reading blocks on the current track into its memory before the CPU asks for them. Should it also do write-behind (reporting back to the CPU that a block has been written once it is stored in the disk controller's memory)?

   Not in general, as we will encounter data loss if the system loses power before the disk controller writes to the disk - we need to ensure that data is actually written if it the drive reports the block has been written. However, if the drive is able to complete the write after power loss (with some sort of emergency power), then write-behind is possible.

2. Suppose that the current position of the disk arm is over cylinder 200. The disk request queue contains requests for sectors on the following cylinders;

$$400, 20, 19, 74, 899$$

   In which order will the requests be handled under:

   (a) FCFS   400, 20, 19, 74, 899

   As the name implies, it processes the requests in order. However, this has a drawback as it can lead to random seek patterns. However, under light load, this is fine as it is a fair scheduling method.

   (b) SSTF   74, 20, 19, 400, 899

   This travels to the cylinder with the shortest distance from the current position of the head. Due to this, the head tends to stay in the middle, possibly causing the innermost and outermost tracks to be starved.

   (c) SCAN   400, 899, 74, 20, 19

   This chooses requests which result in the shortest seek time, but in a given direction. Once the queue has no more requests in the given direction, it starts going in the other direction. However, this can still lead to delays for the extreme locations.

   (d) C-SCAN   400, 899, 19, 20, 74

   This is similar to scan, but once it reaches the outermost track, it goes straight to the innermost track (or vice versa) - hence the scanning only occurs in one direction. This reduces the variance for times for requests to extreme tracks.

3. Disk requests come in to the disk drive for tracks 10, 22, 20, 2, 40, 6, and 38, in that order. A seek takes 5 ms per track moved. In all cases the arm is initially at track 20. How much seek time is needed for;

   (a) FCFS   10, 22, 20, 2, 40, 6, 38

$$5 \cdot (10 + 12 + 2 + 18 + 38 + 34 + 32) = 730 \text{ ms}$$

   (b) SSTF   20, 22, 10, 6, 2, 38, 40

$$5 \cdot (0 + 2 + 12 + 4 + 4 + 36 + 2) = 300 \text{ ms}$$

   (c) SCAN   20, 22, 38, 40, 10, 6, 2

$$5 \cdot (0 + 2 + 16 + 2 + 30 + 4 + 4) = 290 \text{ ms}$$

4. Consider the following parameters describing a disk;

parameter	description
C	number of cylinders
T	number of tracks per cylinder (number of platters)
S	number of sectors per track
ω	rotational velocity (rotations per second)
B	number of bytes per sector

(a) How many bytes of data are on each disk cylinder?

$$(B \cdot S \cdot T) \text{ bytes}$$

(b) Which parameters would be changed, and why, to reduce the expected rotational latency for requests?

The rotational velocity would have to be increased, since the rotational latency is $\frac{1}{2\omega}$.

(c) Which parameters would be change to reduce the disk's transfer time?

The transfer speed is $\frac{1}{SB\omega}$, therefore the number of bytes per track ($S \cdot B$) and / or the rotational velocity would have to be increased.

5. A disk drive has the same parameters as above. Consider s_1 and s_2, consecutive sectors on the same track of the disk (s_2 will pass under a read / write head immediately after s_1). A read request for s_1 arrives at the disk and is serviced. Exactly d seconds ($0 < d < \frac{1}{\omega}$) seconds after the disk completes that request, a read request for sector s_2 arrives at the disk. Assuming no intervening requests, how long will it take the disk to service the request for sector s_2?

$$t_{\text{access}} = t_{\text{seek}} + t_{\text{latency}} + t_{\text{transfer}}$$
$$= 0 + \frac{1}{\omega} - d + \frac{1}{S\omega}$$

6. A disk drive has the same parameters as above. However, for simplicity, we assume that the disk only has one single-sided platter (hence $T = 1$). The following function gives the relationship between seek distance d, in cylinders, and seek time t_{seek} in milliseconds;

$$t_{\text{seek}} = \begin{cases} 0 & d = 0 \\ 5 + 0.05 \cdot d & 0 < d \leq C \end{cases}$$

The sectors are laid out and numbered sequentially, starting with the outer cylinder, as shown in the diagram below.
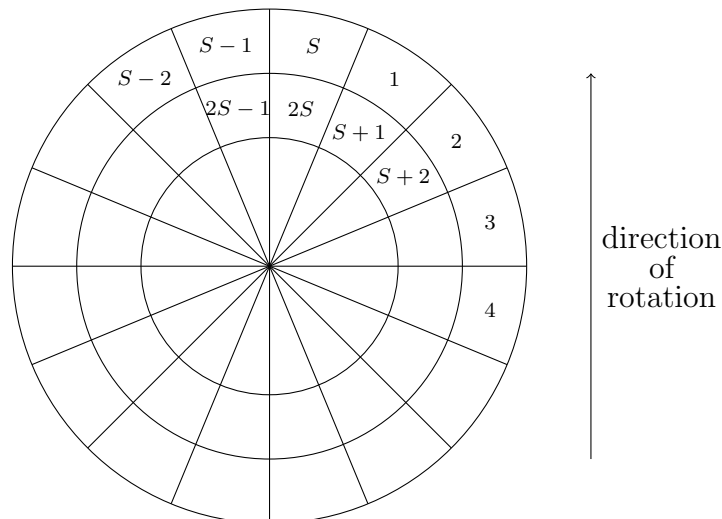
(a) Suppose the disk read / write head is located over cylinder 10. The disk receives a request to read sector S. What is the expected service time for this request?

$$t_{\text{service}} = t_{\text{seek}} + t_{\text{latency}} + t_{\text{transfer}}$$
$$= 5 + 0.05 \cdot 10 + \frac{1}{2\omega} + \frac{1}{S\omega}$$
$$= 5.5 \text{ ms} + \frac{1}{2\omega} + \frac{1}{S\omega}$$

(b) Exactly d milliseconds after completing the request for S, the disk receives a request for sector $S + 1$. What is the expected service time for this request?

$$t_{\text{service}} = t_{\text{seek}} + t_{\text{latency}} + t_{\text{transfer}}$$
$$= 5 + 0.05 \cdot 1 + \frac{1}{\omega} \cdot (d + t_{\text{seek}}) + \frac{1}{S\omega}$$
$$= 5.5 \text{ ms} + \frac{1}{\omega} \cdot (d + t_{\text{seek}}) + \frac{1}{S\omega}$$

7. A disk drive has $T = 1000$ tracks per surface, and $S = 10$ sectors per track. The platters spin at a rate of $\omega = 100$ rotations per second. The following function relates seek distance d in cylinders, to the seek time (in milliseconds);

$$t_{\text{seek}} = 0.1 \cdot d + 5$$

Sectors s_1 and s_2 are consecutive sectors on the same track of cylinder 100 (s_2 will pass under the read / write head immediately after s_1 does).

(a) The read / write heads are initially located over cylinder zero. The disk receives a request for sector s_1. After servicing that request, it is idle for a time, and then receives a request for sector s_2. What is the sum of the expected service times for these two requests?

$$t_{\text{service}} = t_{\text{seek}} + 2 \cdot t_{\text{latency}} + 2 \cdot t_{\text{transfer}}$$
$$= (0.1 \cdot d + 5) + 2 \cdot \frac{1}{2 \cdot \omega \cdot \underbrace{\frac{1}{1000}}_{\text{convert}}} + 2 \cdot \frac{1}{S \cdot \omega \cdot \underbrace{\frac{1}{1000}}_{\text{convert}}}$$
$$= 0.1 \cdot 100 + 5 + 2 \cdot \frac{1}{2 \cdot 0.1} + 2 \cdot \frac{1}{10 \cdot 0.1}$$
$$= 15 + 10 + 2$$
$$= 27 \text{ ms}$$

Note that the time for latency and transfer are the same for the second request, however seek time is not needed.

(b) The read / write heads are initially located over cylinder zero. The disk receives a single request to read sectors s_1 and s_2. What is the expected service time for this request?

$$t_{\text{service}} = t_{\text{seek}} + t_{\text{latency}} + t_{\text{transfer}}$$
$$= (0.1 \cdot d + 5) + \frac{1}{2 \cdot \omega \cdot \underbrace{\frac{1}{1000}}_{\text{convert}}} + \frac{2}{S \cdot \omega \cdot \underbrace{\frac{1}{1000}}_{\text{convert}}}$$
$$= 0.1 \cdot 100 + 5 + \frac{1}{2 \cdot 0.1} + \frac{2}{10 \cdot 0.1}$$
$$= 15 + 5 + 2$$
$$= 22 \text{ ms}$$

Tutorial 9 - File Systems

1.

Tutorial 10 - Security

1.