Example Sheet for Operating Systems I (Part IA)

Solutions for Supervisors
Michaelmas 2018 / Last Updated: March 7, 2019
Note these may be updated in the light of feedback. (Check update time.)

1 Processes & Scheduling

1. (a) Modern computers store data values in a variety of "memories", each with differing size and access speeds. Briefly describe each of the following:
   i. cache memory
   ii. main memory
   iii. registers

   (b) Give an example situation in which operating systems effectively consider disk storage to be a fourth type of "memory".

2. (a) Describe (with the aid of a diagram where appropriate) the representation in main memory of:
   i. An unsigned integer
   ii. A signed integer
   iii. A text string
   iv. An instruction

   (b) Does an operating system need to know whether the contents of a particular register represent a signed or unsigned integer?

   (c) Describe what occurs during a *context switch*.

3. Describe with the aid of a diagram how a simple computer executes a program in terms of the *fetch-execute cycle*, including the ways in which arithmetic instructions, memory accesses and control flow instructions are handled.

4. Process scheduling can be *preemptive* or *non-preemptive*. Compare and contrast these approaches, commenting on issues of simplicity, fairness, performance and required hardware support.

5. (a) Describe how the CPU is allocated to processes if static priority scheduling is used. Be sure to consider the various possibilities available in the case of a tie.

   (b) "All scheduling algorithms are essentially priority scheduling algorithms."
   Discuss this statement with reference to the first-come first-served (FCFS), shortest job first (SJF), shortest remaining time first (SRTF) and round-robin (RR) scheduling algorithms.

   (c) What is the major problem with static priority scheduling and how may it be addressed?

   (d) Why do many CPU scheduling algorithms try to favour I/O intensive jobs?

6. An operating system uses a single queue round-robin scheduling algorithm for all processes. You are told that a *quantum* of three time units is used.

   (a) What can you infer about the scheduling algorithm?

   (b) Why is this sort of algorithm suitable for a multi-user operating system?

   (c) The following processes are to be scheduled by the operating system.

Process	Creation Time	Required Computing Time
P_1	0	9
P_2	1	4
P_3	7	2

   None of the processes ever blocks. New processes are added to the tail of the queue and do not disrupt the currently running process. Assuming context switches are instantaneous, determine the *response time* for each process.

   (d) Give one advantage and one disadvantage of using a small quantum.

7. (a) Describe with the aid of a diagram the life-cycle of a process. You should describe each of the states that it can be in, and the reasons it moves between these states.

   (b) What information does the operating system keep in the process control block?

   (c) What information do the shortest job first (SJF) and shortest remaining time first (SRTF) algorithms require about each job or process? How can this information be obtained?

   (d) Give one advantage and one disadvantage of non-preemptive scheduling.

   (e) What steps does the operating system take when an interrupt occurs? Consider both the programmed I/O and DMA cases, and the interaction with the CPU scheduler.

   (f) What problems could occur if a system experienced a very high interrupt load? What if the device[s] in question were DMA-capable?

2 Memory Management

1. (a) What is the *address binding* problem?

   (b) The address binding problem can be solved at compile time, load time or run time. For *each* case, explain what form the solution takes, and give one advantage and one disadvantage.

2. For each of the following, indicate if the statement is true or false, and explain why:

   (a) Preemptive schedulers require hardware support.

   (b) A context switch can be implemented by a flip-flop stored in the translation lookaside buffer (TLB).

   (c) Non-blocking I/O is possible even when using a block device.

   (d) Shortest job first (SJF) is an optimal scheduling algorithm.

(e) Round-robin scheduling can suffer from the so-called 'convoy effect'.

(f) A paged virtual memory is smaller than a segmented one.

(g) Direct memory access (DMA) makes devices go faster.

(h) System calls are an optional extra in modern operating systems.

(i) In Unix, hard-links cannot span mount points.

(j) The Unix shell supports redirection to the buffer cache.

3. Most operating systems provide each process with its own *address space* by providing a level of indirection between virtual and physical addresses.

   (a) Give *three* benefits of this approach.

   (b) Are there any drawbacks? Justify your answer.

   (c) A processor may support a *paged* or a *segmented* virtual address space.

      i. Sketch the format of a virtual address in each of these cases, and explain using a diagram how this address is translated to a physical one.

      ii. In which case is physical memory allocation easier? Justify your answer.

      iii. Give *two* benefits of the segmented approach.

4. *System calls* are part of most modern operating systems.

   (a) What is the purpose of a system call?

   (b) What mechanism is typically used to implement system calls?

5. (a) Operating systems need to be able to prevent applications from crashing or locking up the system, or from interfering with other applications. Which three kinds of hardware support do we require to accomplish this?

   (b) How do applications request that the operating system perform tasks on their behalf?

   (c) What could we do if we did not have the requisite hardware support?

6. (a) In the context of memory management, under which circumstances do *external* and *internal* fragmentation occur? How can each be handled?

   (b) What is the purpose of a page table? What sort of information might it contain? How does it interact with a TLB?

   (c) Describe with the aid of a diagram a two-level page table. Explain the motivation behind the structure and how it operates.

3 File Systems & IO

1. Explain how a program accesses I/O devices when:

   (a) it is running in supervisor-mode

   (b) it is running in user-mode

2. From the point of view of the device driver, data may be read from an I/O device using *polling*, *interrupt-driven programmed I/O*, or *direct memory access* (DMA). Briefly explain each of these terms, and in each case outline using pseudo-code (or a flow chart) the flow of control in the device driver when reading data from the device.

3. From the point of view of the application programmer, data may be read from a device in a *blocking*, *non-blocking* or *asynchronous* fashion. Using a keyboard as an example device, describe the expected behaviour in each case.

4. File systems comprise a *directory service* and a *storage service*.

   (a) What are the two main functions of the directory service?

   (b) What is a directory *hierarchy*? Explain your answer with the aid of a diagram.

   (c) What information is held in file *meta-data*?

   (d) What is a *hard link*? Does file system support for hard links place any restrictions on the location of file meta-data?

   (e) What is a *soft* (or *symbolic*) link? Does file system support for soft links place any restrictions on the location of file meta-data?

5. Suppose we have a system with three users a, b and c and ten files f_0,f_1,...f_9.

   Further suppose we have four operations for which we wish to control access: *read*, *append*, *replace* and *modify*.

   (a) Do we require all of these or can some be described by combinations of others?

   (b) Create a nontrivial example set of access tuples of the the form (user, file, permission) and show how it might be represented as:

      i. an access matrix,

      ii. access control lists,

      iii. capability sets

6. (a) Compare and contrast *blocking*, *non-blocking* and *asynchronous* I/O.

   (b) Give *four* techniques which can improve I/O performance.

4 Protection, Access Control & Case Studies

1. Describe the basic access control scheme used in the Unix filing system. How does Unix support more advanced access control policies?

2. (a) Describe with the aid of a diagram the on-disk layout of a Unix V7 filesystem. Include in your description the role of the *superblock*, and the way in which free inodes and data blocks are managed.

   (b) Describe with the aid of a diagram a Unix V7 *inode*.

   (c) Estimate the largest file size supported by a Unix V7 filesystem.

   (d) Suggest *one* reliability enhancement and *two* performance enhancements which could be made to the Unix V7 filesystem.

3. (a) Describe, with the aid of diagrams where appropriate, how Unix implements and manages:

   i. a hierarchical name space for files

   ii. allocation of storage on disk

   iii. file-system and file meta-data

   iv. pipes

   (b) A Unix system administrator decides to make a 'versioned' file-system in which there are a number of directories called `/root-dd-mm-yyyy`, each of which holds a copy of the file-system on day `dd`, month `mm` and year `yyyy`. The idea is that at any particular time only the most recent snapshot will be used as the 'real' filesystem root, but that all previous snapshots will be available by explicitly accessing the directory in question. In this way the system administrator hopes to allow resilience to mistaken edits or unintentional deletions by users, or to hardware problems such as a disk head crash.

   To implement this, the system administrator arranges for a program to run every morning at 01:00 which recursively 'copies' the current snapshot to the new one. However to save disk space, hardlinks are used in place of actual copies. Once the 'copy' is complete, the new snapshot is used as the new root.

   To what extent will this scheme provide the functionality the system administrator hopes for? What advantages and disadvantages does it have?

4. Describe *how* CPU scheduling algorithms favour I/O intensive jobs in the Unix operating systems.

5. Describe with the aid of a diagram a Unix *inode*. You should explain the motivation behind *indirect blocks*, and how they are used when accessing a file.

6. Describe the operation of the Unix shell with reference to the process management system calls it makes use of. You might like to use pseudo-code or a diagram to aid with your description.