IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2023

BEng Honours Degree in Computing Part II
MEng Honours Degrees in Computing Part II
BEng Honours Degree in Mathematics and Computer Science Part II
MEng Honours Degree in Mathematics and Computer Science Part II
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the Associateship of the City and Guilds of London Institute

PAPER COMP50004

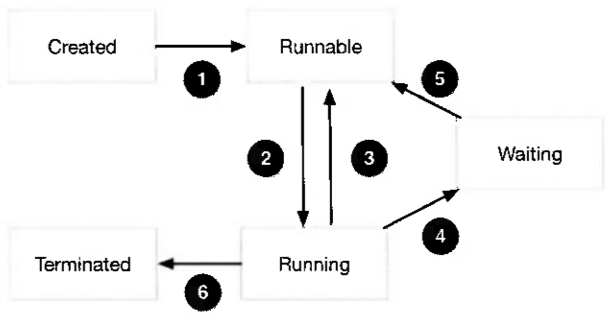
OPERATING SYSTEMS

Friday 12th May 2023, 10:00
Duration: 90 minutes

Answer ALL TWO questions

Paper contains 2 questions
Calculators required

1 An operating system kernel has a preemptive process scheduler. The scheduler uses a time quantum of 250 ms, and the machine has **2** CPU cores. Processes transition between the following states:



The scheduler must schedule a set of processes P1–P4 with the following creation times and priorities on behalf of two users:

Process	Creation time	User	Priority
P1	t = 0 sec	User 1	2
P2	t = 1 sec	User 2	0
P3	t = 1 sec	User 2	1
P4	t = 2 sec	User 1	2

(A lower number indicates a higher priority.)

a Briefly explain what event or action causes each transition ❶–❻.

b Write down the execution sequence of the four processes for the first 3 secs under the following scheduling strategies:

i) Round robin (RR)

ii) Fair-share round robin (FSRR)

iii) Multi-level feedback queues (MLFQ)

c Process P1 spends the majority of its time invoking a system call that requires each time the execution of a computationally expensive kernel task.

i) Explain how this affects the fairness of scheduling decisions under the fair-share round robin (FSRR) strategy.

ii) Describe an extension to the process state diagram above that allows the scheduler to address the issue that you identified under i).

The three parts carry, respectively, 20%, 45%, and 35% of the marks.

2 a We have measured the best- and worst-case execution time of the two following operations:

1) perform a single access to application memory, and

2) ask the OS to allocate memory.

We have also executed this application on three separate systems, each with one of the following memory translation mechanisms:

a) base and limit register,

b) paging with a 2-level hierarchical page table, and

c) paging with an inverted page table.

You should assume that time measurements only account for accesses to main memory, which takes 100 nsec, that we have ten pages of 1 KiB in size, and that we have no caches or TLBs. Specifically, you should only account for accesses to application memory itself, and for accesses to address translation structures that are directly used by the MMU and stored in memory (i.e., do not account for additional application or OS logic and data structures).

i) Assign each of the following measurements to a single (operation, mechanism) pair:

A) 200 nsec

B) 100 nsec

C) 200 to 1100 nsec

D) 300 nsec

E) 0 to 16000 nsec

F) 100 nsec

If option X) above is for the pair "(allocate, base and limit register)", just say "X) 2a".

ii) If we added a TLB to each of the systems, which of the (operation, mechanism) pairs that we measured above would change in the case of a TLB hit? Briefly describe why and by how much it would change. Assume that checking the TLB does not add latency to the access.

b We have two applications that are disk-access intensive; one does mostly reads, whereas the other does mostly writes. Identify whether a single disk or RAID level 5 will give the best performance to each of the following applications, and briefly describe why:

i) A CPU-heavy application.

ii) A CPU-heavy application, concerned with storage reliability.

iii) A disk-heavy application, mostly doing reads.

iv) A disk-heavy application, mostly doing writes.

c We have a traditional (spinning) hard disk with 10,000,000 blocks of 8192 Bytes each, with block pointers of 8 Bytes.

i) What is the maximum file size (in Bytes) that we can have in that disk using a block linkage file system? What is the worst-case number of blocks we would need to read, in order to access an arbitrary position within this file?

Assume that no blocks are cached in memory, and that no space is used outside encoding the contents of this file (e.g., there is no superblock and no directory entries).

ii) What is the maximum file size we can have in that disk using an inode-based file system? What is the worst-case number of blocks we would need to read, in order to access an arbitrary position within this file?

Assume that no blocks are cached in memory, that no space is used outside encoding the contents of this file, and that the file's inode has 16 pointers; 14 of them direct, 1 indirect and 1 doubly indirect.

iii) Assume that we know the disk will always contain a single file, without any directory structures. Briefly describe how you would change the file system to optimize disk block accesses when accessing this file. What are the best-case and worst-case block access counts of this design?

iv) Assume we want to take advantage of predictable file access patterns, by adding a simple prefetcher that moves blocks into main memory ahead of time. Which of the following software layers would be best suited to contain this new logic? Briefly describe why.

* User-level I/O software
* Device-independent OS software

* Device driver
* Interrupt handler

The three parts carry, respectively, 40%, 20%, and 40% of the marks.