**2020-2021 OS**

**\*the answers to these are mostly in tutorials however this provides more in depth answers for some questions**

**Part 1**

1. **A portable OS is one that can be ported from one system architecture to another with little modification. Explain, giving examples, why it is infeasible to build an OS that is portable without any modification at all.**

An OS must interact with the hardware of a system directly. It therefore contains code that depends on the specifics of the processor architecture such as its instruction set and the memory management system. This is accomplished with a machine-dependent layer of the OS which addresses the specifics or the hardware which will be used as examples below. Keeping the machine-dependent layer as minimal as possible aids portability.

Examples:

Different architectures differ in bus architecture used (i.e. VME, ISA, PCI, MCA)

Different architectures differ in word size of the CPU (i.e. 32 or 64 bit)

1. **Briefly describe TWO ways in which deadlocks can be prevented**

There are four conditions stated by Coffman et al (1971) which allow a deadlock, and if one of the conditions is never satisfied a deadlock is impossible.

Mutual exclusion: if no resource were ever assigned exclusively to a single process, we would never have deadlocks. An example of this is using read-only data, which allows multiple process access. This access is more complicated in writing, such as writing to a printer. Instead of allowing two or more processes to write to a printer, the printer output can be spooled, so that several processes can generate output at the same time. Therefore, the only process that requests the physical printer is the printer daemon, which never requests any other resources, and thus there is no deadlock for the printer. In sum, avoid assigning a resource unless absolutely necessary, and try to make sure that as few processes as possible may actually claim the resource.

Hold-and-wait: If we can prevent processes that hold resources from waiting for more resources we can eliminate deadlocks. This can be achieved by requiring all processes to request all their resources before starting execution. In this case, if everything is available, the process will be allocated whatever it needs and can run to completion; if one or more resources is not available, the process will just wait and nothing will be allocated. An issue with this is that a process may not know all the resources they will require. In this case another approach could be to require a process to temporarily release all of its resources it currently holds, then get everything it needs all at once.

No pre-emption condition: Some resources can be virtualised, such as using spooler for printer output to the disk and only the spooler daemon has access to the real printer, however while eliminating deadlock for the printer it would still be possible to have a deadlock over disk space (but unlikely). This is not a method to prevent deadlocks for many systems e.g. databases or tables.

Circular wait: One method of avoiding this is by using numbered resources whereby processes can request resources whenever they want to, but all requests must be made in numerical order. This will never allow a resource allocation graph to have cycles. A problem that might occur with this is that it may become impossible to find an ordering that satisfies everyone.

1. **A multi-threaded process uses a Kernel semaphore S to synchronise between its threads. What issues, if any, arise if the threads are User-level or Kernel-level threads? You can assume no threads in other processes have access to S**.

With user-level threads, when one thread blocks on a semaphore the Kernel thinks the entire process is blocked and does not run it ever again, and consequently the process fails. With kernel threads, a thread can block on a semaphore and the kernel can block on a semaphore and the kernel can run another thread in the same process.

1. **List THREE reasons why scheduling processes and threads on a multi-processor system is more complicated than scheduling them on a uni-processor system.**

On a single processor scheduling is one dimensional and the only situation is determining which thread should be run next.

On a multiprocessor, the schedule has two dimensions: which thread to run and which CPU to run it on. Sometimes it is useful to schedule threads that communicate extensively (e.g. producer consumer) close together in space so that they may benefit from sharing caches.

1. Scheduling algorithms and decisions on a multiprocessor system can be more complex

e.g. affinity scheduling which attempts to have a thread run on the same CPU it ran on last time

1. Sharing of data becomes more challenging

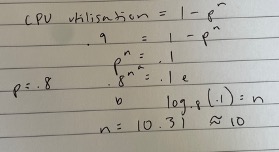
e.g. producer consumer

1. Load balancing across processors

space sharing: scheduling multiple threads at the same time across multiple CPUs

1. Difficulty of debugging due to non determinism
2. Difficult to exploit hardware level features in user-level programming

1. **In a system that allows multi-programming, how many processes need to be in main memory to only waste 10% of CPU? Assume each process spends around 80% waiting for I/O. Show your working clearly.**



1. **Consider the following scenario with three processes *P*1, *P*2, *P*3 and two buffers *B*1 and *B*2.**

**– *P*1 takes a character from Input and writes it to *B*1**

**– *P*2 takes a character from *B*1 and writes it to *B*2**

**– *P*3 takes a character from *B*2 and writes it to Output**

**Each process runs in a never ending loop and can read/write one character at a time (reads are destructive) and each buffer can only hold one character. Given the concurrent nature of access, write a programme (pseudo-code) to coordinate these three processes using semaphores. Please state your assumptions.**

Assumptions: all three process operate on one file at a time

Semaphore empty1 = 1, empty2 = 1

Semaphore full1 = 0, full2 = 0

P1(){

while 1 do:

wait(empty1)

read(next\_file(), buffer\_1)

signall(full1)

}

P2(){

While 1 do:

Wait(full1)

Wait(empty2)

Copy(buffer2, buffer1)

Signal(empty1)

Signal(full2)

}

P3(){

While 1 do:

Wait(full2)

Print(buffer2)

Singla(empty2)

}

**Part 2**

1. **How does the page size defined on a particular architecture affect the working set size of processes on that architecture?**

Working set size – the amount of memory (the pages or frames) that a process requires in a given time interval

If a page is smaller than what the process requires in terms of memory, then a process may use multiple pages, resulting in a larger working set size. With larger pages a process would therefore have fewer pages and therefore smaller working set size.

The smaller the page size, the more accurate determination of work set, which is in-turn smaller and requires less memory.

1. **An alternative to using interrupts for I/O is *polling* (where you keep checking the status). List the circumstances in which using *polling* is a better choice?**

Polling can be more efficient than interrupt driven IO in a few scenarios. If the wait time is likely to be short, or absolute deterministic timing is desired, polling may be more efficient. This may be this case if the device and controller are fast and there is significant data to transfer. With frequent IO, interrupts can be costly in terms of context switching so for high IO events it is beneficial to use polling. A specific example can be seen with small embedded systems where polling is also easier to implement, as no interrupt handler (additional hardware) is necessary.

1. **A file system uses I-nodes with single-indirect and double-indirect blocks and a block size of 1 KB. Approximately by what factor does the maximum possible file size increase if the block size of the file system is doubled.**

1. **Consider a scenario where processes are communicating with each other by writing and reading blocks to disk. Would your choice of disk scheduling algorithm influence the synchronisation between the processes. Briefly discuss your answer.**

Each disk scheduling algorithm has its own way of ordering IO requests. Hence, the IO requests will not be performed in the order they are produced. Therefore, irrespective of disk scheduling algorithm used, the synchronisation between the processes will be affected.

1. **In a file system that uses I-nodes briefly describe (using an example) the advantages or disadvantages of using only triple indirect pointers to locate all file blocks.**

The advantages are that it is easy to implement and expandable, easy to access small files.

The disadvantage is that it is more difficult and inefficient to access large files and that file data is spread around the disk rather than contiguous.

For small files read and space overhead is introduced. The indirect inodes need to be read from the disk, increasing access times. For larger files indirect inodes would be used to reduce the size of one pointer block, so triple indirect pointers are a better fit for large files.

1. **Consider a virtual memory system with a page size of 8 KB and a page table entry of 4 bytes. Please answer the questions below stating any assumptions.**
2. **Assuming that every page table fits into a single page, how many levels of page tables would be required to map a 46-bit virtual address?**

# entries on a page = page size / address = 4KB/46 = 2^10.46438

Therefore need 10.4 bits  11 bits

Offset = 2^13

Therefore d = 13, p = 33 and need 11\*3 = 33

Therefore you would need 3 levels

Number of entries per page = 8KB/4B = 2^11

Number of bits required (for each level since each page table fits on a single page) = 11

Offset = 13

46-13 = 33

33/11 = 3 (2 levels)

**ii) Assuming the presence of a Translation Lookaside Buffer (TLB), how many memory operations are required to read or write a single 32-bit word? Discuss your answer.**

Using a TLB means we can avoid the sequence of memory accesses for the 3 page tables in order to find the frame number, since the TLB is cached.

If, when looking up a page, the TLB hit is successful (page found in the register), then a read or write operation will take just one memory access. However, if the TLB is unsuccessful then there will need 4 memory accesses, 3 to go to each page (outer to inner) and then one more to access the respective frame.