**Exam 2020**

Answers copied from the tutorials

**Question 1:**

1. Mutual exclusion, Hold and wait, No preemption, and Circular wait
2. System call (trap), exception caused by the user process, receiving an interrupt.
3. Registers, stack pointer, and PC need to be saved and restored. For threads from different processes, we also need to load the pointer to the top-level page table of the new process. No need to save this for the old process as it does not change and is already in the PC
4. User-level threads: Yes, since they use locks/semaphores at the user-level and which are ‘invisible’ to the scheduler. Kernel-level threads: Yes and no. Kernel-level threads which run with interrupts disabled would not get context-switched, but can be otherwise.

e)

(i) P1 → P2 → P3 → P4 → P5; Avg = (16+21+35+58+60)

5 = 38

(ii) R1 – P1(5) → P2(5) → P3(5) → P4(5) → P5(2); Time left – P1 = 11, P3 = 9, P4 = 18

R2 – P1(5) → P3(5) → P4(5); Time left – P1 = 6, P3 = 4, P4 = 13

R3 – P1(5) → P3(4) → P4(5); Time left – P1 = 1, P4 = 8

R4 – P1(1) → P4(5); Time left – P4 = 3; R5 – P4(3)

Average turnaround time = (10+22+46+52+60)

5 = 38

(ii) P5 → P2 → P3 → P1 → P4; Avg = (2+7+21+37+60)

5 = 25.4

f) To get the maximum value of X, the processes that increment X must read X before it is decremented

and then write X back after the processes that decrement X have completed. One method in which this

happens:

P0 and P2 both enter CS (S = 0) and both read X = 0

P2 writes X (X = -2) and exits CS (S = 1); P3 enters CS (S = 0) and reads X = -2

P0 writes X (X = 1) and exits CS (S = 1); P1 enters CS (S = 0) and reads X = 1

P3 writes X (X = -4) and exits CS (S = 1)

P1 writes X (X = 2) and leaves CS (S = 2). Hence, max value of X = 2

**Question 2:**

a) ASIDs allow the TLB to protect certain entries by having entries from several processes,

otherwise on a context switch the entire TLB must be flushed

b) Using inodes – offer temporary spatiality and has quick access for both small as well as large

files. FAT would also work but large files need lots of disk accesses.

c) No. The process is requesting blocks one after another and each request will block the process

before the next request is processed. Hence, the requests are processed in order (though addresses are

random). There will be no improvement even if we change the algorithm, since that algorithm will have

to process the requests in the same order

d) There is some performance impact as COPY creates a new copy, while with RENAME

only the inode entry needs to be changed. Some space impact as well since the old file with COPY is

still available. With COPY the file attributes will change (creation time and date, etc.) while with

RENAME this would not be the case.

e) (i) Device-independent OS layer: A block cache is useful across multiple block I/O devices. By putting

this functionality into the device-independent layer, it can be reused/shared by multiple different

devices.

(ii) Interrupt handler (or device driver): If this is a quick but time-critical operation, then it can be

performed within the interrupt handling routine. If updating the drive registers requires more time

(or is device-specific), then it would be done by the device driver.

f)

(i) Number of pages = Address size

P age size = 2^36/2^13 = 2^23

(ii) Average size of process = 8 GB = 2^33 bytes

1 Level – each process would need the entire page table in this case

Size of page table = total number of pages \* size of page table entry = 2^23 \* 4 bytes = 2^25 = 32MB

2 Level – each part of page table being paged must ideally fit on a page

Number of bits for page offset = 13 (size of page = 2^13)

Number of entries on one page = page size

address size = 8 KB/36 bits = 2^10 + . . .

No of bits required = 11

Address bits left for top-level page table = 36 - 11 - 13 = 12

Number of pages for process = size of process

page size = 2^33/2^13 = 2^20

Num inner-level pages = 2^20/2^11 = 2^9; Num outer-level pages = 2^9/2^12 = 1

Total size = Num of inner-level pages \* their size + Num of outer-level pages \* their size

Page table entry = 4 bytes ⇒ Total size = 2^9 \* 2^11 \* 4 + 1 \* 2^12 \* 4 ≈ 4 MB