(1)

(a) In a multiprogramming system, the processor is used by other programs while a process is performing I/O, to increase CPU utilisation. Hardware interrupts are therefore needed to notify the CPU when I/O is complete, so the CPU can continue with the I/O process without degradation of CPU utilisation.

(b) A deadlock occurs when a set of processes are waiting for an event to occur that only another process can cause, as a result, the system does not make progress. This differs from starvation as during starvation, the system as a whole makes progress but a particular process is denied CPU time due to other higher priority processes being continuously scheduled.

(c) Kernel-level threads should be used. If a user level thread blocks on a read() system call, the entire process will be blocked as the kernel is unaware of user-level threads. This defeats the whole purpose of multi-threading and will destroy performance.

(d)

I/O Bound processes - I/O bound processes by definition spend most of their time waiting for I/O. Under this scheduling algorithm they are used 'fairly' and hence have a high priority in using the CPU once complete, providing a fair scheduling solution.

CPU Bound processes - The scheduling algorithm ensures that all processes eventually get CPU time, so no one process is starved. The algorithm is fair in the case of CPU-bound processes.

(e)

```
int customers = 0;
int chairMutex = 1;
int freeChairs = N;
int hairdresser = 1;
```

customer:

```
down(chairMutex);
if (freeChairs > 0){
    freeChairs--;
    up(chairMutex);
    up(customers);
    down(hairdresser);
else {
    up(chairMutex);
}
```

hairdresser:

```
while(true){
  down(chair);
  down(customers);
  freeChairs++;
  up(hairdresser);
  up(chairMutex);
}
```

(2)

(a)

Would be good practice due to security, the user/OS thinks the file is deleted but still resides on disk, poses a security risk if disk is compromised.

Erasing the block prior to deleting it wouldn't be good either due to I/O overhead, ideally it should be done as a background daemon.

(b)

$t_{seek} = 10ms$

$$t_{latency} = \frac{1}{2r} = \frac{1}{2\left(\frac{10000}{60}\right)} = 3ms.$$

$$t_{transfer} = \frac{320}{320} \times \frac{1}{\left(\frac{10000}{60}\right)} = 6ms.$$

For reading all sectors on just track the total time taken is 19ms. We need to read a total of $\frac{2560}{320} = 8$ tracks, so read 7 more, assume seek time is negligible:

$$t_{tot} = 19ms + (7 \times 9ms) = \underline{82ms}$$

(c)

(i) Formatting I/O should be done by user level I/O software. It is lightweight and often specific to the users formatting requirements.

(ii) Permissions should be checked by device independant OS software, permissions ensure managed access and should be done by the kernel, this is device independant and it makes sence to be shared.

(d)

Direct pointers → First 8 blocks, one disk access.

Indirect pointers → $\left(\frac{B}{P}\right)^W = \left(\frac{1024}{4}\right)^1 = 2^8$ blocks, two disk accesses.

Doubly indirect pointers → $\left(\frac{B}{P}\right)^W = \left(\frac{1024}{4}\right)^2 = 2^{16}$ blocks, three disk accesses.

So:

1 disk access if $N \leq 8$

2 disk accesses if $8 < N \leq (8 + 2^8)$

3 disk accesses if $(8 + 2^8) < N$

Formally:

$$f(N) = \begin{cases} 1 & \text{if } N \leq 8 \\ 2 & \text{if } 8 < N < (8+2^8) \\ 3 & \text{if } (8+2^8) < N \end{cases}$$

(e)

(i)

Page size = 16KB so page offset is 2^{14} = 14 Bits.

40 - 14 = 26 Bits to reference page table entries.

Physical address = 32 bits, it is made by combining a frame address + offset so frame address is 32 - 14 = 18 Bits + 4 Bits = 22 Bit page table entry.

Total size = $2^{26} \times \left(\frac{22}{8}\right)$ = 176 MB.

(ii)

TLB contains a page number and frame number. Page number is 26 Bits and frame number is 18 Bits, and 4 permission bits.

TLB entry = 48 Bits

Total size = $\frac{48 \times 16}{8}$ = 96B