4a) Without a buffer, when the incoming data rate spikes, any data which the application cannot process immediately is lost. With a buffer, the spikes in transmission rate can be smoothed by storing unhandled data in a buffer, to be caught up on when the transmission rate drops below the processing rate again.

b) In this case, a single 1MB buffer is not enough to handle the incoming transmission rate of 5MB/s for 2 seconds at a time, and once the buffer fills, the incoming data will be lost as with no buffer, although less data will be lost than with no buffer. A latency of around 1 second will also be introduced, since the buffer must fill before being delivered to the application. Also included is the time taken to move the buffered data to the process, during which time the user process must be suspended, and the buffer cannot take in new data.

c) Again the buffers are too small to avoid losing incoming data during spikes, but now there will be a latency of only 0.5 seconds. In this case, the buffered data from the filled buffer can be transferred to the process while the other is filled, so the time to move the filled buffer is no longer an issue, and there is not point when the buffer cannot take in new data (other than spikes). While the process is taking in data faster than the transmission rate, the buffers must be managed to that they can only be transferred once they are full, which might mean that the process has to wait occasionally.

d) Outside of the spikes, the process can consume data faster than the transmission, so the buffers will be mostly empty. During spikes, 5MB/s for 2 seconds is 10MB of data, of which the process can only handle 2MB, which means that the buffers must take care of the other 8MB, which requires 16 separate 512kB buffers.

e) If the buffer can be simultaneously read and written, it removes the need for the process to wait until the buffer has been entirely written before it can read it. This adds some complication, because the buffer must be read and written cyclically (once the process reaches the end of the buffer, it starts from the beginning again), which may make it complicated if the incoming data speed is different from the process speed, since either one of them must spend time waiting so that they don’t overtake the other. A simpler solution which would reduce the wait time would be to have more, smaller buffers, since then they take less time to fill.

f) When a process switch occurs, the operating system must store the program counter, and any other registers that the process is using, as well as any OS-dependent information that the operating system needs to restore program state. These are stored in memory in a process control block, and a handle to this data structure is stored in a queue of other processes waiting to run.

The IO buffers are not stored along with program state because they are not necessary for the program to resume where it left off in execution - they belong to the operating system, not the program. They are already stored in memory by the OS, and cannot be overwritten by another program, so there is no reason to store them again in a different memory location.