1. **Process Creation**
   Explain the functionality of the `fork()` and `exec()` system calls. How are the two used in together when the Unix shell runs a command or user program.

2. **User level vs. Kernel Threads**
   One of the differences between user level and operating system supported threads is that:
   - while user level threads share a virtual address space that has only one stack segment,
   - OS supported threads can each have their own stack segment.

   Explain one important feature that having a separate stack segment per thread allows OS threads to support that cannot be supported by user level threads.

3. **Interprocess Communication**
   a. If a Unix process reads from a pipe but no data has been written to that pipe, what action does the kernel take in regard to the reading process?
   b. If a writing process then writes more data to the buffer than the reader has requested, where will any excess data (not delivered to the reader) reside?

4. **Process Context**
   A typical process context consists of the general purpose registers, the Program Counter, the Stack Pointer, the Program Status Register (PSR) (which contains the user/kernel mode bit) and a pointer the Virtual Memory View (a data structure that determines the process’ virtual memory configuration). Explain the function of the user/kernel mode bit and its importance in isolating a user process from the kernel and other processes.

5. **Multicore Processors and OS Threads**
   A multicore processor can be used to execute processes in an operating system that has no kernel support for threads, but it can be used in a different way if the kernel does provide such support. Explain how a multicore processor can be used in the first case (no OS support for threads), and why better performance in some programs can be expected in the latter case (with OS support for threads).