I.

1. a. Mutual Exclusion: Each road is not shared. Hold and wait: Vehicles can't release a resource a road until the other road is clear. No preemption: vehicles don't release roads until they are complete. Circular waiting: Each lane needs another and they end up forming a circle. b. Only allow in vehicle from 3 lanes at a time.

2. Dead lock prevention ensure one of the 4 conditions above does not hold so that processes can't deadlock. Dead lock avoidance is keeping processes in a safe state so that deadlocks can not occur. Bankers algorithm is used to ensure a safe state.

3. There are three cases for each process A. the process has no resources B. the process has one resource C. the process has two resources. If a process is an A then it is not blocking any other process. If a process is in C then it has two resources and can complete it's and then will release the resources. The only problem case is B. However because we only have three process, and four resources, One process will get the last resource and will then enter case C and complete. So even if all process has one resource there is still enough resources for one to complete, which means a deadlock cannot occur.

4. Internal fragmentation happens when we over allocate memory for a process. Say I need 4 bytes and I am given 8. 4 of those can not be used because they are assigned to me, but I don't need them. This happens in paging because the smallest unit that can be allocated is a page, which

can be more then is needed. External fragmentation is when an allocated space of memory does not fit exactly into a spot on physical memory. In segmentation not all chunks are the same size and it is hard to get 100% utilization of memory. When placing the allocated memory we might find a spot that is just a little bit bigger then what we need, however if that is the best fit, the memory is placed there. The margin that bigger then what we needed is the external fragmentation because it probably won't be used.

5. Virtual address are mapped to physical addresses by finding the page in the page table that the memory maps to, then applying the offset. In order to access the page table, there needs to be a register with the page table location. The time needed to access a memory location is the time it take to load something from memory twice. Once to get it's table location and a second time to retrieve it from it's actual location. To speed this up we can use a cache called the TLB.

6. Pages are powers of two so that address can be broken into the page and offset. If it was not a power of two we would have to adjust.

7. Paging a page table creates a hierarchical page table. This is done because without it the page table would be too large. There are other options for page table structure such as Hashed page tables and Inverted page tables.

8.

II.

1. a. 0x600000 to 0x601000 b. $sp = 0x7fffffffe790. $fp = 0x7fffffffe7a0. The Stack pointer indicates where the next function can be placed, where the next free address is. The Frame pointer is in a fixed place at the start of a function so a function can hardcode offsets to this in order to access local data. c. rip

2. $pc is now 0x7ffff7a2d830. It has entered into a function that is in lib c, not in our program.

3. Sum is placed on the stack so it's address is just offset from that. x is passed as an argument.

4. As the program runs it uses more and more memory making VIRT RES go up over and over. %CPU stays pretty low. % Memory goes up. SHR stays in the same spot.

5. As more memory is allocated the size of line 4 changes. It grows to lower addresses because the heap grows from the bottom up.

Map 1 737f5b777000 Map 5 711c9adec000

On my machine virtual memory max is set to unlimited. This is larger then physical memory. As the process can write some memory out to the hard drive.

1. I did not see a difference. But i'd image you could not allocate as much memory. The page faults would probably increase.