Project 4 FRQ

Question 1

1. `pde = &pgdir[PDX(va)];`: This line calculates the index into the page directory (`pgdir`) for the given virtual address (`va`). `PDX` is a macro that extracts the page directory index from a virtual address.
2. `if (*pde & PTE_P) {...}`: Checks if the Page Table Entry (PTE) in the page directory is present (`PTE_P` bit is set). If it is present, it retrieves the page table address from the PTE.
3. `pgtab = (pte_t *)P2V(PTE_ADDR(*pde));`: Converts the physical address stored in the PTE to a virtual address using the `P2V` macro.
4. If the PTE is not present, it checks whether allocation is requested (`alloc != 0`) and allocates a new page table using `kalloc()` if necessary.
5. `memset(pgtab, 0, PGSIZE);`: Initializes the new page table with zeroed memory.
6. Updates the PTE in the page directory with the physical address of the new page table and sets the necessary flags (`PTE_P`, `PTE_W`, `PTE_U`).
7. `return &pgtab[PTX(va)];`: Returns the address of the Page Table Entry (PTE) within the page table that corresponds to the provided virtual address (`va`). `PTX` is a macro that extracts the page table index from a virtual address.

We typically use `walkpgdir` when we need to traverse the page tables to find the PTE corresponding to a given virtual address. The arguments include the page directory (`pgdir`), the virtual address (`va`), and a flag (`alloc`) indicating whether to allocate a new page table if it doesn't exist. If `alloc` is non-zero, the function may allocate a new page table if needed.

Question 2

The `mappages` function is used to create Page Table Entries (PTEs) for a range of virtual addresses, mapping them to corresponding physical addresses. This function is typically used during the setup of page tables to establish the mapping between virtual and physical memory regions.

`char *a, *last;`: These variables represent the starting virtual address (`a`) and the last virtual address (`last`) in the range that needs to be mapped. They are initially set to the page-aligned versions of the provided virtual address (`va`) and the page-aligned version of the ending virtual address (`va + size - 1`). The function then iterates over the virtual address range using a loop. In each iteration:

- It uses the `walkpgdir` function to obtain the Page Table Entry (PTE) for the current virtual address (`a`).
- Checks if the PTE is already present (`PTE_P` bit is set). If it is, it panics since remapping is not allowed.
- Sets the PTE to point to the specified physical address (`pa`) with the given permissions (`perm`).
- Moves to the next page-aligned virtual address and increments the physical address by the page size.

Question 3

`memmove()` is used to copy a specified number of bytes from one location in memory to another but `kalloc()` is to allocate memory in the kernel space. In short, `memmove()` is memory movement but `kalloc()` is allocation of new memory.

Question 4

The edge case is that when the counter for that PPN is 0, meaning that there is only one process accessing to it. That process will completely own it and hence we need to change its flag of shared and writable. We need to implement `walkpgdir` to find the PTE entry corresponding to the virtual address we want to such that we can modify its flag accordingly.

Question 5

False, a counter for each process is not necessary.

Question 6

False, we don't need to allocate memory in the implementation of `cow()`.

Question 7

1. Purpose: Copy-on-Write mechanism is implemented to optimize the handling of forked processe but pagefault() can handle more general situations where a process accesses a memory page that is not currently in physical memory.
2. Scope: COW is specifically designed to handle copy-on-write scenarios (mainly fork) but pagefault() is a general mechanism with page faults.