1 Question 1

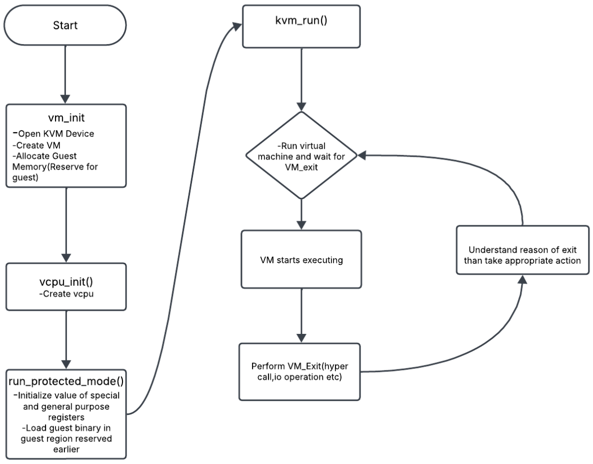

Figure 1: logical actions to setup and execute a VM

2 Question 2

2.1 Question a

The declaration:

`extern const unsigned char guest64[], guest64_end[];`

defines two external symbols that mark the start (`guest64`) and end (`guest64_end`) of a guest binary, typically embedded in the program using a linker script or assembly file. These symbols allow the program to determine the size of the guest binary (`guest64_end - guest64`) and copy it into memory for execution within a virtual machine using KVM.

2.2 Question b

The following code sets up the page tables and special registers for a 64-bit guest in a KVM (Kernel-based Virtual Machine) environment.

```
pml4[0] = PDE64_PRESENT | PDE64_RW | PDE64_USER | pdpt_addr;
pdpt[0] = PDE64_PRESENT | PDE64_RW | PDE64_USER | pd_addr;
pd[0] = PDE64_PRESENT | PDE64_RW | PDE64_USER | PDE64_PS;

sregs->cr3 = pml4_addr;
sregs->cr4 = CR4_PAE;
sregs->cr0 = CR0_PE | CR0_MP | CR0_ET | CR0_NE | CR0_WP | CR0_AM | CR0_PG;
sregs->efer = EFER_LME | EFER_LMA;
```

Explanation:

- `pml4[0]`: Sets up the first entry of the PML4 (Page Map Level 4) table, pointing to the PDPT (Page Directory Pointer Table). The entry includes flags indicating presence, read/write access, and user-level access.

- `pdpt[0]`: Sets the first entry of the PDPT, pointing to the Page Directory (PD) with similar flags.
- `pd[0]`: Sets the first entry of the PD, marking it as a 1GB page using the `PDE64_PS` flag.
- `sregs->cr3 = pml4_addr`: Loads the CR3 register with the base address of the PML4 table, setting up paging.
- `sregs->cr4 = CR4_PAE`: Enables Physical Address Extension (PAE), which is required for 64-bit long mode.
- `sregs->cr0`: Enables protected mode, paging, and other necessary CPU features.
- `sregs->efer`: Enables long mode in the Extended Feature Enable Register (EFER), required for 64-bit execution.

This setup ensures that the guest operates in 64-bit long mode with paging enabled.

2.3 Question c

The following code snippet sets up memory for a virtual machine (VM) using `mmap()` and optimizes it with `madvise()`:

```
vm->mem = mmap(NULL, mem_size, PROT_READ | PROT_WRITE,
               MAP_PRIVATE | MAP_ANONYMOUS | MAP_NORESERVE, -1, 0);

madvise(vm->mem, mem_size, MADV_MERGEABLE);
```

Explanation:

- `mmap()` is used to allocate a memory region of size `mem_size` for the virtual machine.
- The parameters used in `mmap()`:
  - `NULL`: The kernel selects the memory address.
  - `mem_size`: Specifies the size of the allocated memory.
  - `PROT_READ | PROT_WRITE`: Grants read and write access.
  - `MAP_PRIVATE`: The mapping is private (changes are not shared).
  - `MAP_ANONYMOUS`: The memory is not backed by a file.
  - `MAP_NORESERVE`: Prevents reserving swap space for the mapping.
  - `-1, 0`: Since `MAP_ANONYMOUS` is used, these are ignored.
- `madvise(vm->mem, mem_size, MADV_MERGEABLE)`:
  - Suggests to the kernel that identical memory pages can be merged using Kernel Same-page Merging (KSM).
  - Helps reduce memory footprint by deduplicating identical pages.

2.4 Question d

```
case KVM_EXIT_IO:
    if (vcpu->kvm_run->io.direction == KVM_EXIT_IO_OUT &&
        vcpu->kvm_run->io.port == 0xE9) {

        char *p = (char *)vcpu->kvm_run;
        fwrite(p + vcpu->kvm_run->io.data_offset,
               vcpu->kvm_run->io.size, 1, stdout);
        fflush(stdout);
        continue;
    }
```

Explanation:

- This code handles the `KVM_EXIT_IO` case in simple-kvm.c
- The condition checks if:
  - The virtual CPU (vCPU) is performing an `OUT` operation (`KVM_EXIT_IO_OUT`).
  - The I/O port being accessed is `0xE9`, which is often used as a debug output port in KVM-based systems.
- If the condition is met:
  - A pointer `p` is set to the start of the shared `kvm_run` structure.
  - The function `fwrite()` writes the data from the I/O operation to `stdout`.
  - The data is located at an offset of `io.data_offset` within `kvm_run`.
  - The size of the data is specified by `io.size`.
  - `fflush(stdout)` ensures immediate output.
- This mechanism is commonly used for debugging guest output, where the guest can send debug messages via the `0xE9` port, which are then printed to the host's terminal.

2.5 Question e

```
memcpy(&memval, &vm->mem[0x400], sz);
```

Explanation:

- This line copies a block of memory from the guest virtual machine's allocated memory (`vm->mem`) into the variable `memval`.
- The source address is `&vm->mem[0x400]`, which means it starts copying from the memory address offset `0x400` (1024 in decimal) within the virtual machine's memory.
- The destination is `&memval`, meaning the copied data will be stored in `memval`.
- The size of the copied data is specified by `sz`.
- This operation is typically used to read data from a specific memory location in the virtual machine.