Sandhya Chandramohan
CS 238P Homework #2
UCI Student ID: 61167891

Question 1: Explain why the _start label is assigned V2P_WO(entry)?

The output of `readelf -s kernel | grep _start` shows that the address of the _start symbol is `0010000c`.

And similarly, the output of `readelf -s kernel | grep entry` shows that the address of the entry label is `8010000c`.

The processor boots in physical address space (16-bit 8086 mode). As soon as it switches on 32-bit mode, segmentation hardware becomes active. Then the kernel enables paging. As soon as the paging hardware becomes active, all addresses are translated through the page tables. The kernel then switches its EIP and ESP registers to point to the kernel's virtual address space. From then on, the kernel operates purely out of its virtual address space (above KERNBASE), and the identity mapping (below KERNBASE) can be removed and used for user processes.

Start address is a physical address because at the time of branching to the start address, we are still operating in the physical address space. The first few instructions in the kernel (entry.S) execute from physical address space and initialize the page table and enable paging. After paging is enabled, the kernel will start executing from virtual address space.

The symbols in the entry code of the kernel (entry.S) were given virtual addresses (above 0x80000000) by the linker. However, when we jumped to the entry code from the bootloader, we ran it from its physical addresses.

`V2P_WO` as defined in `memlayout.h` is a macro that converts the given virtual address to a physical address without casts by subtracting the `KERNBASE` (first virtual address) `0x80000000` from it.

Therefore, the `_start` label is assigned `V2P_WO(entry)` so that the `V2P_WO` macro may convert the virtual address entry `8010000c` to physical address `0010000c` by subtracting `KERNBASE` from it, since paging has not been enabled at this point and the kernel is still executing in the physical address space.

Question 2: Why breaking on entry doesn't work?

```
(gdb) b entry
Breakpoint 1 at 0x8010000c: file entry.S, line 47.
(gdb) c
Continuing.
```

Breaking on `entry` does not work because the address of the entry label is `0x8010000c` as can be seen above, which is a virtual address above `KERNBASE`. But the kernel is still operating in the physical address space at that point. Only after paging is enabled via the instruction `movl %eax, %cr0` in line 57, the kernel will start executing from the virtual address space. The symbols in the entry code of the kernel (entry.S) were given virtual addresses (above 0x80000000) by the linker. However, when we jumped to the entry code from the bootloader, we ran it from its physical addresses. Therefore breaking does not work at `entry` in the `entry.S` file line 47.

Question 3: What is on the stack?

The output is:

```
(gdb) x/24x $esp
0x8010b580
<stack+4032>:  0x80112640     0x80106e6c     0x00000000     0x00000000
0x8010b590
<stack+4048>:  0x80112780     0x00010074     0x8010b5bc     0x80102e23
0x8010b5a0
<stack+4064>:  0x801154a8     0x80400000     0x00000000     0x00000000
0x8010b5b0
<stack+4080>:  0x00000000     0x00000000     0x00010074     0x00007bf8
0x8010b5c0 <bcache>:   0x00000000     0x00000000     0x00000000
        0x00000000
0x8010b5d0
<bcache+16>:   0x00000000     0x00000000     0x00000000     0x00000000
```

The call stack is divided up into contiguous pieces called stack frames. The frame contains the arguments given to the function, the function's local variables, and the address at which the function is executing.

```
(gdb) info stack
#0  kinit1 (vstart=0x801154a8, vend=vend@entry=0x80400000) at kalloc.c:36
#1  0x80102e23 in main () at main.c:20
```

- inner stack frame 0 -> kinit1 (vstart=0x801154a8, vend=vend@entry=0x80400000) at kalloc.c:36
- outer stack frame 1 -> 0x80102e23 in main () at main.c:20

The description of the call stack is as follows:

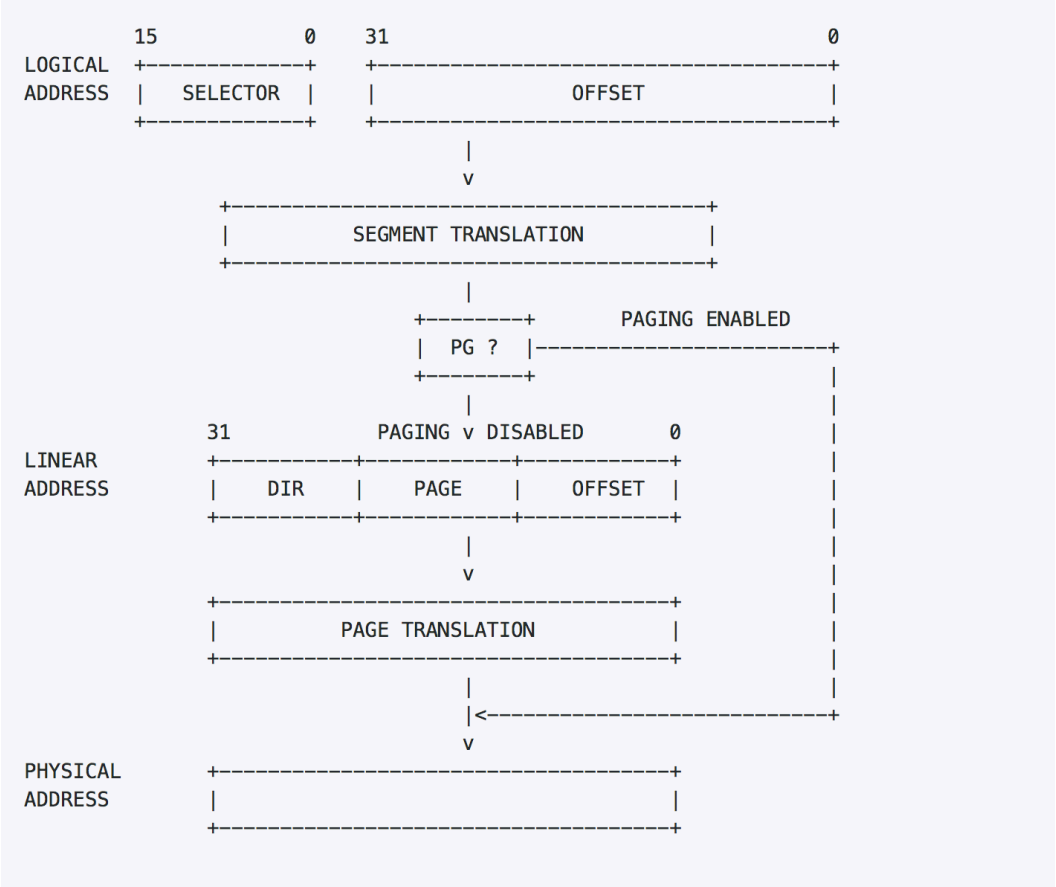
```
0x8010b580 <stack+4032>: 0x80112640 <--  Top of the stack, local variable
0x8010b584 <stack+4036>: 0x80106e6c <-- local variable

The next 2 bytes were allocated at the beginning of inner stack frame 0 to
be used as
scratch space for local variables and for passing arguments to functions
called

0x8010b588 <stack+4040>: 0x00000000 <--
0x8010b58c <stack+4044>: 0x00000000 <--
0x8010b590 <stack+4048>: 0x80112780 <-- EBX register saved in the inner
stack frame 0
0x8010b594 <stack+4052>: 0x00010074 <-- ESI register saved in the inner
stack frame 0
0x8010b598 <stack+4056>: 0x8010b5bc <-- address of local variables, EBP of
inner stack frame 0
0x8010b59c <stack+4060>: 0x80102e23 <-- return address of inner stack
frame 0
0x8010b5a0 <stack+4064>: 0x801154a8 <--  inner frame 0 argument vstart
0x8010b5a4 <stack+4068>: 0x80400000 <--  inner frame 0 argument vend

The next 4 bytes were allocated at the beginning of outer stack frame 0 to
be used as
scratch space for local variables and for passing arguments to functions
called

0x8010b5a8 <stack+4072>: 0x00000000 <--
0x8010b5ac <stack+4076>: 0x00000000 <--
0x8010b5b0 <stack+4080>: 0x00000000 <--
0x8010b5b4 <stack+4084>: 0x00000000 <--
0x8010b5b8 <stack+4088>: 0x00010074 <-- EBX register saved in the inner
stack frame 1
0x8010b5bc <stack+4092>: 0x00007bf8 <-- address of local variables, EBP of
outer stack frame 1
0x8010b5c0 <bcache>:   0x00000000 <-- return address of outer stack frame 1
0x8010b5c4 <bcache+4>: 0x00000000 <-- starting address of the outer stack
frame 1

Any addresses above this are not defined and not part of the stack.
```

Sandhya Chandramohan
CS 238P Homework #2
UCI Student ID: 61167891

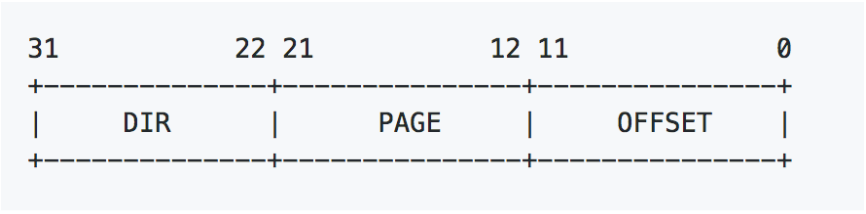
Question 2.1: Explain how logical to physical address translation works

Logical addresses are transformed to physical addresses in the following steps:

1. Segment translation: logical address -> linear address
2. Page Translation: linear address -> physical address



Format of linear address:



GDT that has a segment descriptor 1 configured to have the base of 0x1000, and your data segment register (ds selects this descriptor, i.e, your ds points to GDT entry 1).

The Base value in GDT entry 1 is: 0x1000
The logical address given is 0x803004
=> the linear address is: 0x804004 = 0000 0000 1000 0000 0100 0000 0000 0100
The first 10 bits point to entry 2 in the page directory at physical address 0x5000 stored in the CR3 register
Entry 2 in the page directory points to the entry physical address of the page table 0x8000.
The second 10 bits point to the entry 4 in the page table.
The PPN in the page table is 0x2000.
Therefore, the physical address is 20 bits of the PPN in the page table and the 12 bits of the offset.
=> physical address = 0000 0000 0000 0000 0010 0000 0000 0100
= 0x2004