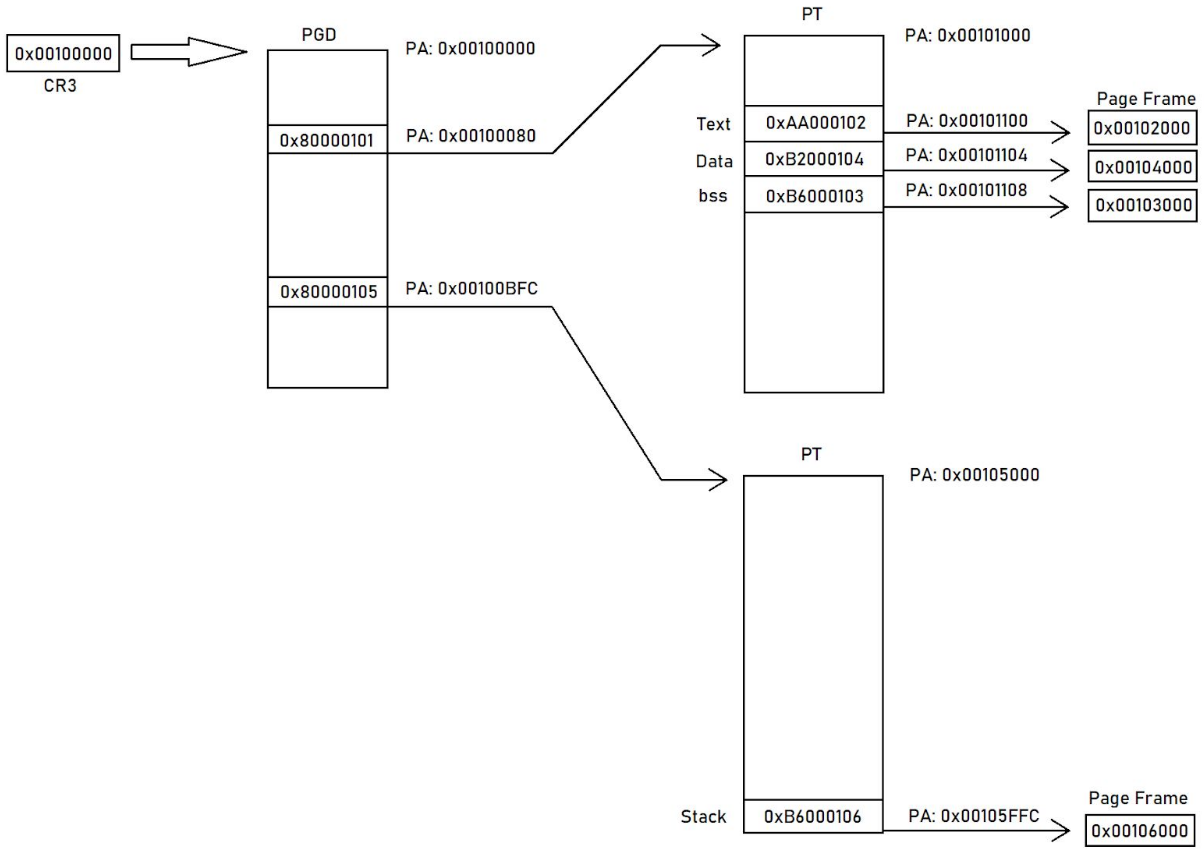
Derek Lee, Shine Li Computer Operating Systems Fall 2020

Professor Hakner PSET #5

Problem 1



PGD: Page Global Directory

PDE: Page Directory Entry

VA: Virtual Address

PA: Physical Address

PT: Page Table

PTE: Page Table Entry

PFN: Page Frame Number

1. The process is created. One page frame is allocated for the process' PGD. The lowest free page frame is at PA 0x00100000. This page frame is assigned to the PGD, and 0x00100000 is stored in register CR3.
2. The text region is created in the VA Space. It is given the VA 0x08040000. The most significant five bits of this address (0x08040) will be used to key the PTE. 0x08040 corresponds to 0b0000 1000 0000 0100 0000 (Spaces added for clarity). Since this architecture uses a 10/10 bit split, the most significant ten bits are taken: 0b0000 1000 00. This binary number corresponds to 32 in decimal, so the PFN of the PT containing the VA 0x08040000 can be found by looking at the 32nd PDE in the PGD. We also want to know the Physical Address of the 32nd PDE (in the PGD) - to do this, recall that each PDE takes up 4-bytes (or 32 bits), so the 32nd PDE starts at byte = 32*4 = 128. Convert to hexadecimal: 128 = 0x080. Since the PGD starts at PA 0x00100000, the PDE we are looking for starts at PA 0x00100080 (the PA of the PGD plus the 0x080 offset we found). However, a page frame has yet to be allocated for the PT containing the VA 0x08040000. The page frame for the PT will not be allocated until the program attempts to access a page frame corresponding to a PTE within that PT and has a page fault. Only then will a page frame be recursively allocated for this PT.
3. The data region is created in the Virtual Address Space next. It is to be contiguous with the text region. Since the text region has the VA 0x08040000, and is one page long, the data region will start at the VA 0x08041000. This is the next available space in the VA Space, as there are no guard zones between the memory regions. We examine the most significant 20-bits of the VA: 0x08041 = 0b0000 1000 0000 0100 0001 (spaces added for clarity). Notice that of these 20-bits, the most significant 10-bits (used to key the PT which will hold the PTE corresponding to this VA) is 0b0000 1000 00. This matches the most significant 10 bits of the text region, meaning they will be stored in the same PT. To reiterate, a page frame will not be allocated for this PT until a page fault is encountered while attempting to access a VA that is to be mapped through this PT.
4. The bss region is created next in the Virtual Address Space. Just as the data region was contiguous to the text region, the bss region is to be contiguous to the data region. Explanations for the bss region are similar to those of the text and data regions: the bss

region will start at the VA 0x08042000. It will be keyed to the same PT as the previous two regions. Once again, this PT has yet to be allocated a page frame at this time.

5. The stack is allocated at the highest VA for user processes (0xBFFFF000), and grows down toward lower addresses. However, it is assumed that the stack will not grow beyond 1 page. To find the PT that this VA is keyed to, examine the most significant 20-bits: 0xBFFFF = 0b1011 1111 1111 1111 1111 (spaces added for clarity). Of these, the most significant 10 bits are 0b1011 1111 11 = 767 in decimal. Thus, the PDE for the PT containing the PTE corresponding to the VA 0xBFFFF000 is the 767th entry in the PGD. Obviously, this is a different PT than the PT used in the text, data, and bss regions (the PT used for those three regions only covered the VAs from 0x08000000 to 0x083FFFFF). As in the case of the PT that is to contain the PTEs corresponding to the text, data, and bss regions, the PT that is to contain the PTE corresponding to the stack has not been allocated a page frame yet - for the same reason.
6. With the text, data, bss, and stack set up in Virtual Memory, the program can begin execution. The program attempts to fetch the first opcode from memory and execute it, but encounters a page fault. A page frame is demand-paged in for the text region. In order to do this, the kernel must first recursively allocate page frame(s) for the PT(s) that are involved in resolving the text region's VA to a PA. In this 2-level architecture, this means that a singular PT must have a page frame allocated to it. Since page frames are allocated continuously, the page frame allocated for the PT will be the next available page frame, located at the PA 0x00101000 (obtained by incrementing the least significant digit of the five most significant digits). With the page frame for the PT allocated, the PTE corresponding to the text region can be written to the PT. The 10 least significant bits of the 20 most significant bits of the VA of the text region are used to key the PTE inside the PT. These bits are 0b00 0100 0000, which translates to 64 in decimal, meaning the PTE of interest is the 64th entry in the PT. Each PTE takes up 4-bytes (or 32 bits), so the 64th entry starts at byte = 64 * 4 = 256. Convert to hexadecimal: 256 = 0x100. Therefore, the PTE containing the PFN corresponding to the VA of 0x08040000 is located at the PA 0x00101100 (obtained by adding 0x100 to the PA of the PT 0x00101000). Now that a page frame has been allocated to the PT, a page frame may now be allocated to the text memory region. Page frames are assumed to be allocated sequentially, and the next

available page frame is at the PA 0x00102000; this page frame is allocated to the text region. Regarding the flag bits for the PTE associated with text: the Present bit is set, as a page frame has been allocated for text (the Present bit will be set for every PDE and PTE that follow); the Supervisor bit is not set (0=user), as this process is assumed to be allowed to run in user mode (for this reason, the supervisor bit will not be set for every PDE and PTE that follow); the text region must have read and execute permissions, so those two access bits are set; the dirty bit is not set for the text region because it was not modified; the accessed bit will be set, as the text will need to be accessed while the program runs - the same can be said for the text, bss, and stack.

7. The first line of the program to be executed is “int b[40];”, which requires that space be reserved in the bss memory region. The program attempts to access the page frame corresponding to this region (zero-writing the entire region on first access), but encounters a page fault. By this point, a page frame has been assigned to the PT containing the PTE corresponding to the bss region. A page frame must still be allocated for the bss region. The next available page frame is at the PA 0x00103000; this page frame is allocated to the bss region. To find the PA of the PTE corresponding to the bss region examine the least significant 10 bits of the 20 most significant bits of the VA: 0b00 0100 0010. Notice that these bits are the equal to the latter 10-bits of the most significant 20-bits of the text region incremented by two. We can deduce that the PTE corresponding to the bss region is the PTE one PTE after the PTE corresponding to the text region (the PTE immediately following the PTE corresponding to the text region is the PTE corresponding to the data region). We also deduce that the PA of the PTE corresponding to the bss region is 0x00101108 (2*4-bytes after the PA of the PTE corresponding to the text region).We expect the bss region to fit within a single page, as the contents of the bss region are only: int b[40]; (we expect int to be 4 bytes), so no further page frames need be allocated for the bss region. The bss region will need read and write permissions, so these flag bits are set. The bss region will be dirty after the line “b[0] = d[3];” later in the program.
8. The next line of the program to be executed is “char d[] = “ABC123”;”, which requires that the data region be written to. The program attempts to access the page frame corresponding to this region, but encounters a page fault. By this point, a page frame has

been assigned to the PT containing the PTE corresponding to the data region. A page frame must still be allocated for the data region. The next available page frame is at the PA 0x00104000; this page frame is allocated to the data region. A similar procedure is used to find the PA of the PTE corresponding to the data region: 0x00101104. We expect the data region to fit within a single page, as the contents of the data region are only: char d[] = “ABC123”; (we expect each char to be 1 byte), so no further page frames need be allocated for the data region. The data region will need read and write permissions, so these flag bits are set.

9. The first function that is pushed onto the stack is main(); any C-libraries are ignored. The program attempts to push main() to the stack, but encounters a page fault. As of now, a page frame has yet to be allocated for the PT containing the PTE corresponding to the stack. The kernel recursively allocates a page frame for this PT. The next available page frame is at the PA 0x00105000; this page frame is allocated to the PT. A page frame can now be allocated to the stack region. The next available page frame is at the PA 0x00106000; this page frame is allocated to the stack. The PA of the PTE corresponding to the stack can be found in a similar manner to how the PAs of the PTEs corresponding to text, bss, and data were found: the resulting PA is 0x00105FFC. The stack requires read and write permissions; these bit flags are set. The stack will be written to as f1() is pushed on/popped off of it, so the dirty bit will be set.

Problem 2

a) No, it does not achieve true LRU behavior. Achieving true LRU behavior would require keeping track of the time at which each item was accessed. The Linux Page Frame Reclamation Algorithm (PFRA) approximates this, because the hardware only has a boolean ACCESSED bit.

   The PFRA scans page descriptors on its active list. If the ACCESSED bit is set, it is cleared. The PG_referenced flag is set to the previous value of the ACCESSED bit. When the PFRA revisits a page that had PG_referenced set to 0 and the ACCESSED bit(s) is still clear, the page is moved to the tail end of the inactive list.

Once the PFRA stops searching for pages in the active list to move to the inactive list, it pulls pages from the head of the inactive list. Thus, the pages that have been on the list for the longest will get pulled first, which approximates LRU behavior. The PFRA checks the page once more to see if its ACCESSED bit is still cleared. If it is no longer cleared, the page is moved back to the active list. Otherwise, the page is reclaimed.

b) The radix-64 tree is used to quickly find the page frame associated with a particular offset in a file that is mapped into memory.

   If the page frame is already opened (e.g. in another process), given an offset, the radix-64 tree can quickly find that page frame corresponding to that part of the file.

c) The page fault handler examines the task struct, which has a pointer to the mm_struct, which has an *mmap member. It checks the vm_area_structs that are pointed to by the *mmap member. If the fault address is above the kernel memory starting point (attempting to access kernel memory from user mode), it sends a SIGSEGV. This does not apply in this case, as the fault address is below. It examines the *mmap member of the mm_struct and the *vm_next, *vm_start, *vm_end, and vm_flags members of the vm_area_struct to check if the fault address is within a defined region. If the fault address is not within a defined region, it sends a SIGSEGV. The handler examines the vm_flags member of the vm_area_struct to determine if the fault is a protections fault. If it is within a defined region and the fault is a protections fault, it also sends a SIGSEGV.