**Part 2: Virtual Memory Simulation**

Seed 1

Base-and-Bounds register information:

*Base : 0x0000363c (decimal 13884)*

*Limit : 290*

*Virtual Address Trace*

*VA 0: 0x0000030e (decimal: 782) --> PA or segmentation violation?*

*VA 1: 0x00000105 (decimal: 261) --> PA or segmentation violation?*

*VA 2: 0x000001fb (decimal: 507) --> PA or segmentation violation?*

*VA 3: 0x000001cc (decimal: 460) --> PA or segmentation violation?*

*VA 4: 0x0000029b (decimal: 667) --> PA or segmentation violation?*

VA 0: out-of-bounds

VA 1: in-bounds; 14145

VA 2: out-of-bounds

VA 3: out-of-bounds

VA 4: out-of-bounds

Seed 2

*Base-and-Bounds register information:*

*Base : 0x00003ca9 (decimal 15529)*

*Limit : 500*

*Virtual Address Trace*

*VA 0: 0x00000039 (decimal: 57) --> PA or segmentation violation?*

*VA 1: 0x00000056 (decimal: 86) --> PA or segmentation violation?*

*VA 2: 0x00000357 (decimal: 855) --> PA or segmentation violation?*

*VA 3: 0x000002f1 (decimal: 753) --> PA or segmentation violation?*

*VA 4: 0x000002ad (decimal: 685) --> PA or segmentation violation?*

VA 0: in-bounds; 15586

VA 1: in-bounds; 15615

VA 2: out-of-bounds

VA 3: out-of-bounds

VA 4: out-of-bounds

Seed 3

*Base-and-Bounds register information:*

*Base : 0x000022d4 (decimal 8916)*

*Limit : 316*

*Virtual Address Trace*

*VA 0: 0x0000017a (decimal: 378) --> PA or segmentation violation?*

*VA 1: 0x0000026a (decimal: 618) --> PA or segmentation violation?*

*VA 2: 0x00000280 (decimal: 640) --> PA or segmentation violation?*

*VA 3: 0x00000043 (decimal: 67) --> PA or segmentation violation?*

*VA 4: 0x0000000d (decimal: 13) --> PA or segmentation violation?*

VA 0: out-of-bounds

VA 1: out-of-bounds

VA 2: out-of-bounds

VA 3: in-bounds; 8983

VA 4: in-bounds; 8929

-l should be set to a limit value of 930, as it’s just barely higher than the highest VA Trace value, 929. If the bounds register is set to 930, then all virtual addresses produced by the process/seed would be marked as valid.

The highest value that the base can be set to so that the address space, as restricted by the limit, fits into physical memory is 16284 (decimal). This base is based off the limit of 100 as specified by the limit parameter and is, therefore, 100 bytes less than the 16kb (16384 bytes) of physical memory size as specified in the program. If the base is set any higher than 15284, then a virtual address that’s too high and above the limit may try to access somewhere in physical memory that doesn’t exist/cannot be accessed. This way, only VA traces within both the limit and the physical memory size are considered valid. If the limit/bound were not an issue, though, then the highest base value would be 15360, as the entire address space of 1k (1024 bytes) that would need to be taken into consideration.

**Part 3: Glossary**

|  |  |
| --- | --- |
| Policy | In an OS, policies are the group of predetermined algorithms responsible for making decisions within the OS, such as process scheduling. These policies do not elaborate on how to perform a certain task but rather instruct what task to complete in the first place. These processes determine “which” function an OS may want to perform.  [Citation A](https://www.8bitavenue.com/policy-vs-mechanism-in-operating-system/)  [Citation B](https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-intro.pdf) |
| Mechanism | Mechanisms are responsible for carrying out the tasks set forth by the policy algorithms in the OS. These processes provide “how” an OS may perform a function through low-level machinery and intelligent programming. They implement functionality as specified by a higher-level policy.  [Citation C](https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-intro.pdf) |
| System Call | System calls are a set of functions, often in C, that specifically relate to the OS in a variety of ways that are, in a way, “higher-level” than other functions that may be defined within a given C program. System calls have predetermined functionality and are vital to how the OS handles processes in any computer.  [Citation D](https://www.geeksforgeeks.org/introduction-of-system-call/) |
| API | An Application Programming Interface, or API, is a virtual database of operations, functions, commands, data, and other useful operations provided by developers for other programmers to tinker/interact with. This promotes collaboration of ideas across platforms, companies, and communities.  [Citation E](https://techterms.com/definition/api) |
| File Descriptor | A file descriptor, commonly abbreviated as “fd” in programs, is an integer used to represent a specific file that has been opened within a program. Each file in a program has a unique file descriptor, otherwise the program could accidentally read from/write to the wrong file.  [Citation F](https://stackoverflow.com/questions/5256599/what-are-file-descriptors-explained-in-simple-terms) |
| Interrupt | An interrupt is a mechanism used by the CPU to stop one process and continue, temporarily, to another.  [Citation G](https://www.smspower.org/Development/InterruptMechanism#:~:text=Interrupts%20are%20a%20mechanism%20to,and%20temporarily%20switch%20to%20another.&text=This%20makes%20it%20mandatory%20to,active%2Dlow%20level%20sensitive%20input.) |
| Trap | A trap is an exception that is reported just after a trapping instruction is executed. Traps allow for the execution of a different process while preserving continuity for the current program.  [Citation H](https://stackoverflow.com/questions/3149175/what-is-the-difference-between-trap-and-interrupt) |
| Trap Table | A trap table stores the destinations that correspond to the trap’s instructions. It is accessed after the register state of one program is saved when a trap is called to preserve the place of the process in the program.  [Citation I](https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-mechanisms.pdf) |
| Stack Pointer | A stack is a small data set, or register, where addresses of the most recent program requests are stored in a Last In, First Out method. A stack pointer, on the other hand, refers specifically to the piece of data on the top of the stack, or the most recent entry into the stack.  [Citation J](https://www.quora.com/What-is-a-stack-pointer) |
| Kernel Mode | The Kernel Mode of a CPU is a state in which the CPU has universal, unrestricted access to the computer hardware, including memory. Kernel mode is implemented with incredible precision to prevent processes from accessing pieces of memory/parts of hardware//the computer that it shouldn’t for security reasons.  [Citation K](https://blog.codinghorror.com/understanding-user-and-kernel-mode/) |
| Kernel Stack | The Kernel Stack is used to keep track of which processes switch the CPU from User Mode to Kernel Mode through a stack datatype. Because it is in the Kernel space, it is not directly accessible when in User Mode or from a user process. When a system call is used, the CPU mode switches to Kernel, and the Kernel Stack of the running process is then used during that system call.  [Citation L](https://www.baeldung.com/linux/kernel-stack-and-user-space-stack#:~:text=The%20kernel%20stack%20is%20part,the%20running%20process%20is%20used.) |
| Standard Output | Standard output is where the main data stream from a process or program called from the command line is displayed. Standard output is often located in a terminal window where text can be easily printed to.  [Citation M](http://www.linfo.org/standard_output.html) |