Exercise 1

D) [OPTIONAL] Use an AI tool designed for assisting with programming and technical inquiries to help you read the code snippet between lines 378 and 397 of the following Linux kernel source code file Find information in Section 5.4.1 of the AMD64 manual to explain what is going on between lines 387 and 390.

5.4.1 Field Definitions

The following sections describe each field within the page-translation table entries.

Translation-Table Base Address Field The translation-table base-address field points to the physical base address of the next-lower-level table in the page-translation hierarchy. Page data-structure tables are always aligned on 4-Kbyte boundaries, so only the address bits above bit 11 are stored in the translation-table base-address field. Bits 11:0 are assumed to be 0. The size of the field depends on the mode:

- In normal (non-PAE) paging (CR4.PAE=0), this field specifies a 32-bit physical address.
- In PAE paging (CR4.PAE=1), this field specifies a 52-bit physical address.

52 bits correspond to the maximum physical-address size allowed by the AMD64 architecture. If a processor implementation supports fewer than the full 52-bit physical address, software must clear the unimplemented high-order translation-table base-address bits to 0. For example, if a processor implementation supports a 40-bit physical-address size, software must clear bits 51:40 when writing a translation-table base-address field in a page data-structure entry.

Physical-Page Base Address Field The physical-page base-address field points to the base address of the translated physical page. This field is found only in the lowest level of the page-translation hierarchy. The size of the field depends on the mode:

- In normal (non-PAE) paging (CR4.PAE=0), this field specifies a 32-bit base address for a physical page.
- In PAE paging (CR4.PAE=1), this field specifies a 52-bit base address for a physical page

Physical pages can be 4 Kbytes, 2 Mbytes, 4 Mbytes, or 1-Gbyte and they are always aligned on an address boundary corresponding to the physical-page length. For example, a 2-Mbyte physical page is always aligned on a 2-Mbyte address boundary. Because of this alignment, the low-order address bits are assumed to be 0, as follows:

- 4-Kbyte pages, bits 11:0 are assumed 0.
- 2-Mbyte pages, bits 20:0 are assumed 0.
- 4-Mbyte pages, bits 21:0 are assumed 0.
- 1-Gbyte pages, bits 29:0 are assumed 0.

Present (P) Bit Bit 0. This bit indicates whether the page-translation table or physical page is loaded in physical memory. When the P bit is cleared to 0, the table or physical page is not loaded in physical memory. When the P bit is set to 1, the table or physical page is loaded in physical memory.

Software clears this bit to 0 to indicate a page table or physical page is not loaded in physical memory. A page-fault exception (#PF) occurs if an attempt is made to access a table or page when the P bit is 0. System software is responsible for loading the missing table or page into memory and setting the P bit to 1.

When the P bit is 0, indicating a not-present page, all remaining bits in the page data-structure entry are available to software.

Entries with P cleared to 0 are never cached in TLB nor will the processor set the Accessed or Dirty bit for the table entry.

Read/Write (R/W) Bit Bit 1. This bit controls read/write access to all physical pages mapped by the table entry. For example, a page-map level-4 R/W bit controls read/write access to all 128M (512 x 512 x 512) physical pages it maps through the lower-level translation tables. When the R/W bit is cleared to 0, access is restricted to read-only. When the R/W bit is set to 1, both read and write access is allowed. See “Page-Protection Checks” on page 161 for a description of the paging read/write protection mechanism.

User/Supervisor (U/S) Bit Bit 2. This bit controls user (CPL 3) access to all physical pages mapped by the table entry. For example, a page-map level-4 U/S bit controls the access allowed to all 128M (512 x 512 x 512) physical pages it maps through the lower-level translation tables. When the U/S bit is cleared to 0, access is restricted to supervisor level (CPL 0, 1, 2). When the U/S bit is set to 1, both user and supervisor access is allowed. See “Page-Protection Checks” on page 161 for a description of the paging user/supervisor protection mechanism.

Page-Level Writethrough (PWT) Bit Bit 3. This bit indicates whether the page-translation table or physical page to which this entry points has a writeback or writethrough caching policy. When the PWT bit is cleared to 0, the table or physical page has a writeback caching policy. When the PWT bit is set to 1, the table or physical page has a writethrough caching policy. See “Memory Caches” on page 204 for additional information on caching.

Page-Level Cache Disable (PCD) Bit Bit 4. This bit indicates whether the page-translation table or physical page to which this entry points is cacheable. When the PCD bit is cleared to 0, the table or physical page is cacheable. When the PCD bit is set to 1, the table or physical page is not cacheable. See “Memory Caches” on page 204 for additional information on caching.

Accessed (A) Bit Bit 5. This bit indicates whether the page-translation table or physical page to which this entry points has been accessed. The A bit is set to 1 by the processor the first time the table or physical page is either read from or written to. The A bit is never cleared by the processor. Instead, software must clear this bit to 0 when it needs to track the frequency of table or physical-page accesses.

Dirty (D) Bit Bit 6. This bit is only present in the lowest level of the page-translation hierarchy. It indicates whether the physical page to which this entry points has been written. The D bit is set to 1 by the processor the first time there is a write to the physical page. The D bit is never cleared by the processor. Instead, software must clear this bit to 0 when it needs to track the frequency of physical-page writes.

Page Size (PS) Bit Bit 7. This bit is present in page-directory entries and long-mode page-directory-pointer entries. When the PS bit is set in the page-directory-pointer entry (PDPE) or page-directory entry (PDE), that entry is the lowest level of the page-translation hierarchy. When the PS bit is cleared to 0 in all levels above PTE, the lowest level of the page-translation hierarchy is the page-table entry (PTE), and the physical-page size is 4 Kbytes. The physical-page size is determined as follows:

- If EFER.LMA=1 and PDPE.PS=1, the physical-page size is 1 Gbyte.
- If CR4.PAE=0 and PDE.PS=1, the physical-page size is 4 Mbytes.
- If CR4.PAE=1 and PDE.PS=1, the physical-page size is 2 Mbytes.

See Table 5-1 on page 129 for a description of the relationship between the PS bit, PAE, physical-page sizes, and page-translation hierarchy.

Global Page (G) Bit. Bit 8. This bit is only present in the lowest level of the page-translation hierarchy. It indicates the physical page is a global page. The TLB entry for a global page (G=1) is not invalidated when CR3 is loaded either explicitly by a MOV CRn instruction or implicitly during a task switch. Use of the G bit requires the page-global enable bit in CR4 to be set to 1 (CR4.PGE=1). See "Global Pages" on page 158 for more information on the global-page mechanism.

Available to Software (AVL) Bit These bits are not interpreted by the processor and are available for use by system software.

Page-Attribute Table (PAT) Bit This bit is only present in the lowest level of the page-translation hierarchy, as follows:

- If the lowest level is a PTE (PDE.PS=0), PAT occupies bit 7.
- If the lowest level is a PDE (PDE.PS=1) or PDPE (PDPE.PS=1), PAT occupies bit 12.

The PAT bit is the high-order bit of a 3-bit index into the PAT register (Figure 7-13 on page 227). The other two bits involved in forming the index are the PCD and PWT bits. Not all processors support the PAT bit by implementing the PAT registers. See "Page-Attribute Table Mechanism" on page 227 for a description of the PAT mechanism and how it is used.

Memory Protection Key (MPK) Bits Bits 62:59. When Memory Protection Keys are enabled (CR4.PKE=1), this 4-bit field selects the memory protection key for the physical page mapped by this entry. Ignored if memory protection keys are disabled (CR4.PKE=0). (See "Memory Protection Keys (MPK) Bit" on page 164 for a description of this mechanism.)

No Execute (NX) Bit Bit 63. This bit is present in the translation-table entries defined for PAE paging, with the exception that the legacy-mode PDPE does not contain this bit. This bit is not supported by non-PAE paging.

The NX bit can only be set when the no-execute page-protection feature is enabled by setting EFER.NXE to 1 (see "Extended Feature Enable Register (EFER)" on page 55). If EFER.NXE=0, the NX bit is treated as reserved. In this case, a page-fault exception (#PF) occurs if the NX bit is not cleared to 0.

This bit controls the ability to execute code from all physical pages mapped by the table entry. For example, a page-map level-4 NX bit controls the ability to execute code from all 128M (512 x 512 x 512) physical pages it maps through the lower-level translation tables. When the NX bit is cleared to 0, code can be executed from the mapped physical pages. When the NX bit is set to 1, code cannot be executed from the mapped physical pages. See "No Execute (NX) Bit" on page 156 for a description of the no-execute page-protection mechanism.

Reserved Bits Software should clear all reserved bits to 0. If the processor is in long mode, or if page-size and physical-address extensions are enabled in legacy mode, a page-fault exception (#PF) occurs if reserved bits are not cleared to 0.

Kernel Code Excerpt

```
        /* Setup EFER (Extended Feature Enable Register) */
        movl    $MSR_EFER, %ecx
        rdmsr
        /*
        * Preserve current value of EFER for comparison and to skip
        * EFER writes if no change was made (for TDX guest)
        */
        movl    %eax, %edx
        btsl    $_EFER_SCE, %eax    /* Enable System Call */
        btl $20,%edi        /* No Execute supported? */
        jnc     1f
        btsl    $_EFER_NX, %eax
        btsq    $_PAGE_BIT_NX,early_pmd_flags(%rip)

        /* Avoid writing EFER if no change was made (for TDX guest) */
1:
        cmpl    %edx, %eax
        je  1f
        xor %edx, %edx
        wrmsr               /* Make changes effective */
1:
```

Análise do código Analysis

Este trecho de código está a configurar o Extended Feature Enable Register (EFER) no kernel do Linux. Este registo serve para ativar funcionalidades do sistema operativo, como:

- **SYSCALL / SYSRET** - intruções que permitem a passagem de *user mode* para *kernel mode* e vice-versa
- **No Execute (NX)** - bit de segurança que impede a execução de código nas áreas de memória onde o mesmo se encontra com o valor 1, ajudando assim a prevenir ataques de execução de código arbitrário
- **Long Mode** - para ativar o modo de 64 bits, permitindo tanto ao sistema operativo como às aplicações, o uso de endereços de memória a 64 bits.

Explicação passo-a-passo:

1. **Configuração / inicialização do EFER (Extended Feature Enable Register)**:

```
movl    $MSR_EFER, %ecx
rdmsr
```

- `movl $MSR_EFER, %ecx`: Carrega o endereço do *Load the Model-Specific Register (MSR)* para `EFER` no registo `ecx` .
- `rdmsr`: Lê o valor atual do `MSR EFER` para os registos `eax` e `edx`.

2. **Preservar o valor atual do *EFER* para comparação e evitar escritas no mesmo se nenhuma alteração for efetuada**:

```
movl    $eax, %edx
```

- Copia o value de `eax` (que tem os 32 bits da parte baixa do `EFER`) para `edx` para comparação posterior.

3. **Ativação de Chamadas de Sistema**:

```
btsl    $_EFER_SCE, %eax
```

- `btsl $_EFER_SCE, %eax` Define o bit de *Set the System Call Enable (SCE)* em `eax`. Isto ativa a instrução `SYSCALL`.

4. **Verificar se a funcionalidade *No Execute (NX)* é suportada**:

```
btl $20, %edi
jnc 1f
```

- `btl $20, %edi`: Testa o 20º bit do registo `edi` para verificar se existe suporte `NX`.
- `jnc 1f`: Salta para a *label* `1` caso `NX` não seja suportado (o 20º bit está a 0).

5. **Ativar *No Execute (NX)* se suportado**:

```
btsl    $_EFER_NX, %eax
btsq    $_PAGE_BIT_NX, early_pmd_flags(%rip)
```

- `btsl $_EFER_NX, %eax`: Define o bit `NX` no registo `eax` se `NX` for suportado.
- `btsq $_PAGE_BIT_NX, early_pmd_flags(%rip)`: Põe bit `NX` nas `flags` do diretório de páginas iniciais.

6. **Evitar a escrita no *EFER* se nenhuma alteração for efetuada**:

```
1:
cmpl %edx, %eax
je 1f
xor %edx, %edx
wrmsr
```

- `cmpl %edx, %eax`: Compara o valor original do `eax` (guardado em `edx`) com o `eax` modificado.
- `je 1f`: Salta para a *label* `1` se nenhuma alteração foi efetuada. (ou seja, se os valores forem iguais).
- `xor %edx, %edx`: Limpa o registo `edx` (mete-o a zeros).
- `wrmsr`: Escreve o valor modificado de `eax` e `edx` de volta para o `EFER MSR` para tornar as operações efetivas.

Resumo Neste código está a configurar o *EFER* para ativar a instrução `SYSCALL` e, se suportado, o bit de proteção `No Execute (NX)`. Lê o valor atual do `EFER`, modifica-o conforme necessário, e escreve-o de volta apenas se foram efetuadas alterações. Isto é de particular importancia para ambientes como *hosts TDX (Trusted Domain Extensions)*, onde escritas desnecessárias no `EFER` devem ser evitadas.

> Trusted Domain Extension (TDX) é uma tecnologia que permite a criação de ambientes de execução isolados e seguros dentro de uma máquina virtual.

Relação do código com os tópicos de *Paging* e *Segmentation*

Paging

Paging é o esquema de gestão do espaço de endereçamento virtual que elimina a necessidade de alocação contínua de memória física. Permite que o espaço de endereçamento físico de um processo seja "não contínuo", o que auxilia numa utilização eficiente da memória e na sua proteção.

1. **Bit No Execute (NX)**:
   - O bit `NX` faz parte das entradas da tabela de páginas e é usado para marcar páginas como não executáveis, proporcionando uma camada extra de segurança ao impedir a execução de código nas páginas onde o mesmo tem o valor 1.
2. **Page Fault Handling**:
   - Ao definir o bit `NX`, o sistema pode gerar `page faults` se existir uma tentativa de executar código a partir de uma página marcada como "não executável". Esta funcionalidade é crucial para impor

proteção na memória e prevenir certos tipos de ataques, como `buffer overflows`.

Segmentation

A segmentação é outro esquema de gestão de memória que divide a memória em diferentes segmentos, cada um com um propósito específico (por exemplo, .text, .data, stack).

1. **Bit System Call Enable (SCE)**:
   - O código define o bit `SCE` no `EFER` (`btsl $_EFER_SCE, %eax`), o que ativa a instrução `SYSCALL`.
   - A instrução `SYSCALL` é usada para fazer chamadas de sistema, que são uma forma de aplicações em *user mode* solicitarem serviços do *kernel*. Esta instrução depende da segmentação para alternar de forma segura do *user mode* para o *kernel mode*.
2. **Descriptores de Segment**:
   - Quando uma `SYSCALL` é executada, o CPU usa descriptores de *segment* para alterar para o código e `stack` apropriados para execução em *kernel mode*. Desta forma, garante-se que o *kernel* tenha o seu próprio espaço de memória protegido, separado das aplicações em *user mode*.

Leitura adicional / Mais informações

- CPU Registers
- Control Register
- AMD64 Architecture Programmer's Manual - Volume2: System Programming