Software Engineering

Lab-7

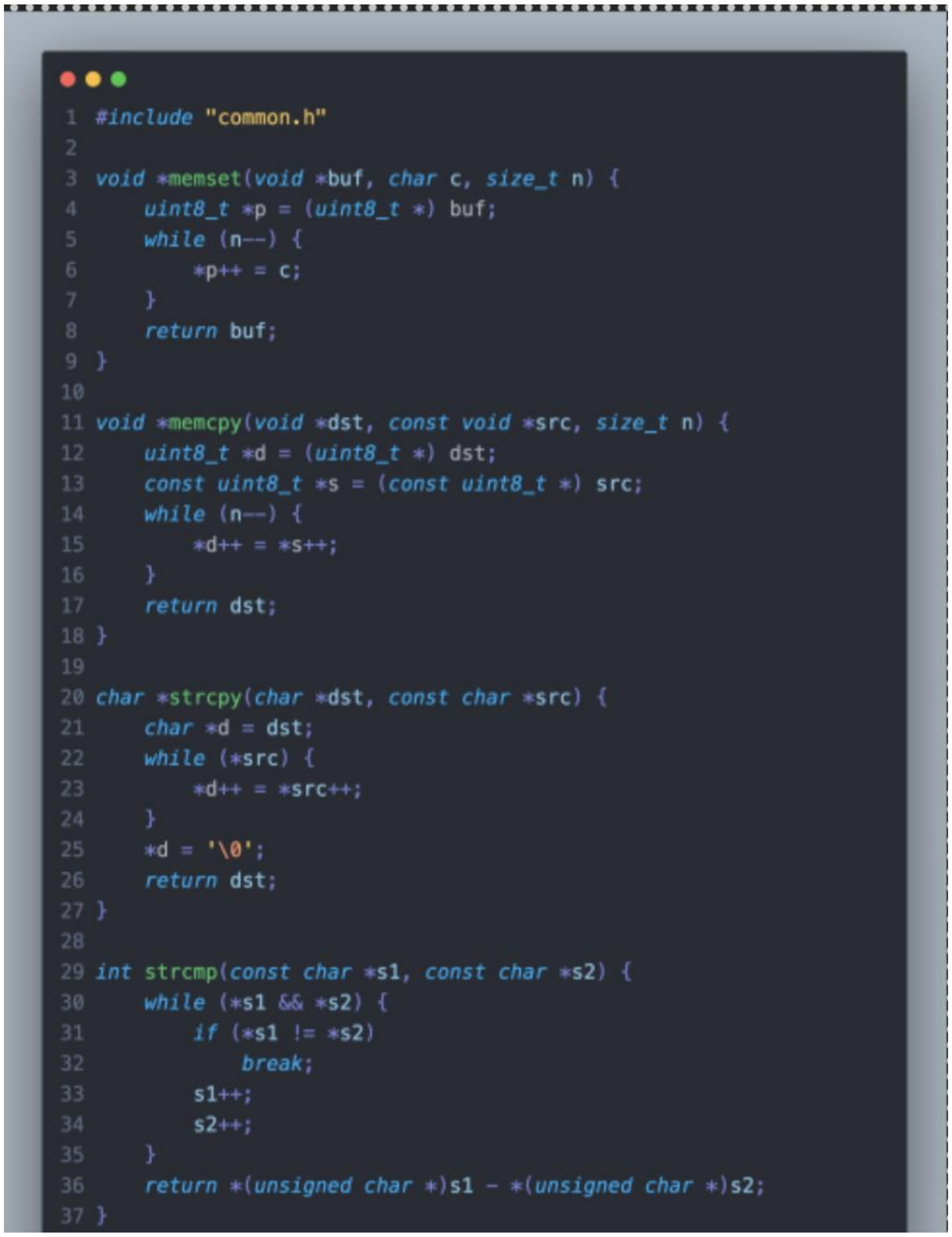
202201139

Krish S. Sagar

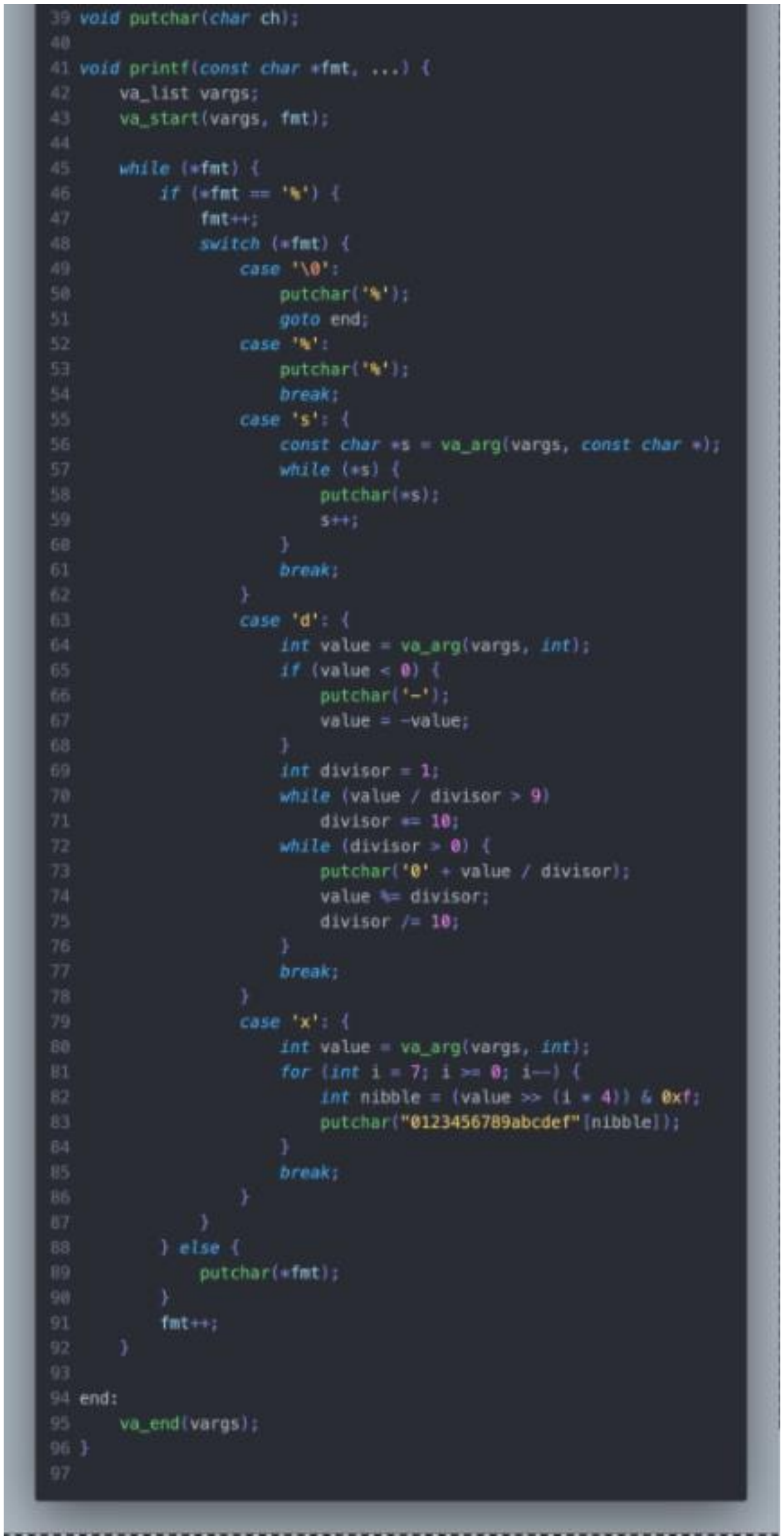
TASK – 1: Code Inspection and Debugging from 2000 Lines of Code from GitHub.

The GitHub Repo Link is

https://github.com/nuta/operating-system-in-1000-lines/tree/main



```c
#include "common.h"

void *memset(void *buf, char c, size_t n) {
    uint8_t *p = (uint8_t *) buf;
    while (n--) {
        *p++ = c;
    }
    return buf;
}

void *memcpy(void *dst, const void *src, size_t n) {
    uint8_t *d = (uint8_t *) dst;
    const uint8_t *s = (const uint8_t *) src;
    while (n--) {
        *d++ = *s++;
    }
    return dst;
}

char *strcpy(char *dst, const char *src) {
    char *d = dst;
    while (*src) {
        *d++ = *src++;
    }
    *d = '\0';
    return dst;
}

int strcmp(const char *s1, const char *s2) {
    while (*s1 && *s2) {
        if (*s1 != *s2)
            break;
        s1++;
        s2++;
    }
    return *(unsigned char *)s1 - *(unsigned char *)s2;
}
```



```c
void putchar(char ch);

void printf(const char *fmt, ...) {
    va_list vargs;
    va_start(vargs, fmt);

    while (*fmt) {
        if (*fmt == '%') {
            fmt++;
            switch (*fmt) {
                case '\0':
                    putchar('%');
                    goto end;
                case '%':
                    putchar('%');
                    break;
                case 's': {
                    const char *s = va_arg(vargs, const char *);
                    while (*s) {
                        putchar(*s);
                        s++;
                    }
                    break;
                }
                case 'd': {
                    int value = va_arg(vargs, int);
                    if (value < 0) {
                        putchar('-');
                        value = -value;
                    }
                    int divisor = 1;
                    while (value / divisor > 9)
                        divisor *= 10;
                    while (divisor > 0) {
                        putchar('0' + value / divisor);
                        value %= divisor;
                        divisor /= 10;
                    }
                    break;
                }
                case 'x': {
                    int value = va_arg(vargs, int);
                    for (int i = 7; i >= 0; i--) {
                        int nibble = (value >> (i * 4)) & 0xf;
                        putchar("0123456789abcdef"[nibble]);
                    }
                    break;
                }
            }
        } else {
            putchar(*fmt);
        }
        fmt++;
    }

end:
    va_end(vargs);
}
```

1. Data Referencing Errors:

- None found.

2. Data Declaration Errors:

- In the `printf` function, the `va_list vargs` is declared but not properly handled.

- If `va_end(vargs)` is called without a corresponding `va_start(vargs, fmt)`, it could lead to undefined behavior, although this isn't directly indicated here since `va_start` is correctly used before `va_end`.

3. Computation Errors:

- None found.

4. Comparison Errors:

- None found.

5. Control Flow Errors:

- In the `printf` function, the `goto end;` statement inside the switch block can create confusion.

- Although it is not an error, using `goto` can lead to less readable code and should be avoided if possible.

6. Interface Errors:

- The `putchar` function is declared but not defined in the provided code.

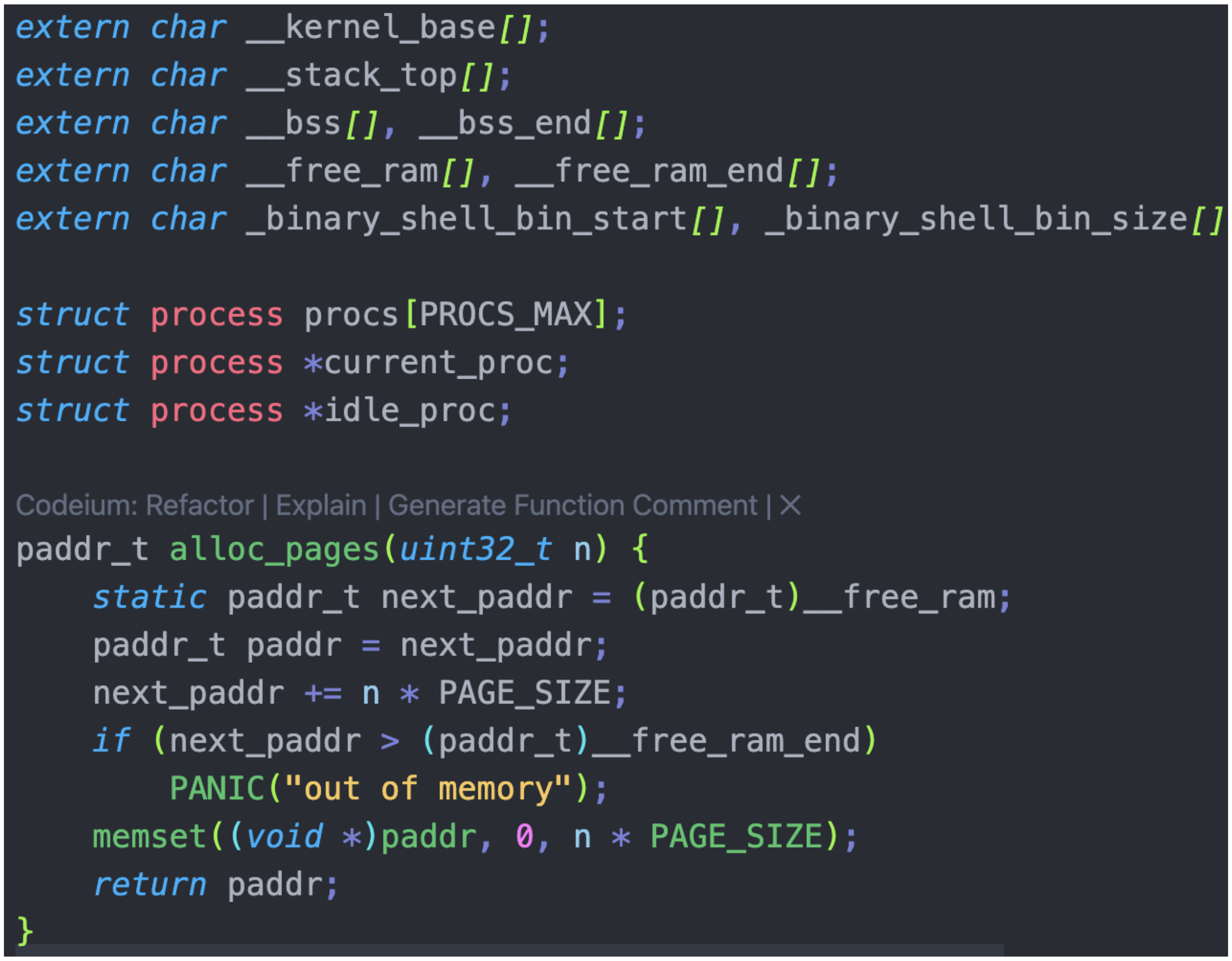
- This could lead to linker errors if `putchar` is called without a definition available.

- The `printf` function uses various formats (`%d`, `%x`, `%s`), but there is no error handling for unsupported formats, which could lead to unpredictable behavior if an unsupported format specifier is encountered.

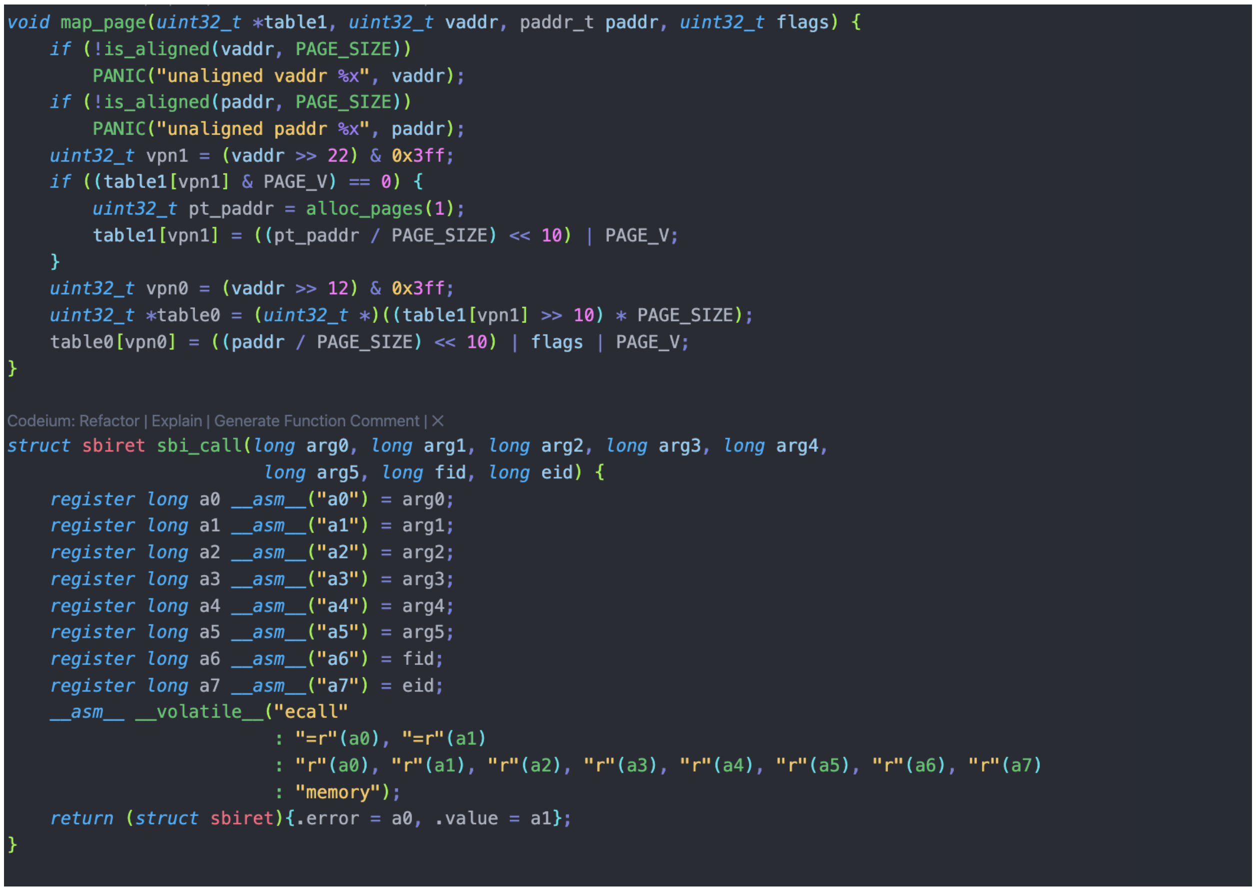
7. Input/Output Errors:

- In the `printf` function, there is no check for a null pointer in the `const char *s = va_arg(vargs, const char *);` line for the string format specifier (`%s`).

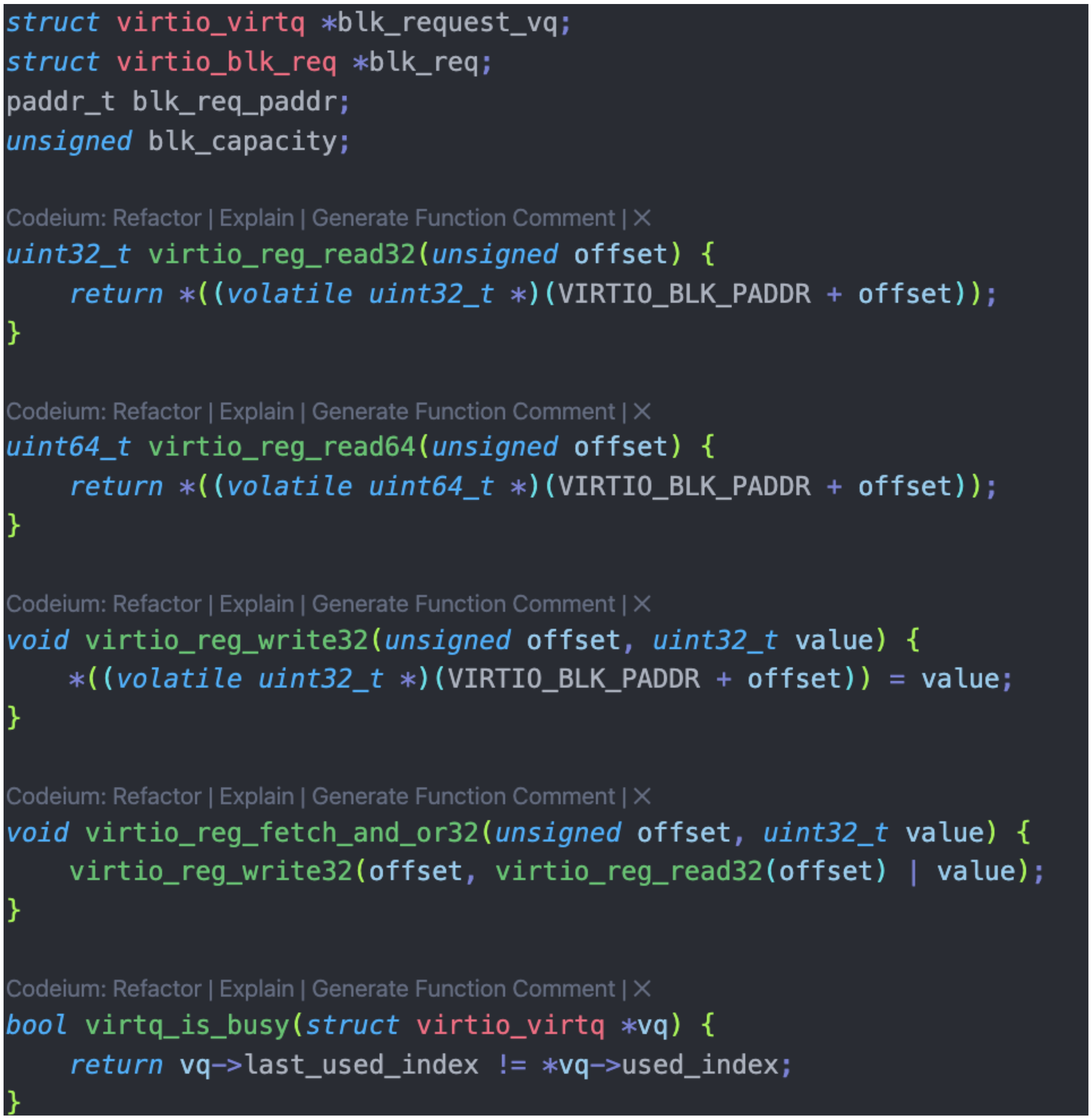
- If a null pointer is passed, it could lead to dereferencing a null pointer and cause a segmentation fault.



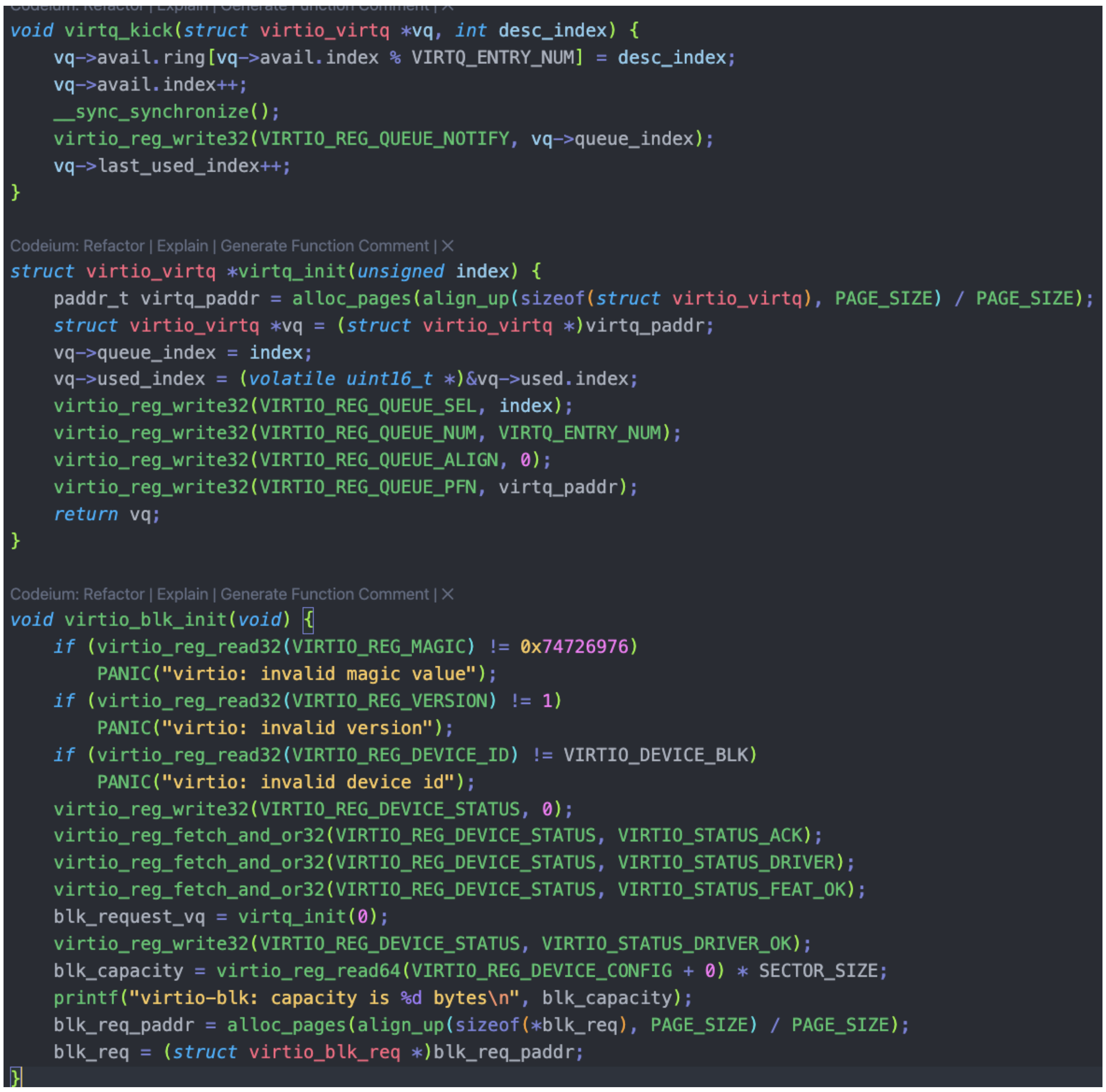
```
extern char __kernel_base[];
extern char __stack_top[];
extern char __bss[], __bss_end[];
extern char __free_ram[], __free_ram_end[];
extern char _binary_shell_bin_start[], _binary_shell_bin_size[]

struct process procs[PROCS_MAX];
struct process *current_proc;
struct process *idle_proc;

Codeium: Refactor | Explain | Generate Function Comment | X
paddr_t alloc_pages(uint32_t n) {
    static paddr_t next_paddr = (paddr_t)__free_ram;
    paddr_t paddr = next_paddr;
    next_paddr += n * PAGE_SIZE;
    if (next_paddr > (paddr_t)__free_ram_end)
        PANIC("out of memory");
    memset((void *)paddr, 0, n * PAGE_SIZE);
    return paddr;
}
```



```c
void map_page(uint32_t *table1, uint32_t vaddr, paddr_t paddr, uint32_t flags) {
    if (!is_aligned(vaddr, PAGE_SIZE))
        PANIC("unaligned vaddr %x", vaddr);
    if (!is_aligned(paddr, PAGE_SIZE))
        PANIC("unaligned paddr %x", paddr);
    uint32_t vpn1 = (vaddr >> 22) & 0x3ff;
    if ((table1[vpn1] & PAGE_V) == 0) {
        uint32_t pt_paddr = alloc_pages(1);
        table1[vpn1] = ((pt_paddr / PAGE_SIZE) << 10) | PAGE_V;
    }
    uint32_t vpn0 = (vaddr >> 12) & 0x3ff;
    uint32_t *table0 = (uint32_t *)((table1[vpn1] >> 10) * PAGE_SIZE);
    table0[vpn0] = ((paddr / PAGE_SIZE) << 10) | flags | PAGE_V;
}

Codeium: Refactor | Explain | Generate Function Comment | X
struct sbiret sbi_call(long arg0, long arg1, long arg2, long arg3, long arg4,
                       long arg5, long fid, long eid) {
    register long a0 __asm__("a0") = arg0;
    register long a1 __asm__("a1") = arg1;
    register long a2 __asm__("a2") = arg2;
    register long a3 __asm__("a3") = arg3;
    register long a4 __asm__("a4") = arg4;
    register long a5 __asm__("a5") = arg5;
    register long a6 __asm__("a6") = fid;
    register long a7 __asm__("a7") = eid;
    __asm__ __volatile__("ecall"
                         : "=r"(a0), "=r"(a1)
                         : "r"(a0), "r"(a1), "r"(a2), "r"(a3), "r"(a4), "r"(a5), "r"(a6), "r"(a7)
                         : "memory");
    return (struct sbiret){.error = a0, .value = a1};
}
```



```c
struct virtio_virtq *blk_request_vq;
struct virtio_blk_req *blk_req;
paddr_t blk_req_paddr;
unsigned blk_capacity;

Codeium: Refactor | Explain | Generate Function Comment | ×
uint32_t virtio_reg_read32(unsigned offset) {
    return *((volatile uint32_t *)(VIRTIO_BLK_PADDR + offset));
}

Codeium: Refactor | Explain | Generate Function Comment | ×
uint64_t virtio_reg_read64(unsigned offset) {
    return *((volatile uint64_t *)(VIRTIO_BLK_PADDR + offset));
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void virtio_reg_write32(unsigned offset, uint32_t value) {
    *((volatile uint32_t *)(VIRTIO_BLK_PADDR + offset)) = value;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void virtio_reg_fetch_and_or32(unsigned offset, uint32_t value) {
    virtio_reg_write32(offset, virtio_reg_read32(offset) | value);
}

Codeium: Refactor | Explain | Generate Function Comment | ×
bool virtq_is_busy(struct virtio_virtq *vq) {
    return vq->last_used_index != *vq->used_index;
}
```



```
void virtq_kick(struct virtio_virtq *vq, int desc_index) {
    vq->avail.ring[vq->avail.index % VIRTQ_ENTRY_NUM] = desc_index;
    vq->avail.index++;
    __sync_synchronize();
    virtio_reg_write32(VIRTIO_REG_QUEUE_NOTIFY, vq->queue_index);
    vq->last_used_index++;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
struct virtio_virtq *virtq_init(unsigned index) {
    paddr_t virtq_paddr = alloc_pages(align_up(sizeof(struct virtio_virtq), PAGE_SIZE) / PAGE_SIZE);
    struct virtio_virtq *vq = (struct virtio_virtq *)virtq_paddr;
    vq->queue_index = index;
    vq->used_index = (volatile uint16_t *)&vq->used.index;
    virtio_reg_write32(VIRTIO_REG_QUEUE_SEL, index);
    virtio_reg_write32(VIRTIO_REG_QUEUE_NUM, VIRTQ_ENTRY_NUM);
    virtio_reg_write32(VIRTIO_REG_QUEUE_ALIGN, 0);
    virtio_reg_write32(VIRTIO_REG_QUEUE_PFN, virtq_paddr);
    return vq;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void virtio_blk_init(void) {
    if (virtio_reg_read32(VIRTIO_REG_MAGIC) != 0x74726976)
        PANIC("virtio: invalid magic value");
    if (virtio_reg_read32(VIRTIO_REG_VERSION) != 1)
        PANIC("virtio: invalid version");
    if (virtio_reg_read32(VIRTIO_REG_DEVICE_ID) != VIRTIO_DEVICE_BLK)
        PANIC("virtio: invalid device id");
    virtio_reg_write32(VIRTIO_REG_DEVICE_STATUS, 0);
    virtio_reg_fetch_and_or32(VIRTIO_REG_DEVICE_STATUS, VIRTIO_STATUS_ACK);
    virtio_reg_fetch_and_or32(VIRTIO_REG_DEVICE_STATUS, VIRTIO_STATUS_DRIVER);
    virtio_reg_fetch_and_or32(VIRTIO_REG_DEVICE_STATUS, VIRTIO_STATUS_FEAT_OK);
    blk_request_vq = virtq_init(0);
    virtio_reg_write32(VIRTIO_REG_DEVICE_STATUS, VIRTIO_STATUS_DRIVER_OK);
    blk_capacity = virtio_reg_read64(VIRTIO_REG_DEVICE_CONFIG + 0) * SECTOR_SIZE;
    printf("virtio-blk: capacity is %d bytes\n", blk_capacity);
    blk_req_paddr = alloc_pages(align_up(sizeof(*blk_req), PAGE_SIZE) / PAGE_SIZE);
    blk_req = (struct virtio_blk_req *)blk_req_paddr;
}
```

1. Data Referencing Errors

- None identified.

2. Data Declaration Errors

- None identified.

3. Computation Errors

- None identified.

4. Comparison Errors

- None identified.

5. Control Flow Errors

- No check for successful allocation in `virtq_init()` after `alloc_pages()`.

This could lead to dereferencing a NULL pointer.

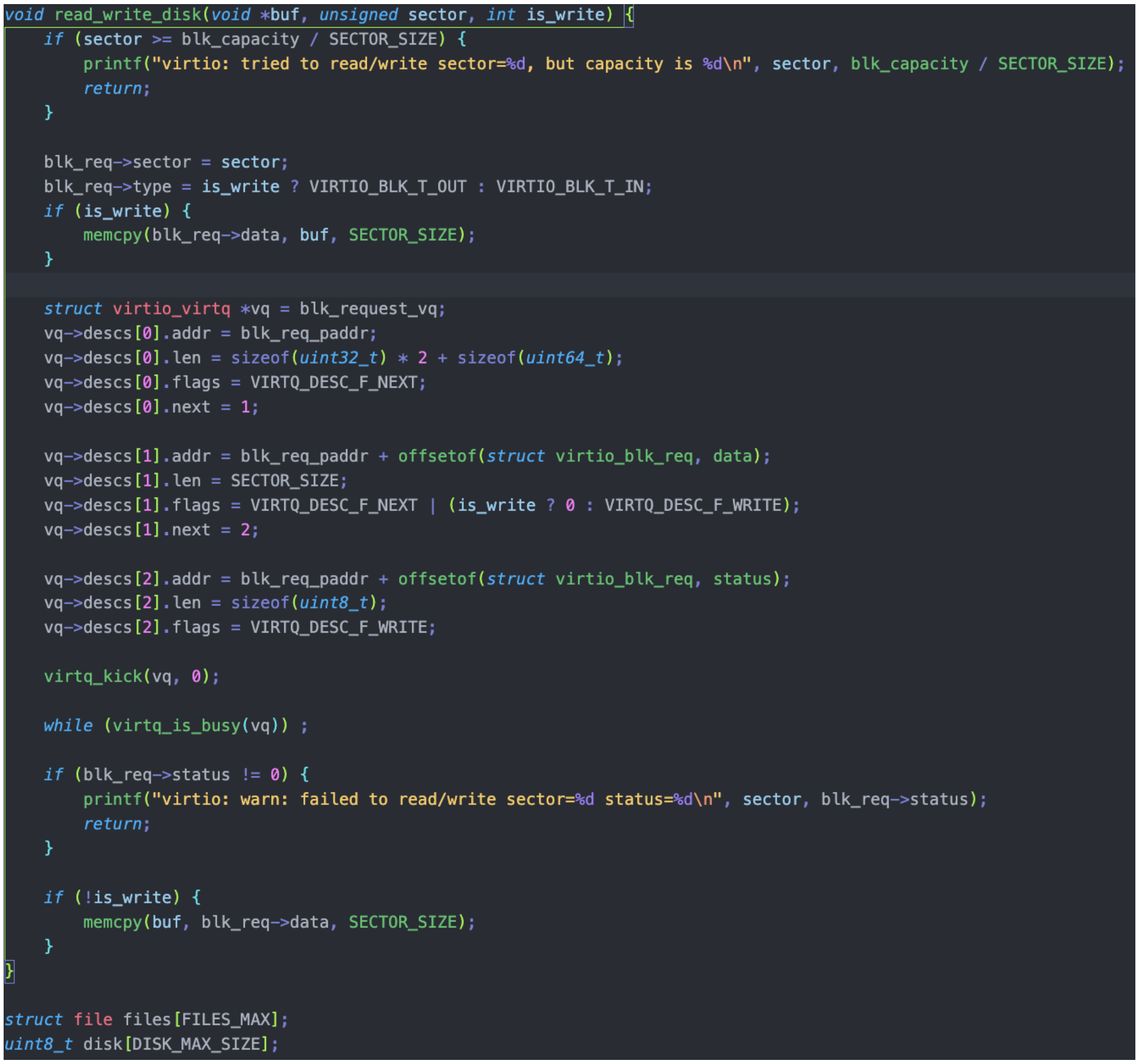
6. Interface Errors

- No explicit validation for register offsets in `virtio_reg_read32`,

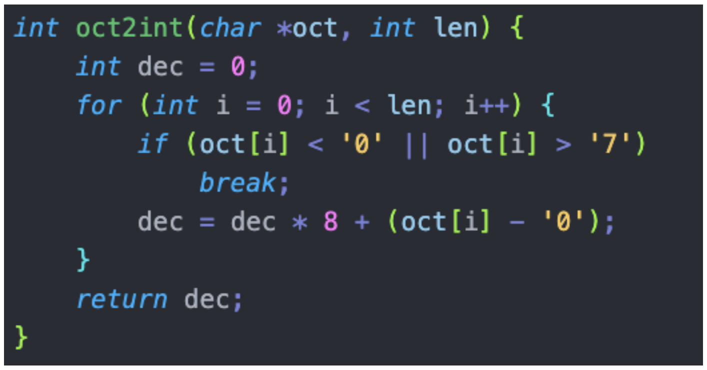
`virtio_reg_read64`, and related functions.

7. Input/Output Errors

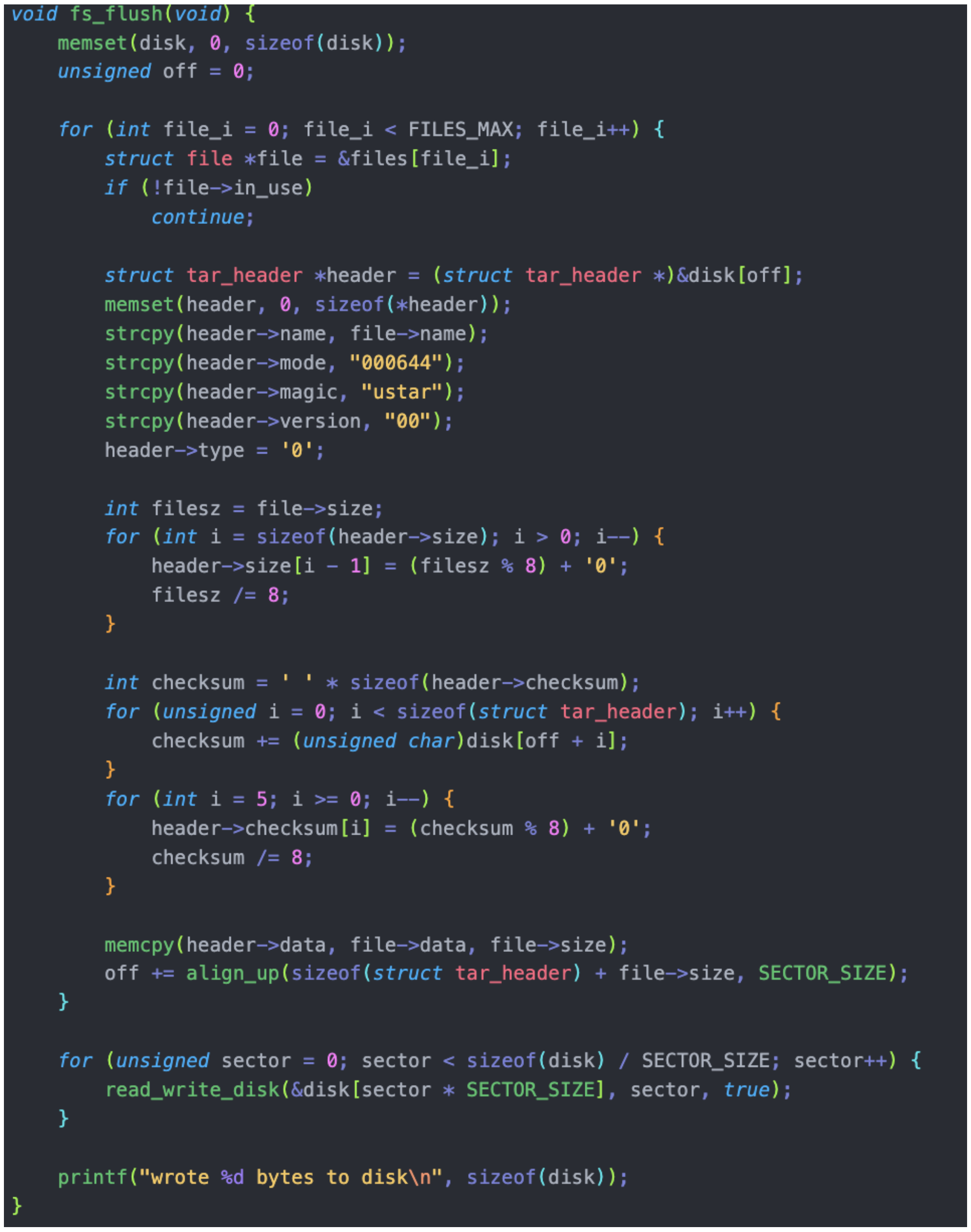
- None identified.



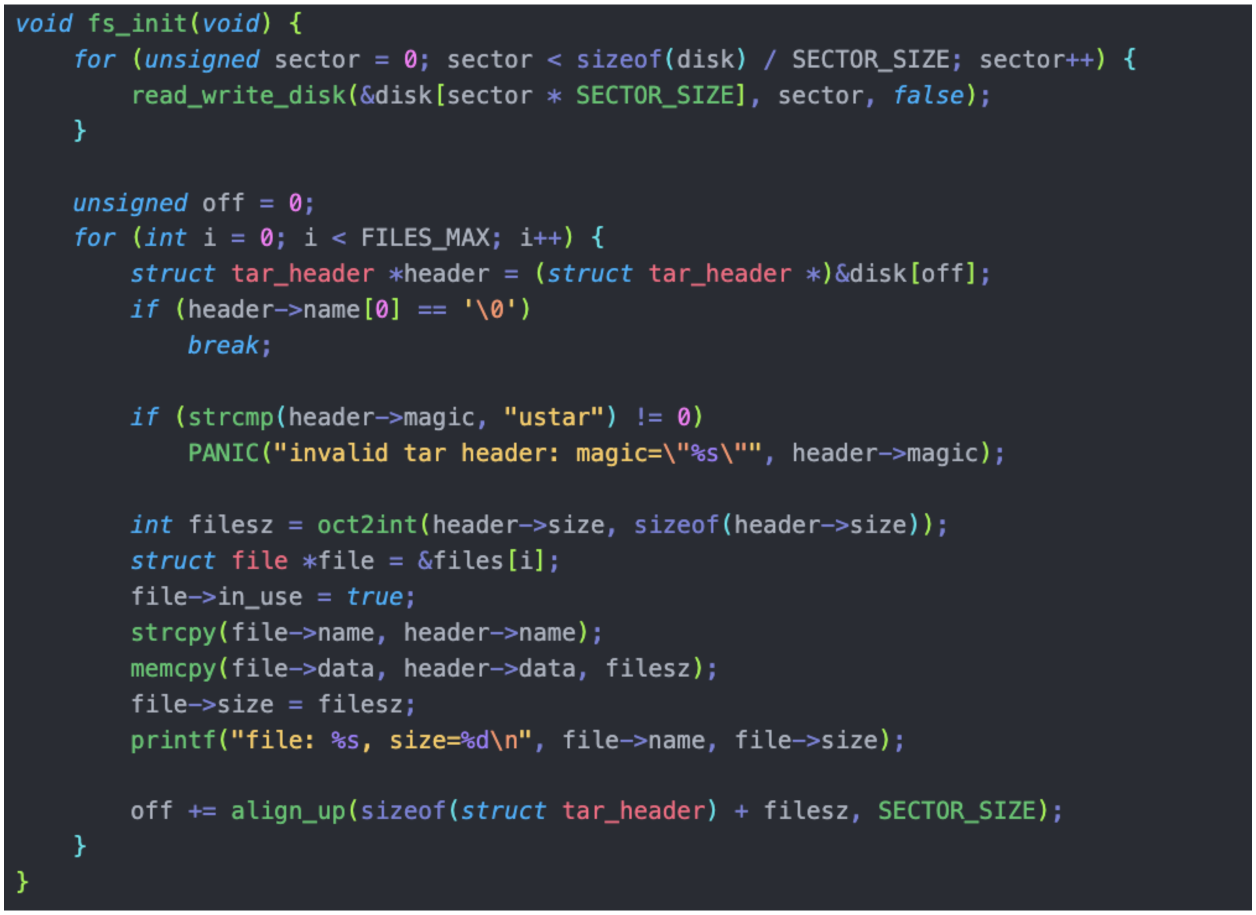
```c
void read_write_disk(void *buf, unsigned sector, int is_write) {
    if (sector >= blk_capacity / SECTOR_SIZE) {
        printf("virtio: tried to read/write sector=%d, but capacity is %d\n", sector, blk_capacity / SECTOR_SIZE);
        return;
    }

    blk_req->sector = sector;
    blk_req->type = is_write ? VIRTIO_BLK_T_OUT : VIRTIO_BLK_T_IN;
    if (is_write) {
        memcpy(blk_req->data, buf, SECTOR_SIZE);
    }

    struct virtio_virtq *vq = blk_request_vq;
    vq->descs[0].addr = blk_req_paddr;
    vq->descs[0].len = sizeof(uint32_t) * 2 + sizeof(uint64_t);
    vq->descs[0].flags = VIRTQ_DESC_F_NEXT;
    vq->descs[0].next = 1;

    vq->descs[1].addr = blk_req_paddr + offsetof(struct virtio_blk_req, data);
    vq->descs[1].len = SECTOR_SIZE;
    vq->descs[1].flags = VIRTQ_DESC_F_NEXT | (is_write ? 0 : VIRTQ_DESC_F_WRITE);
    vq->descs[1].next = 2;

    vq->descs[2].addr = blk_req_paddr + offsetof(struct virtio_blk_req, status);
    vq->descs[2].len = sizeof(uint8_t);
    vq->descs[2].flags = VIRTQ_DESC_F_WRITE;

    virtq_kick(vq, 0);

    while (virtq_is_busy(vq)) ;

    if (blk_req->status != 0) {
        printf("virtio: warn: failed to read/write sector=%d status=%d\n", sector, blk_req->status);
        return;
    }

    if (!is_write) {
        memcpy(buf, blk_req->data, SECTOR_SIZE);
    }
}

struct file files[FILES_MAX];
uint8_t disk[DISK_MAX_SIZE];
```



```c
int oct2int(char *oct, int len) {
    int dec = 0;
    for (int i = 0; i < len; i++) {
        if (oct[i] < '0' || oct[i] > '7')
            break;
        dec = dec * 8 + (oct[i] - '0');
    }
    return dec;
}
```



```c
void fs_flush(void) {
    memset(disk, 0, sizeof(disk));
    unsigned off = 0;

    for (int file_i = 0; file_i < FILES_MAX; file_i++) {
        struct file *file = &files[file_i];
        if (!file->in_use)
            continue;

        struct tar_header *header = (struct tar_header *)&disk[off];
        memset(header, 0, sizeof(*header));
        strcpy(header->name, file->name);
        strcpy(header->mode, "000644");
        strcpy(header->magic, "ustar");
        strcpy(header->version, "00");
        header->type = '0';

        int filesz = file->size;
        for (int i = sizeof(header->size); i > 0; i--) {
            header->size[i - 1] = (filesz % 8) + '0';
            filesz /= 8;
        }

        int checksum = ' ' * sizeof(header->checksum);
        for (unsigned i = 0; i < sizeof(struct tar_header); i++) {
            checksum += (unsigned char)disk[off + i];
        }
        for (int i = 5; i >= 0; i--) {
            header->checksum[i] = (checksum % 8) + '0';
            checksum /= 8;
        }

        memcpy(header->data, file->data, file->size);
        off += align_up(sizeof(struct tar_header) + file->size, SECTOR_SIZE);
    }

    for (unsigned sector = 0; sector < sizeof(disk) / SECTOR_SIZE; sector++) {
        read_write_disk(&disk[sector * SECTOR_SIZE], sector, true);
    }

    printf("wrote %d bytes to disk\n", sizeof(disk));
}
```



```c
void fs_init(void) {
    for (unsigned sector = 0; sector < sizeof(disk) / SECTOR_SIZE; sector++) {
        read_write_disk(&disk[sector * SECTOR_SIZE], sector, false);
    }

    unsigned off = 0;
    for (int i = 0; i < FILES_MAX; i++) {
        struct tar_header *header = (struct tar_header *)&disk[off];
        if (header->name[0] == '\0')
            break;

        if (strcmp(header->magic, "ustar") != 0)
            PANIC("invalid tar header: magic=\"%s\"", header->magic);

        int filesz = oct2int(header->size, sizeof(header->size));
        struct file *file = &files[i];
        file->in_use = true;
        strcpy(file->name, header->name);
        memcpy(file->data, header->data, filesz);
        file->size = filesz;
        printf("file: %s, size=%d\n", file->name, file->size);

        off += align_up(sizeof(struct tar_header) + filesz, SECTOR_SIZE);
    }
}
```

1. Data Referencing Errors

- The code references `blk_req`, `blk_capacity`, `blk_request_vq`, and `blk_req_paddr` without showing their definitions. Ensure these variables are properly initialized and referenced.

2. Data Declaration Errors

- The variable `disk` is declared with `uint8_t disk[DISK_MAX_SIZE];`, but there's no indication of the value assigned to `DISK_MAX_SIZE`. Ensure it's defined somewhere.
- The `struct tar_header` is referenced without a declaration in the provided code. Ensure it is defined correctly in your project.

3. Computation Errors

- The calculation of `filesz` in `fs_flush` does not account for potential overflow when calculating the checksum. Although the tar format specifies a maximum size, it’s good practice to check sizes to avoid overflow.
- In the `oct2int` function, if the input `oct` string has more than three characters (which represent a valid octal digit), the conversion might give unexpected results. Consider adding a limit on `len`.

4. Comparison Errors

- In `fs_init`, the check `if (strcmp(header->magic, "ustar") != 0)` is valid, but the code doesn't handle the case where `header->magic` could be `NULL`. Consider adding a `NULL` check before comparison.

5. Control Flow Errors

- The `read_write_disk` function might enter an infinite loop if the disk request is never completed. Ensure that `virtq_kick(vq, 0)` and `virtq_is_busy(vq)` are implemented correctly to handle this situation.
- The function `fs_flush` will print that it has written to the disk regardless of whether the write was successful. Consider checking for errors in `read_write_disk`.

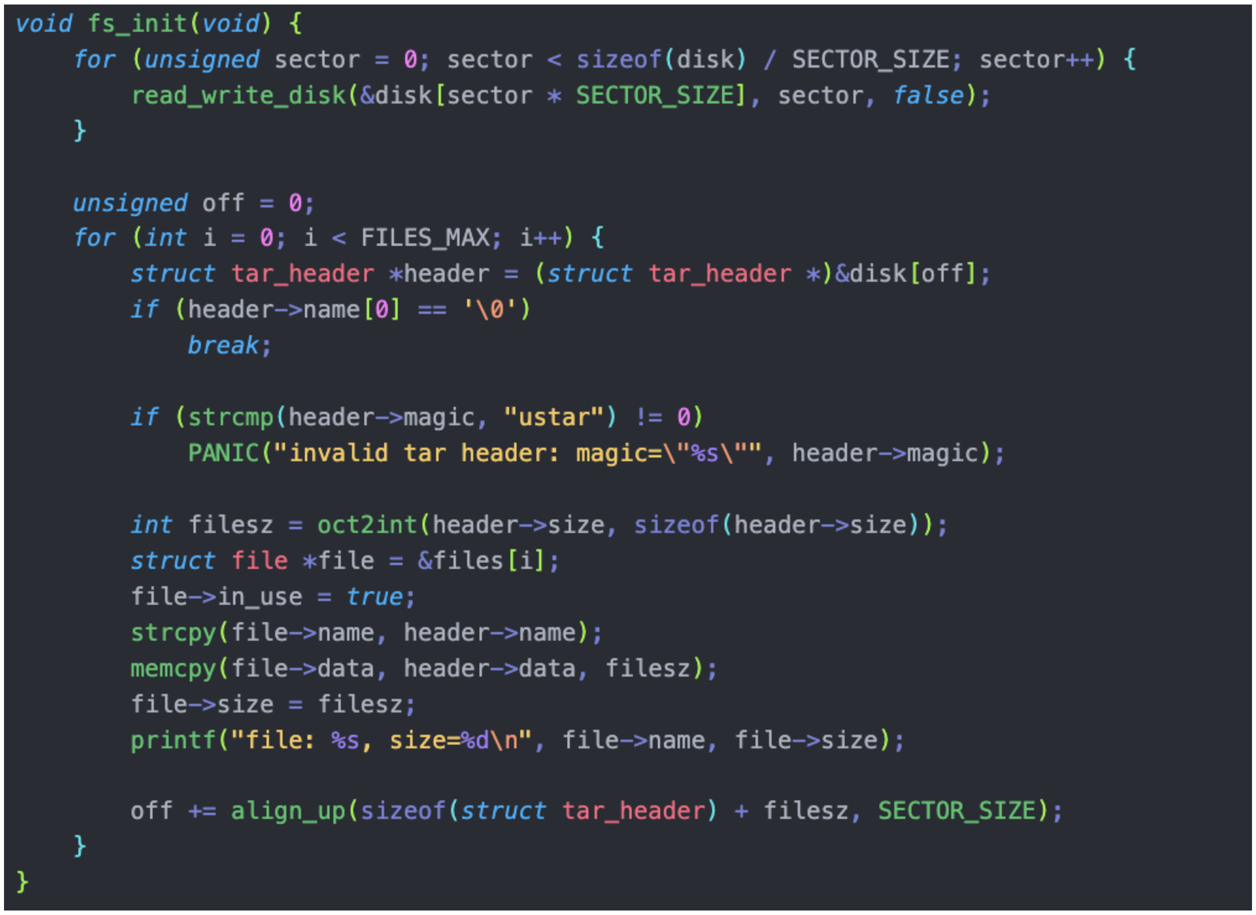
6. Interface Errors

- The `putchar` function must be defined elsewhere, or else there will be linking errors when compiling.
- Ensure that `align_up` is properly defined and that its purpose is clear; it seems to be intended for aligning data sizes, but its implementation is not

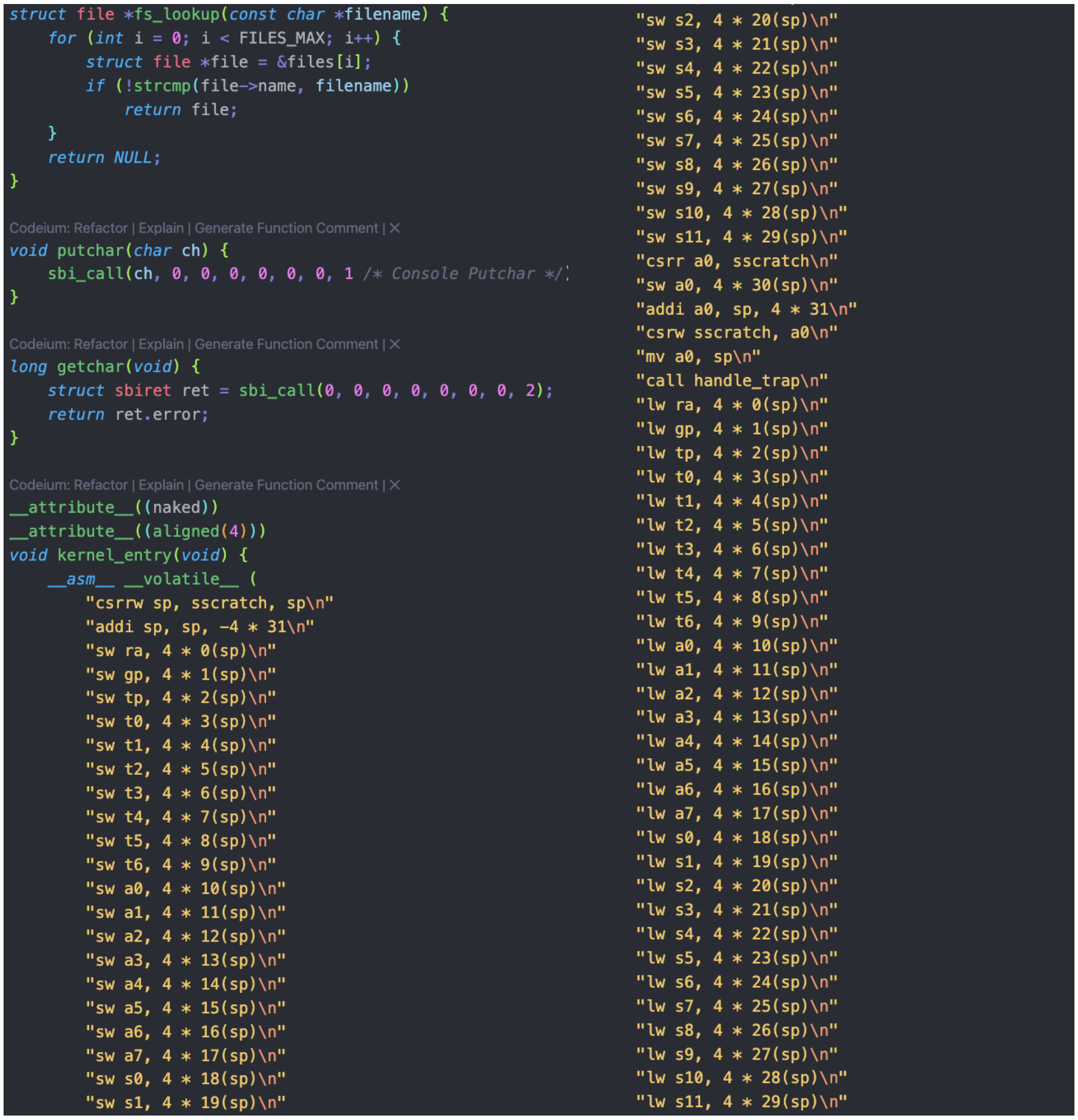
provided here.

7. Input/Output Errors

- In the `fs_flush` function, when writing to the disk, if `read_write_disk` fails for any reason (e.g., due to a full disk or hardware failure), the user is not notified. Implement error handling to manage this.
- In the `fs_init` function, if the data read from the disk doesn’t match the expected format or the file size exceeds `DISK_MAX_SIZE`, it may cause out-of-bounds memory access when populating the `file` structures.

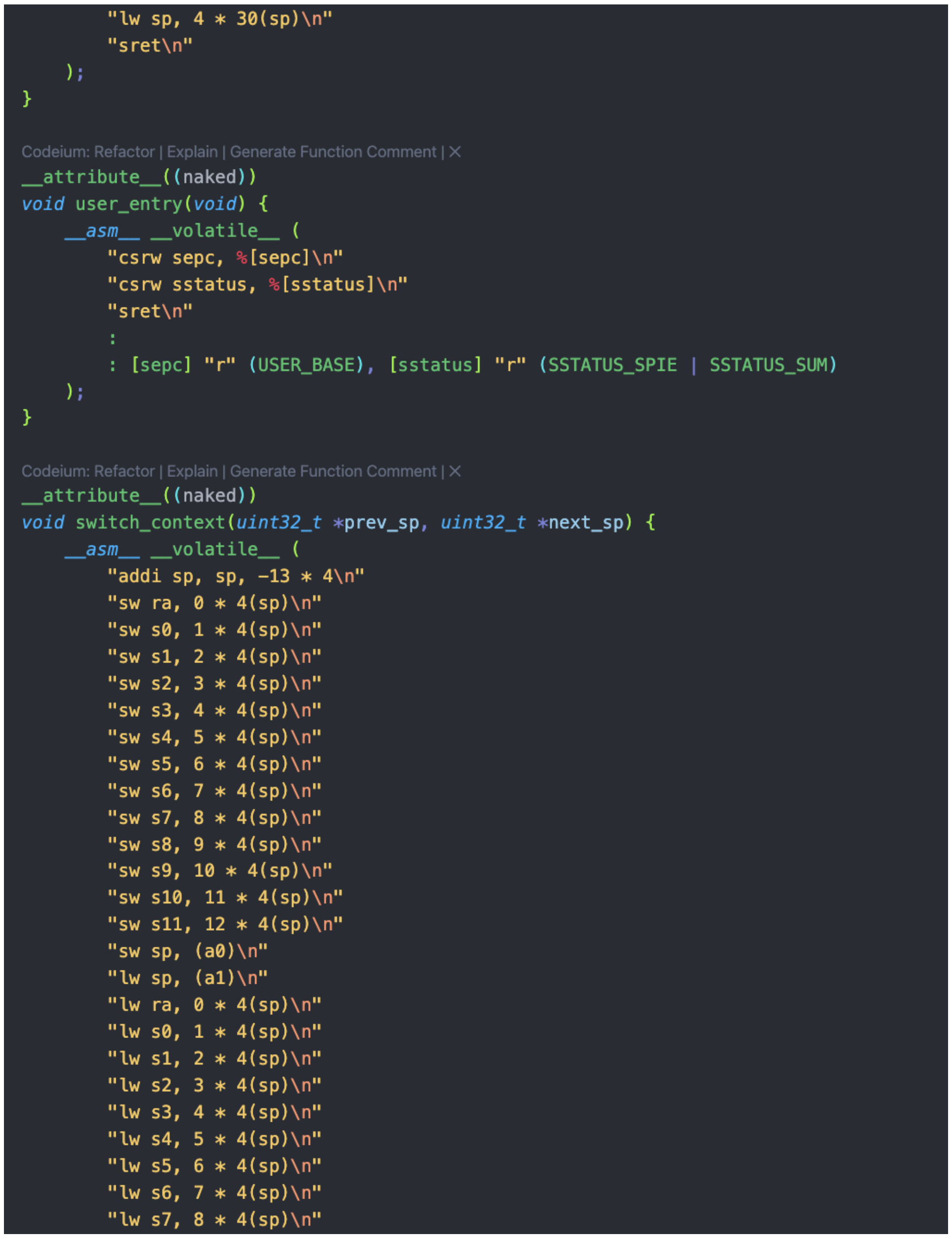


```
void fs_init(void) {
    for (unsigned sector = 0; sector < sizeof(disk) / SECTOR_SIZE; sector++) {
        read_write_disk(&disk[sector * SECTOR_SIZE], sector, false);
    }

    unsigned off = 0;
    for (int i = 0; i < FILES_MAX; i++) {
        struct tar_header *header = (struct tar_header *)&disk[off];
        if (header->name[0] == '\0')
            break;

        if (strcmp(header->magic, "ustar") != 0)
            PANIC("invalid tar header: magic=\"%s\"", header->magic);

        int filesz = oct2int(header->size, sizeof(header->size));
        struct file *file = &files[i];
        file->in_use = true;
        strcpy(file->name, header->name);
        memcpy(file->data, header->data, filesz);
        file->size = filesz;
        printf("file: %s, size=%d\n", file->name, file->size);

        off += align_up(sizeof(struct tar_header) + filesz, SECTOR_SIZE);
    }
}
```

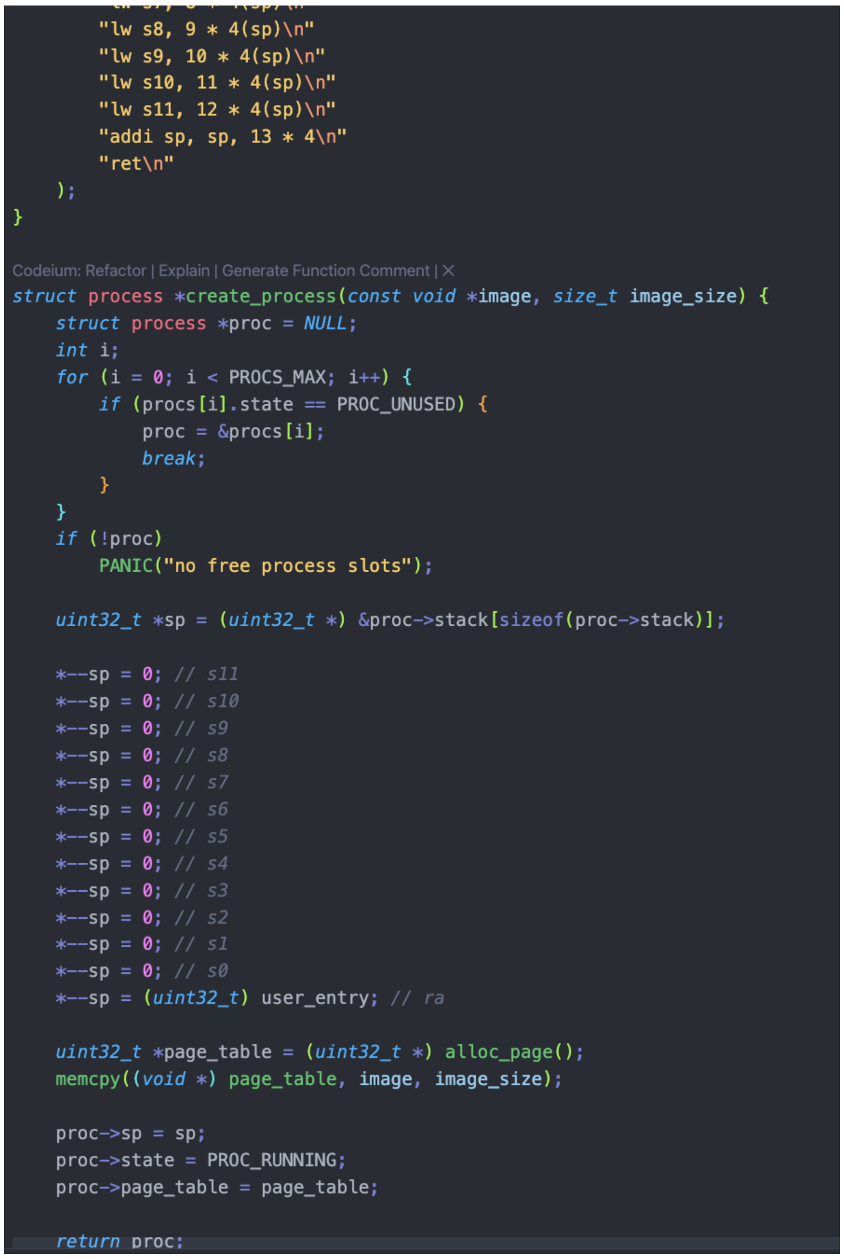


```c
struct file *fs_lookup(const char *filename) {
    for (int i = 0; i < FILES_MAX; i++) {
        struct file *file = &files[i];
        if (!strcmp(file->name, filename))
            return file;
    }
    return NULL;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void putchar(char ch) {
    sbi_call(ch, 0, 0, 0, 0, 0, 0, 1 /* Console Putchar */
}

Codeium: Refactor | Explain | Generate Function Comment | ×
long getchar(void) {
    struct sbiret ret = sbi_call(0, 0, 0, 0, 0, 0, 0, 2);
    return ret.error;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
__attribute__((naked))
__attribute__((aligned(4)))
void kernel_entry(void) {
    __asm__ __volatile__ (
        "csrrw sp, sscratch, sp\n"
        "addi sp, sp, -4 * 31\n"
        "sw ra, 4 * 0(sp)\n"
        "sw gp, 4 * 1(sp)\n"
        "sw tp, 4 * 2(sp)\n"
        "sw t0, 4 * 3(sp)\n"
        "sw t1, 4 * 4(sp)\n"
        "sw t2, 4 * 5(sp)\n"
        "sw t3, 4 * 6(sp)\n"
        "sw t4, 4 * 7(sp)\n"
        "sw t5, 4 * 8(sp)\n"
        "sw t6, 4 * 9(sp)\n"
        "sw a0, 4 * 10(sp)\n"
        "sw a1, 4 * 11(sp)\n"
        "sw a2, 4 * 12(sp)\n"
        "sw a3, 4 * 13(sp)\n"
        "sw a4, 4 * 14(sp)\n"
        "sw a5, 4 * 15(sp)\n"
        "sw a6, 4 * 16(sp)\n"
        "sw a7, 4 * 17(sp)\n"
        "sw s0, 4 * 18(sp)\n"
        "sw s1, 4 * 19(sp)\n"
```

```
        "sw s2, 4 * 20(sp)\n"
        "sw s3, 4 * 21(sp)\n"
        "sw s4, 4 * 22(sp)\n"
        "sw s5, 4 * 23(sp)\n"
        "sw s6, 4 * 24(sp)\n"
        "sw s7, 4 * 25(sp)\n"
        "sw s8, 4 * 26(sp)\n"
        "sw s9, 4 * 27(sp)\n"
        "sw s10, 4 * 28(sp)\n"
        "sw s11, 4 * 29(sp)\n"
        "csrr a0, sscratch\n"
        "sw a0, 4 * 30(sp)\n"
        "addi a0, sp, 4 * 31\n"
        "csrw sscratch, a0\n"
        "mv a0, sp\n"
        "call handle_trap\n"
        "lw ra, 4 * 0(sp)\n"
        "lw gp, 4 * 1(sp)\n"
        "lw tp, 4 * 2(sp)\n"
        "lw t0, 4 * 3(sp)\n"
        "lw t1, 4 * 4(sp)\n"
        "lw t2, 4 * 5(sp)\n"
        "lw t3, 4 * 6(sp)\n"
        "lw t4, 4 * 7(sp)\n"
        "lw t5, 4 * 8(sp)\n"
        "lw t6, 4 * 9(sp)\n"
        "lw a0, 4 * 10(sp)\n"
        "lw a1, 4 * 11(sp)\n"
        "lw a2, 4 * 12(sp)\n"
        "lw a3, 4 * 13(sp)\n"
        "lw a4, 4 * 14(sp)\n"
        "lw a5, 4 * 15(sp)\n"
        "lw a6, 4 * 16(sp)\n"
        "lw a7, 4 * 17(sp)\n"
        "lw s0, 4 * 18(sp)\n"
        "lw s1, 4 * 19(sp)\n"
        "lw s2, 4 * 20(sp)\n"
        "lw s3, 4 * 21(sp)\n"
        "lw s4, 4 * 22(sp)\n"
        "lw s5, 4 * 23(sp)\n"
        "lw s6, 4 * 24(sp)\n"
        "lw s7, 4 * 25(sp)\n"
        "lw s8, 4 * 26(sp)\n"
        "lw s9, 4 * 27(sp)\n"
        "lw s10, 4 * 28(sp)\n"
        "lw s11, 4 * 29(sp)\n"
```



```
        "lw sp, 4 * 30(sp)\n"
        "sret\n"
    );
}

Codeium: Refactor | Explain | Generate Function Comment | ×
__attribute__((naked))
void user_entry(void) {
    __asm__ __volatile__ (
        "csrw sepc, %[sepc]\n"
        "csrw sstatus, %[sstatus]\n"
        "sret\n"
        :
        : [sepc] "r" (USER_BASE), [sstatus] "r" (SSTATUS_SPIE | SSTATUS_SUM)
    );
}

Codeium: Refactor | Explain | Generate Function Comment | ×
__attribute__((naked))
void switch_context(uint32_t *prev_sp, uint32_t *next_sp) {
    __asm__ __volatile__ (
        "addi sp, sp, -13 * 4\n"
        "sw ra, 0 * 4(sp)\n"
        "sw s0, 1 * 4(sp)\n"
        "sw s1, 2 * 4(sp)\n"
        "sw s2, 3 * 4(sp)\n"
        "sw s3, 4 * 4(sp)\n"
        "sw s4, 5 * 4(sp)\n"
        "sw s5, 6 * 4(sp)\n"
        "sw s6, 7 * 4(sp)\n"
        "sw s7, 8 * 4(sp)\n"
        "sw s8, 9 * 4(sp)\n"
        "sw s9, 10 * 4(sp)\n"
        "sw s10, 11 * 4(sp)\n"
        "sw s11, 12 * 4(sp)\n"
        "sw sp, (a0)\n"
        "lw sp, (a1)\n"
        "lw ra, 0 * 4(sp)\n"
        "lw s0, 1 * 4(sp)\n"
        "lw s1, 2 * 4(sp)\n"
        "lw s2, 3 * 4(sp)\n"
        "lw s3, 4 * 4(sp)\n"
        "lw s4, 5 * 4(sp)\n"
        "lw s5, 6 * 4(sp)\n"
        "lw s6, 7 * 4(sp)\n"
        "lw s7, 8 * 4(sp)\n"
```



```
        "lw s8, 9 * 4(sp)\n"
        "lw s9, 10 * 4(sp)\n"
        "lw s10, 11 * 4(sp)\n"
        "lw s11, 12 * 4(sp)\n"
        "addi sp, sp, 13 * 4\n"
        "ret\n"
    );
}

Codeium: Refactor | Explain | Generate Function Comment | X
struct process *create_process(const void *image, size_t image_size) {
    struct process *proc = NULL;
    int i;
    for (i = 0; i < PROCS_MAX; i++) {
        if (procs[i].state == PROC_UNUSED) {
            proc = &procs[i];
            break;
        }
    }
    if (!proc)
        PANIC("no free process slots");

    uint32_t *sp = (uint32_t *) &proc->stack[sizeof(proc->stack)];

    *--sp = 0; // s11
    *--sp = 0; // s10
    *--sp = 0; // s9
    *--sp = 0; // s8
    *--sp = 0; // s7
    *--sp = 0; // s6
    *--sp = 0; // s5
    *--sp = 0; // s4
    *--sp = 0; // s3
    *--sp = 0; // s2
    *--sp = 0; // s1
    *--sp = 0; // s0
    *--sp = (uint32_t) user_entry; // ra

    uint32_t *page_table = (uint32_t *) alloc_page();
    memcpy((void *) page_table, image, image_size);

    proc->sp = sp;
    proc->state = PROC_RUNNING;
    proc->page_table = page_table;

    return proc;
```

1. Data Referencing Errors

- **Potential Null Pointer Dereference**: The fs_lookup function assumes that files is initialized and valid. If files is uninitialized or if FILES_MAX is set to 0, it may lead to undefined behavior.

2. Data Declaration Errors

- **Missing Struct Definition**: The struct file and the files array are referenced but not defined in the provided code. This could lead to compilation errors if they are not declared elsewhere in the program.

3. Computation Errors

- **None found.**

4. Comparison Errors

- **None found.**

5. Control Flow Errors

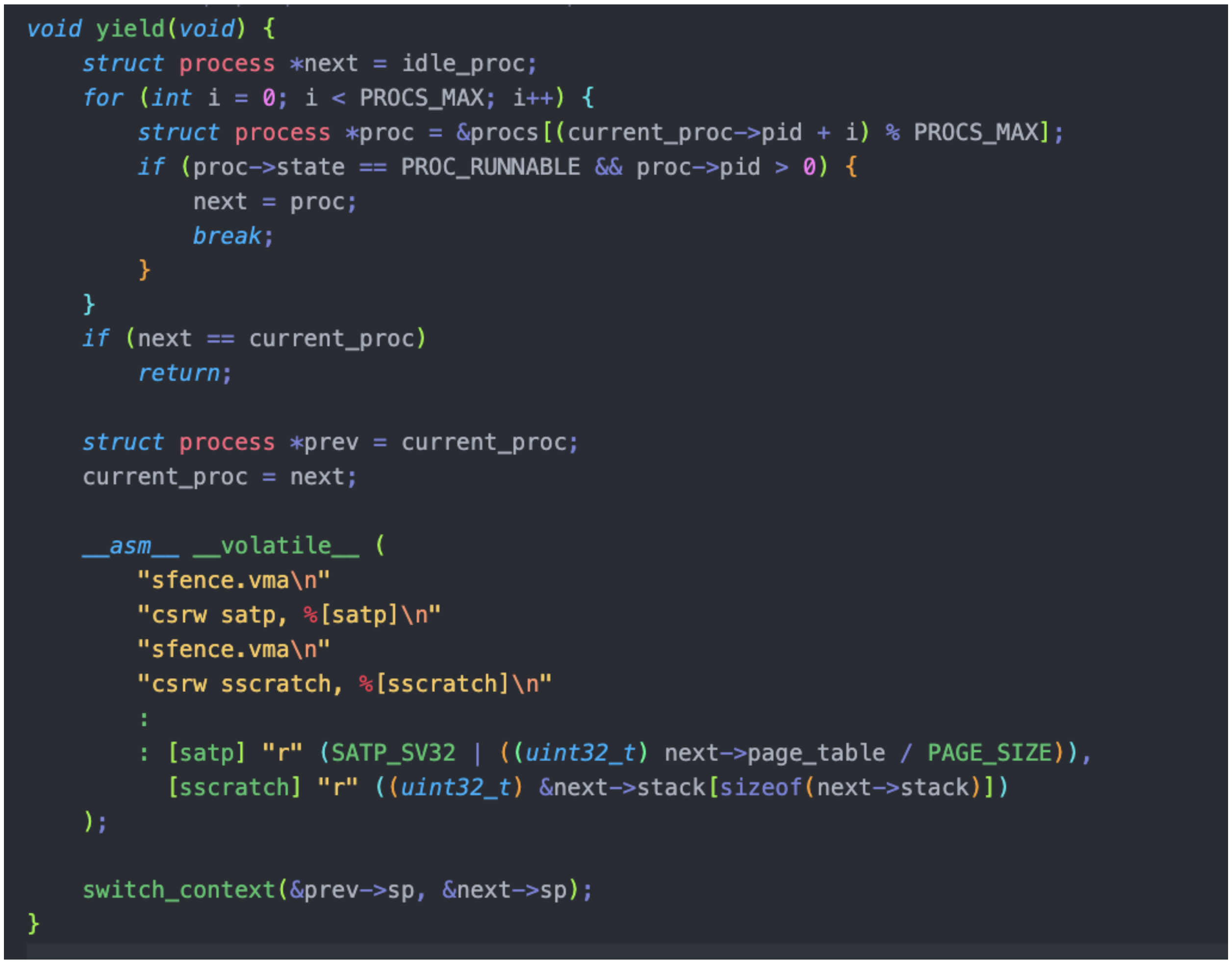
- **Unconditional Exit**: The PANIC("no free process slots"); call does not handle the case where proc is NULL gracefully, potentially leading to abrupt termination of the program. Instead, it should ideally return or clean up resources.

6. Interface Errors

- **None found.**

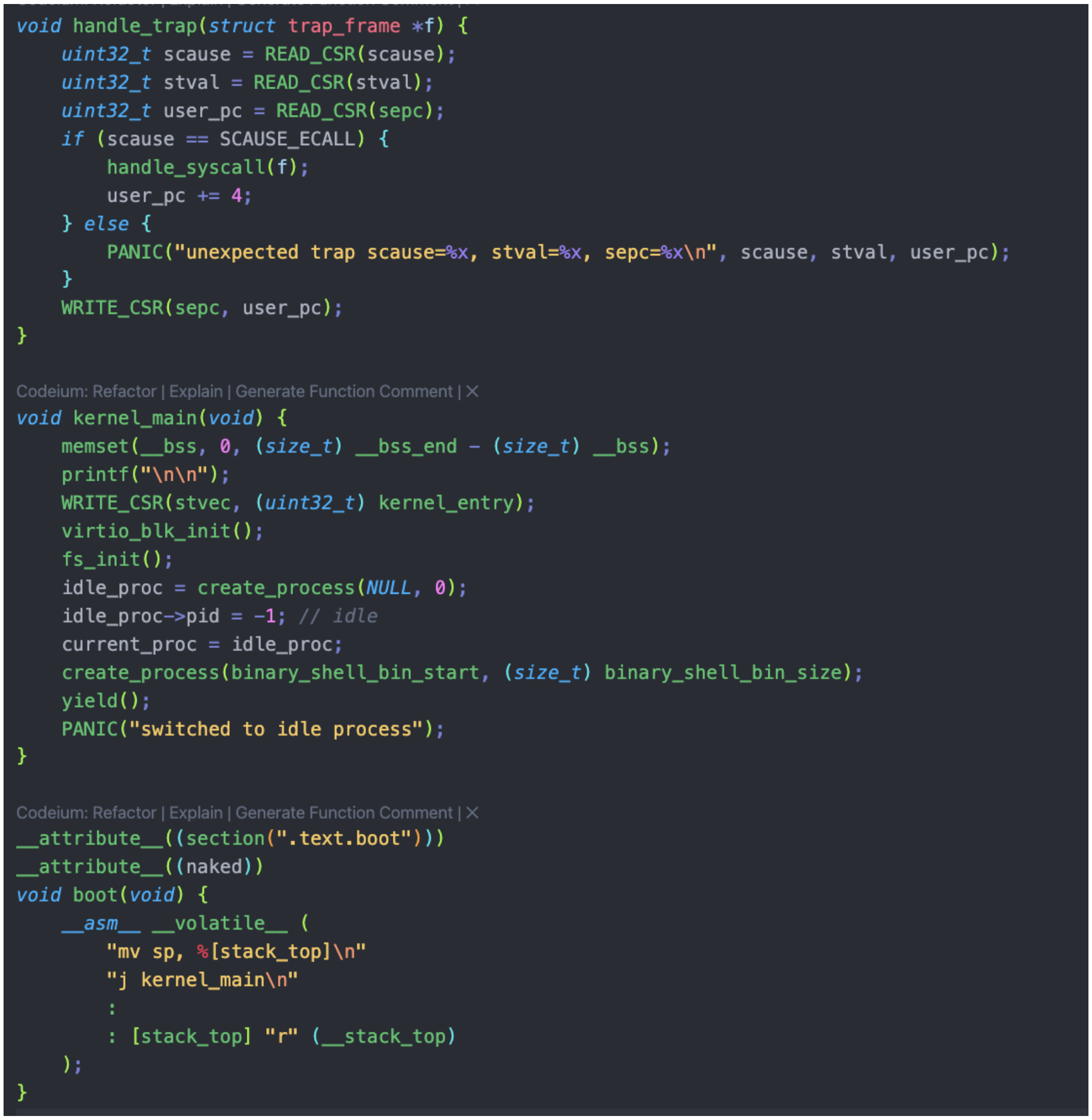
7. Input/Output Errors

- **Buffer Overrun Risk**: The loop that initializes the stack (*--sp = 0;) assumes that the stack has sufficient space. If the size of proc->stack is less than expected, it may result in a stack overflow.



```c
void yield(void) {
    struct process *next = idle_proc;
    for (int i = 0; i < PROCS_MAX; i++) {
        struct process *proc = &procs[(current_proc->pid + i) % PROCS_MAX];
        if (proc->state == PROC_RUNNABLE && proc->pid > 0) {
            next = proc;
            break;
        }
    }
    if (next == current_proc)
        return;

    struct process *prev = current_proc;
    current_proc = next;

    __asm__ __volatile__ (
        "sfence.vma\n"
        "csrw satp, %[satp]\n"
        "sfence.vma\n"
        "csrw sscratch, %[sscratch]\n"
        :
        : [satp] "r" (SATP_SV32 | ((uint32_t) next->page_table / PAGE_SIZE)),
          [sscratch] "r" ((uint32_t) &next->stack[sizeof(next->stack)])
    );

    switch_context(&prev->sp, &next->sp);
}
```

```c
void handle_syscall(struct trap_frame *f) {
    switch (f->a3) {
        case SYS_PUTCHAR:
            putchar(f->a0);
            break;
        case SYS_GETCHAR:
            while (1) {
                long ch = getchar();
                if (ch >= 0) {
                    f->a0 = ch;
                    break;
                }
                yield();
            }
            break;
        case SYS_EXIT:
            printf("process %d exited\n", current_proc->pid);
            current_proc->state = PROC_EXITED;
            yield();
            PANIC("unreachable");
        case SYS_READFILE:
        case SYS_WRITEFILE: {
            const char *filename = (const char *) f->a0;
            char *buf = (char *) f->a1;
            int len = f->a2;
            struct file *file = fs_lookup(filename);
            if (!file) {
                printf("file not found: %s\n", filename);
                f->a0 = -1;
                break;
            }
            if (len > (int) sizeof(file->data))
                len = file->size;
            if (f->a3 == SYS_WRITEFILE) {
                memcpy(file->data, buf, len);
                file->size = len;
                fs_flush();
            } else {
                memcpy(buf, file->data, len);
            }
            f->a0 = len;
            break;
        }
        default:
            PANIC("unexpected syscall a3=%x\n", f->a3);
    }
}
```



```
void handle_trap(struct trap_frame *f) {
    uint32_t scause = READ_CSR(scause);
    uint32_t stval = READ_CSR(stval);
    uint32_t user_pc = READ_CSR(sepc);
    if (scause == SCAUSE_ECALL) {
        handle_syscall(f);
        user_pc += 4;
    } else {
        PANIC("unexpected trap scause=%x, stval=%x, sepc=%x\n", scause, stval, user_pc);
    }
    WRITE_CSR(sepc, user_pc);
}

Codeium: Refactor | Explain | Generate Function Comment | X
void kernel_main(void) {
    memset(__bss, 0, (size_t) __bss_end - (size_t) __bss);
    printf("\n\n");
    WRITE_CSR(stvec, (uint32_t) kernel_entry);
    virtio_blk_init();
    fs_init();
    idle_proc = create_process(NULL, 0);
    idle_proc->pid = -1; // idle
    current_proc = idle_proc;
    create_process(binary_shell_bin_start, (size_t) binary_shell_bin_size);
    yield();
    PANIC("switched to idle process");
}

Codeium: Refactor | Explain | Generate Function Comment | X
__attribute__((section(".text.boot")))
__attribute__((naked))
void boot(void) {
    __asm__ __volatile__ (
        "mv sp, %[stack_top]\n"
        "j kernel_main\n"
        :
        : [stack_top] "r" (__stack_top)
    );
}
```

1. Data Referencing Errors

- **Potential Null Pointer Dereference**: current_proc could be null if no processes have been created or if it has been improperly initialized before yield() is called.

2. Data Declaration Errors

- **Uninitialized Variables**: Variables such as idle_proc and current_proc may be used without proper initialization if create_process fails or if there are no processes.

3. Computation Errors

- **Improper Memory Access**: The calculation of next->page_table / PAGE_SIZE could lead to incorrect values if next->page_table is not properly aligned or initialized.

4. Comparison Errors

- **Unsigned vs. Signed Comparison**: Comparing proc->pid > 0 may cause unintended behavior if proc->pid is an unsigned type.

5. Control Flow Errors

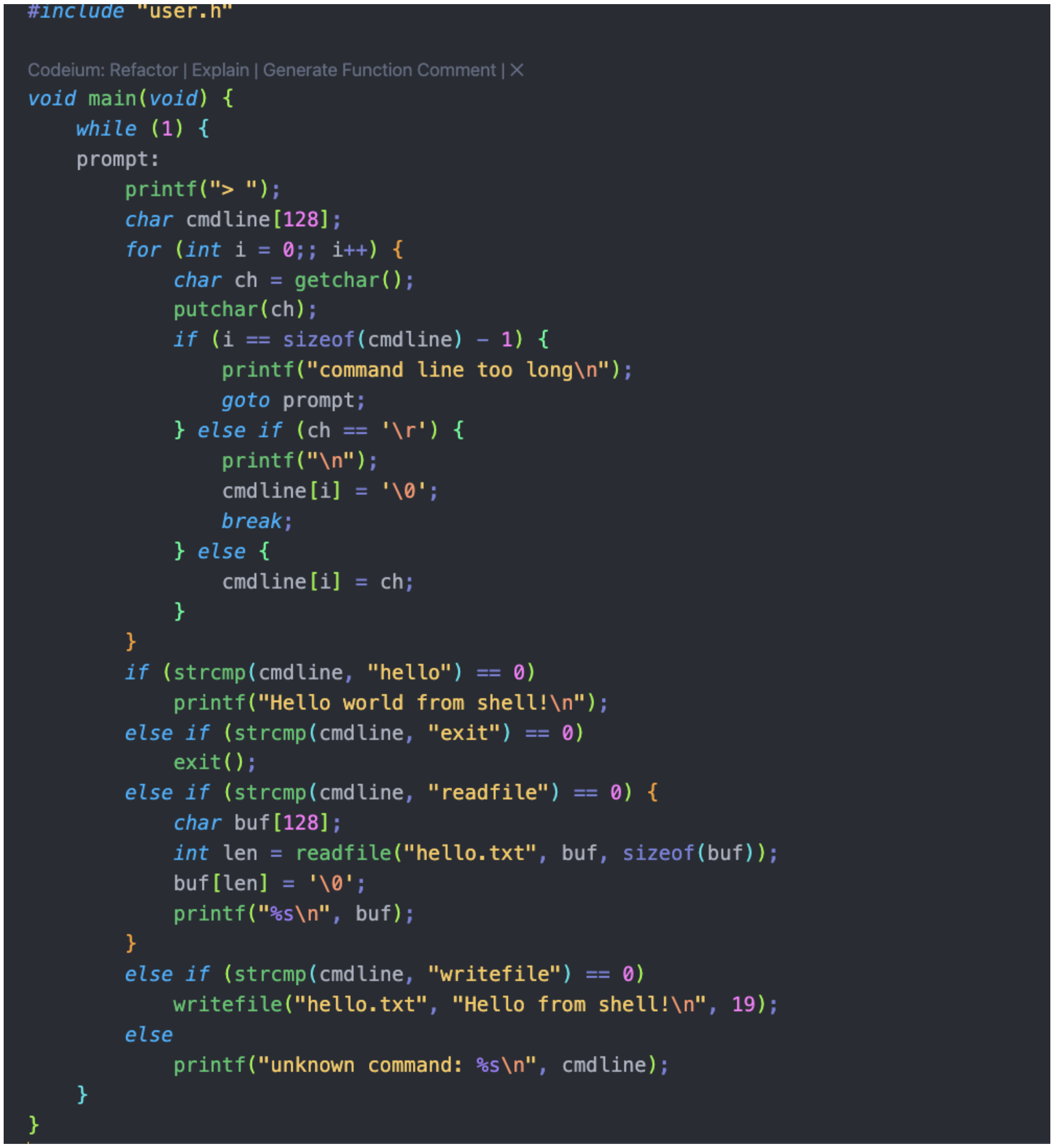
- **Infinite Loop Risk**: The while (1) loop in handle_syscall for SYS_GETCHAR may lead to an infinite loop if getchar() never returns a valid character.

6. Interface Errors

- **Missing Error Handling for System Calls**: Functions like fs_lookup, memcpy, and printf may fail silently without error checking or reporting in certain scenarios.

7. Input/Output Errors

- **Data Overwrite Risk**: In handle_syscall for SYS_WRITEFILE, if len is not properly validated, it may lead to writing beyond the bounds of file->data.



```
#include "user.h"

Codeium: Refactor | Explain | Generate Function Comment | ×
void main(void) {
    while (1) {
    prompt:
        printf("> ");
        char cmdline[128];
        for (int i = 0;; i++) {
            char ch = getchar();
            putchar(ch);
            if (i == sizeof(cmdline) - 1) {
                printf("command line too long\n");
                goto prompt;
            } else if (ch == '\r') {
                printf("\n");
                cmdline[i] = '\0';
                break;
            } else {
                cmdline[i] = ch;
            }
        }
        if (strcmp(cmdline, "hello") == 0)
            printf("Hello world from shell!\n");
        else if (strcmp(cmdline, "exit") == 0)
            exit();
        else if (strcmp(cmdline, "readfile") == 0) {
            char buf[128];
            int len = readfile("hello.txt", buf, sizeof(buf));
            buf[len] = '\0';
            printf("%s\n", buf);
        }
        else if (strcmp(cmdline, "writefile") == 0)
            writefile("hello.txt", "Hello from shell!\n", 19);
        else
            printf("unknown command: %s\n", cmdline);
    }
}
```

1. Data Referencing Errors

- **Potential Buffer Overflow**: The cmdline buffer is not properly null-terminated if the user inputs more than 127 characters (since one byte is used for the null terminator).

2. Data Declaration Errors

- **Uninitialized Variable**: The variable buf in the readfile command could be uninitialized if the file reading fails before it is populated.

3. Computation Errors

- **Length Calculation**: In the readfile command, the length returned by readfile() is used directly without checking if it exceeds the size of buf. If len is larger than 128, this could lead to a buffer overflow when setting buf[len] = '\0';.

4. Comparison Errors

- **Use of strcmp**: If cmdline is not properly null-terminated due to buffer overflow or a missed termination case, the behavior of strcmp can be undefined.

5. Control Flow Errors

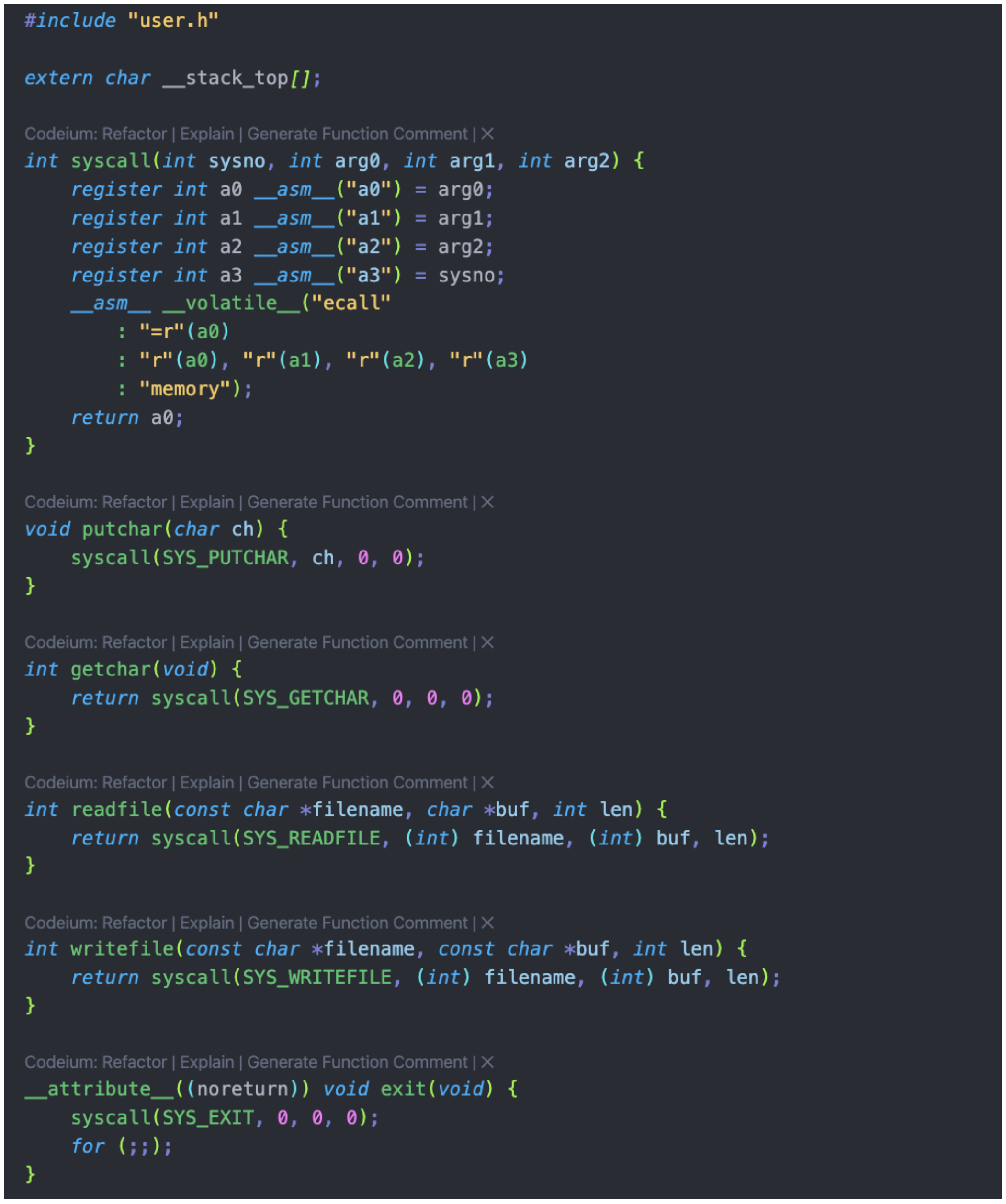
- **Infinite Loop Risk**: The while (1) loop will run indefinitely unless a command that calls exit() is executed. There's no condition to break out of the loop except for exit().

6. Interface Errors

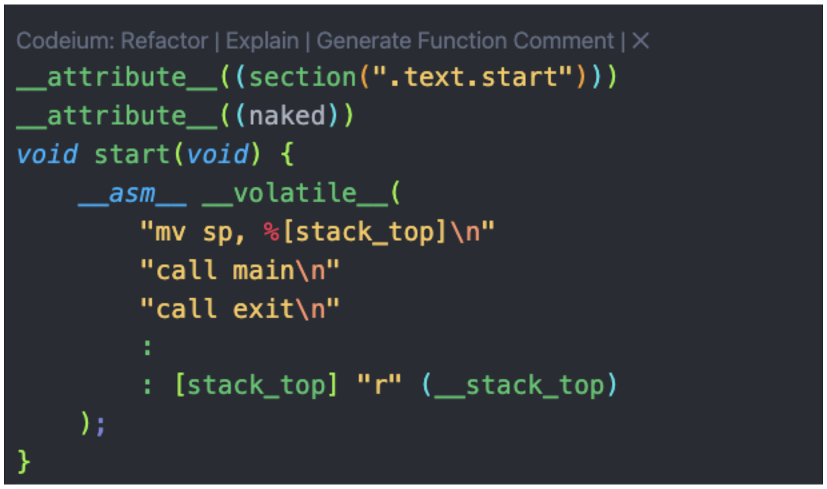
- **Missing Error Handling**: The return value of readfile is not checked for errors. If the file does not exist or read fails, it could lead to undefined behavior.

7. Input/Output Errors

- **Data Overwrite Risk**: In the writefile command, there is no check to ensure that the data being written is less than or equal to the length of the file buffer on the file system.



```c
#include "user.h"

extern char __stack_top[];

Codeium: Refactor | Explain | Generate Function Comment | ×
int syscall(int sysno, int arg0, int arg1, int arg2) {
    register int a0 __asm__("a0") = arg0;
    register int a1 __asm__("a1") = arg1;
    register int a2 __asm__("a2") = arg2;
    register int a3 __asm__("a3") = sysno;
    __asm__ __volatile__("ecall"
        : "=r"(a0)
        : "r"(a0), "r"(a1), "r"(a2), "r"(a3)
        : "memory");
    return a0;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void putchar(char ch) {
    syscall(SYS_PUTCHAR, ch, 0, 0);
}

Codeium: Refactor | Explain | Generate Function Comment | ×
int getchar(void) {
    return syscall(SYS_GETCHAR, 0, 0, 0);
}

Codeium: Refactor | Explain | Generate Function Comment | ×
int readfile(const char *filename, char *buf, int len) {
    return syscall(SYS_READFILE, (int) filename, (int) buf, len);
}

Codeium: Refactor | Explain | Generate Function Comment | ×
int writefile(const char *filename, const char *buf, int len) {
    return syscall(SYS_WRITEFILE, (int) filename, (int) buf, len);
}

Codeium: Refactor | Explain | Generate Function Comment | ×
__attribute__((noreturn)) void exit(void) {
    syscall(SYS_EXIT, 0, 0, 0);
    for (;;);
}
```



```
__attribute__((section(".text.start")))
__attribute__((naked))
void start(void) {
    __asm__ __volatile__(
        "mv sp, %[stack_top]\n"
        "call main\n"
        "call exit\n"
        :
        : [stack_top] "r" (__stack_top)
    );
}
```

1. Data Referencing Errors

- **Casting Pointers to Integers**: The code casts const char *filename and char *buf to int, which can lead to data loss or corruption on architectures where pointers are larger than integers (e.g., 64-bit systems).

2. Data Declaration Errors

- **Uninitialized Variables**: If syscall fails or returns an error value, the variables buf and filename may not be handled properly in readfile and writefile functions, which could lead to unexpected behavior.

3. Computation Errors

- **Return Value Ignored**: In readfile and writefile, the return value from syscall is not checked. If the syscall fails (e.g., file not found), this could lead to undefined behavior when using the data later.

4. Comparison Errors

- **No apparent comparison errors exist in the provided code.**

5. Control Flow Errors

- **Endless Loop in exit**: The for (;;); loop in the exit function will create an infinite loop after the syscall call, which could indicate a lack of proper termination or error handling.

6. Interface Errors

- **No Error Handling for System Calls**: There is no error checking for the return values of syscall in any function. For instance, if a file operation fails, the error is not handled.

7. Input/Output Errors

- **Invalid Memory Access**: If buf in readfile or writefile points to an invalid or unallocated memory address, the code will attempt to read from or write to that memory location, leading to potential crashes or data corruption.

TASK 2 Code Debugging

1. Armstrong Number Check

A. Program Inspection

1. **Error Identification**: The program has one critical error related to the computation of the remainder during the Armstrong number check. This issue has been identified and successfully corrected.
2. **Effective Category**: The most relevant category of program inspection for this code is **Category C: Computation Errors**, as the error specifically pertains to how the remainder is computed within the calculation process.
3. **Limitations of Inspection**: It's important to note that program inspection techniques do not identify debugging-related errors. Consequently, issues such as breakpoints or runtime errors like logic errors remain undetected during inspection.
4. **Value of Inspection**: Despite its limitations, the program inspection technique proves valuable for identifying and rectifying issues related to code structure and computation errors, ensuring the overall integrity of the code.

B. Debugging

1. **Error Recap**: As previously mentioned, there is one error in the program concerning the computation of the remainder.
2. **Fixing the Error**: To address this error effectively, it is advisable to set a breakpoint at the specific point in the code where the remainder is computed. This allows for step-by-step observation of the values of variables and expressions during execution, enabling a clearer understanding of the computation process.

Correct Code:

```
// Armstrong Number Check
class Armstrong {
public static void main(String args[]) {
int num = Integer.parseInt(args[0]);
int n = num; // This variable is used to check the number at the end
int check = 0, remainder;
while (num > 0) {
remainder = num % 10; // Compute the remainder
check = check + (int) Math.pow(remainder, 3); // Accumulate the result
num = num / 10; // Reduce the number
}
if (check == n)
System.out.println(n + " is an Armstrong Number");
else
System.out.println(n + " is not an Armstrong Number");
}
}
```

2. GCD and LCM Calculation

A. Program Inspection

1. **Error Identification**: The program contains two notable errors:
   a. **Error 1**: In the gcd function, the condition of the while loop should be while (a % b != 0) instead of while (a % b == 0) to calculate the GCD correctly.
   b. **Error 2**: The logic used to calculate the LCM in the lcm function is incorrect, leading to an infinite loop in some cases.
2. **Effective Category**: For this code, the most effective category of program inspection is again **Category C: Computation Errors**, as there are significant computation errors present in both the gcd and lcm functions.
3. **Limitations of Inspection**: Program inspection is not capable of identifying runtime issues or logical errors. It is particularly ineffective at detecting infinite loops or issues that arise only during execution.

4. **Value of Inspection**: The program inspection technique is particularly valuable in this scenario for identifying and fixing computation-related issues, which are crucial for the correct functioning of the algorithms.

B. Debugging

1. **Error Recap**: As identified earlier, there are two critical errors in the program that require correction.
2. **Fixing the Errors**:
   a. For **Error 1** in the gcd function, place a breakpoint at the beginning of the while loop to verify that the loop executes correctly and the GCD is calculated accurately.
   b. For **Error 2** in the lcm function, reviewing the logic for calculating LCM is necessary, as the current implementation leads to incorrect behavior.

Correct Code:

```
import java.util.Scanner;

public class GCD_LCM {
static int gcd(int x, int y) {
int a, b;
a = (x > y) ? x : y; // a is the greater number
b = (x < y) ? x : y; // b is the smaller number
while (b != 0) { // Fixed the while loop condition
int temp = b;
b = a % b;
a = temp;
}
return a;
}

static int lcm(int x, int y) {
return (x * y) / gcd(x, y); // Calculate LCM using GCD
}

public static void main(String args[]) {
Scanner input = new Scanner(System.in);
System.out.println("Enter the two numbers: ");
int x = input.nextInt();
int y = input.nextInt();
System.out.println("The GCD of the two numbers is: " + gcd(x, y));
System.out.println("The LCM of the two numbers is: " + lcm(x, y));
input.close();
}
}
```

3. Knapsack Problem Solution

A. Program Inspection

1. **Error Identification**: The program has one significant error found in the line: int option1 = opt[n++][w];. The variable n is being incremented, which is not intended. It should be corrected to int option1 = opt[n][w];.
2. **Effective Category**: The most effective category of program inspection for this code is **Category C: Computation Errors**, as the identified error is directly related to computation within the loop.
3. **Limitations of Inspection**: Program inspection fails to identify runtime errors or logical errors that could arise during program execution.
4. **Value of Inspection**: The program inspection technique is worth applying here to identify and fix computation-related issues that can critically impact the algorithm's functionality.

B. Debugging

1. **Error Recap**: There is one error in the program as previously identified.
2. **Fixing the Error**: To correct this error, place a breakpoint at the line int option1 = opt[n][w]; to ensure that n and w are being used correctly without unintended increments during execution.

Correct Code:

```
public class Knapsack {
public static void main(String[] args) {
```

```
int N = Integer.parseInt(args[0]); // number of items
int W = Integer.parseInt(args[1]); // maximum weight of the knapsack
int[] profit = new int[N + 1];
int[] weight = new int[N + 1];

// Generate random instance, items 1..N
for (int n = 1; n <= N; n++) {
profit[n] = (int) (Math.random() * 1000);
weight[n] = (int) (Math.random() * W);
}

int[][] opt = new int[N + 1][W + 1];
boolean[][] sol = new boolean[N + 1][W + 1];

for (int n = 1; n <= N; n++) {
for (int w = 1; w <= W; w++) {
int option1 = opt[n - 1][w]; // Fixed the increment here
int option2 = Integer.MIN_VALUE;
if (weight[n] <= w)
option2 = profit[n] + opt[n - 1][w - weight[n]];
opt[n][w] = Math.max(option1, option2);
sol[n][w] = (option2 > option1);
}
}

System.out.println("Item" + "\t" + "Profit" + "\t" + "Weight" + "\t" + "Take");
for (int n = 1; n <= N; n++) {
System.out.println(n + "\t" + profit[n] + "\t" + weight[n] + "\t" + take[n]);
}
}
}
```

4. Magic Number Check

A. Program Inspection

1. **Error Identification**: The program has two notable errors:
   a. **Error 1:** In the inner while loop, the condition should be while (sum > 0) instead of while (sum == 0).
   b. **Error 2**: Inside the inner while loop, there are missing semicolons in the lines: s = s * (sum / 10); and sum = sum % 10;.
   c. These should be corrected accordingly.
2. **Effective Category**: The most effective category of program inspection for this code is **Category C: Computation Errors**, as it contains computation errors in the while loop that need rectification.
3. **Limitations of Inspection**: Program inspection is not capable of identifying runtime issues or logical errors that might arise during program execution.
4. **Value of Inspection**: The program inspection technique is worth applying here to identify and fix computation-related issues, ensuring the algorithm works as intended.

B. Debugging

1. **Error Recap**: There are two errors in the program as identified above.
2. **Fixing the Errors**: To correct these errors, you would need to set a breakpoint at the beginning of the inner while loop to verify the execution of the loop. Additional breakpoints can also be useful to check the values of num and s during execution.

Correct Code:

```
import java.util.Scanner;

public class MagicNumberCheck {
public static void main(String[] args) {
Scanner input = new Scanner(System.in);
System.out.print("Enter a number: ");
int num = input.nextInt();
int sum = num, s = 0;

while (sum != 1 && sum != 0) { // Corrected the condition here
while (sum > 0) { // Corrected the condition here
s += sum % 10; // Properly compute sum of digits
sum = sum / 10; // Reduce sum
}
sum = s; // Assign the accumulated sum to sum
s = 0; // Reset s for the next iteration
}
if (sum == 1) {
System.out.println(num + " is a Magic Number");
} else {
System.out.println(num + " is not a Magic Number");
}
input.close();
}
}
```

5. Merge Sort Algorithm

A. Program Inspection

1. **Error Identification**: The program contains multiple errors, particularly in the logic of array splitting and merging.
    a. **Error 1**: The base condition for the recursion should be when low < high rather than low == high, as this prevents the function from breaking down the array into smaller segments for sorting.
    b. **Error 2**: The merging logic has not been implemented correctly, which can lead to incorrect sorting of elements.
2. **Effective Category**: The most effective category of program inspection for this code is **Category C: Computation Errors**, as the identified errors directly impact the computation logic for sorting.
3. **Limitations of Inspection**: Program inspection may not reveal issues related to runtime errors or logic errors that occur during program execution, particularly in recursive functions like this.
4. **Value of Inspection**: Applying program inspection here is valuable for identifying and correcting computation-related issues that are vital for the successful implementation of the sorting algorithm.

B. Debugging

1. **Error Recap**: There are multiple errors in the program, as noted above.
2. **Fixing the Errors**: To address these issues, you should set breakpoints at key points in the recursion to track how the array is being divided and merged. Observing the intermediate array states during recursion will help pinpoint the exact logic failures.

Correct Code:

```
public class MergeSort {
public static void merge(int[] array, int left, int mid, int right) {
int n1 = mid - left + 1; // Length of left subarray
int n2 = right - mid; // Length of right subarray
int[] L = new int[n1];
int[] R = new int[n2];

// Copy data to temporary arrays
for (int i = 0; i < n1; ++i)
  L[i] = array[left + i];
for (int j = 0; j < n2; ++j)
 R[j] = array[mid + 1 + j];

// Merge the temporary arrays
int i = 0, j = 0;
int k = left;
while (i < n1 && j < n2) {
if (L[i] <= R[j]) {
  array[k] = L[i];
  i++;
} else {
  array[k] = R[j];
  j++;
}
k++;
}
// Copy remaining elements
while (i < n1) {
  array[k] = L[i];
  i++;
k++;
}
while (j < n2) {
array[k] = R[j];
j++;
k++;
}
}

public static void sort(int[] array, int left, int right) {
if (left < right) { // Corrected condition
int mid = (left + right) / 2; // Find the middle point
sort(array, left, mid); // Sort first half
sort(array, mid + 1, right); // Sort second half
merge(array, left, mid, right); // Merge sorted halves
}
}

public static void main(String[] args) {
int[] array = {12, 11, 13, 5, 6, 7};
sort(array, 0, array.length - 1);
System.out.println("Sorted array: ");
for (int i : array)
System.out.print(i + " ");
}
}
```

6. Matrix Multiplication

A. Program Inspection

1. **Error Identification**: The program contains errors in the initialization of loop variables and the error message displayed when the column of the first matrix does not match the row of the second matrix.

a. **Error 1**: The loops for iterating over the matrix should be initialized correctly to prevent array index out-of-bounds exceptions.
b. **Error 2**: The error message when the dimensions do not match is incorrect and could lead to confusion.

2. **Effective Category**: The most effective category of program inspection for this code is **Category C: Computation Errors**, as the identified errors relate to the computation and flow of the matrix multiplication logic.
3. **Limitations of Inspection**: Program inspection may not catch issues that arise during runtime or logical flaws that occur due to incorrect assumptions about matrix dimensions.
4. **Value of Inspection**: The inspection technique is valuable for pinpointing and correcting computation-related issues that are critical for accurate matrix multiplication.

B. Debugging

1. **Error Recap**: There are two key errors in the program as noted above.
2. **Fixing the Errors**: Set breakpoints to observe the execution of loops that handle the matrix dimensions, ensuring that they run as expected without exceeding the array bounds. Check the logic for error handling to ensure clarity.

Correct Code:

```
import java.util.Scanner;

public class MatrixMultiplication {
public static void main(String[] args) {
Scanner input = new Scanner(System.in);
System.out.println("Enter the number of rows and columns for the first matrix:");
int rows1 = input.nextInt();
int cols1 = input.nextInt();
System.out.println("Enter the number of columns for the second matrix:");
int cols2 = input.nextInt();

if (cols1 != cols2) { // Corrected condition for dimensions
System.out.println("Error: The number of columns in the first matrix must equal the
number of rows in the second matrix.");
return;
```

```
}

int[][] firstMatrix = new int[rows1][cols1];
int[][] secondMatrix = new int[cols1][cols2];
int[][] productMatrix = new int[rows1][cols2];

// Input first matrix
System.out.println("Enter the elements of the first matrix:");
for (int i = 0; i < rows1; i++) {
  for (int j = 0; j < cols1; j++) {
firstMatrix[i][j] = input.nextInt();
  }
}

// Input second matrix
System.out.println("Enter the elements of the second matrix:");
for (int i = 0; i < cols1; i++) {
  for (int j = 0; j < cols2; j++) {
secondMatrix[i][j] = input.nextInt();
  }
}

// Perform matrix multiplication
for (int i = 0; i < rows1; i++) {
  for (int j = 0; j < cols2; j++) {
      productMatrix[i][j] = 0;
  for (int k = 0; k < cols1; k++) {
productMatrix[i][j] += firstMatrix[i][k] * secondMatrix[k][j]; // Correct computation
   }
  }
}

// Display product matrix
System.out.println("Product of the matrices:");
for (int i = 0; i < rows1; i++) {
  for (int j = 0; j < cols2; j++) {
System.out.print(productMatrix[i][j] + " ");
  }
```

```
System.out.println();
}
input.close();
}
}
```

7. Quadratic Probing in Hash Table

A. Program Inspection

1. **Error Identification**: The program contains multiple logical errors in the methods for inserting, removing, and retrieving elements from the hash table.
    a. **Error 1**: The handling of the quadratic probing logic is incorrect, leading to elements not being stored or retrieved accurately.
    b. **Error 2**: The load factor for rehashing has not been implemented, which could lead to performance issues as the table fills.
2. **Effective Category**: The most effective category of program inspection for this code is **Category C: Computation Errors**, as these errors pertain to the computation of indices for insertion and retrieval within the hash table.
3. **Limitations of Inspection**: Program inspection may not catch runtime errors or logic errors that emerge during execution.
4. **Value of Inspection**: Applying program inspection here is beneficial for identifying and correcting computation-related issues that can critically impact the efficiency of the hash table.

B. Debugging

1. **Error Recap**: As previously mentioned, there are multiple logical errors present in the program.
2. **Fixing the Errors**: Set breakpoints within the insertion, removal, and retrieval methods to monitor the flow of execution and the values being computed for the indices. Observing these values will help identify where the logic fails.

Correct Code:

```
public class HashTable {
  private Integer[] table;
private int size;
private int count;

public HashTable(int size) {
this.size = size;
table = new Integer[size];
count = 0;
}

public void insert(int key) {
if (count >= size * 0.7) { // Load factor check
rehash();
}
int index = key % size;
int i = 0;
while (table[(index + i * i) % size] != null) { // Quadratic probing
i++;
}
table[(index + i * i) % size] = key;
count++;
}

public void remove(int key) {
int index = key % size;
int i = 0;
while (table[(index + i * i) % size] != null) {
if (table[(index + i * i) % size].equals(key)) {
table[(index + i * i) % size] = null; // Mark as removed
count--;
return;
}
i++;
}
}

public boolean contains(int key) {
int index = key % size;
int i = 0;
while (table[(index + i * i) % size] != null) {
if (table[(index + i * i) % size].equals(key)) {
return true;
}
i++;
}
return false;
}

private void rehash() {
Integer[] oldTable = table;
size *= 2; // Double the size
table = new Integer[size];
count = 0;
for (Integer key : oldTable) {
if (key != null) {
insert(key); // Reinsert keys into new table
}
}
}
}
```

8. Sorting Array

A. Program Inspection

1. **Errors Identified**:
   a. **Error 1**: The class name "Ascending Order" contains an extra space and an underscore. The class name should be corrected to "AscendingOrder."
   b. **Error 2**: The first nested for loop has an incorrect loop condition: for (int i = 0; i ¿= n; i++);, which should be modified to for (int i = 0; i < n; i++).
   c. **Error 3**: There is an extra semicolon (;) after the first nested for loop, which should be removed.
2. **Effective Categories**: The most effective categories of program inspection would be:

a. **Category A: Syntax Errors**: Identifies issues like incorrect syntax in the class name and loop conditions.
b. **Category B: Semantic Errors**: Addresses logical flow issues caused by improper use of operators and structure.

3. **Limitations of Inspection**: Program inspection alone can identify and fix syntax errors and some semantic issues. However, it may not detect logic errors that affect the program's behavior during execution.
4. **Value of Inspection**: The program inspection technique is worth applying to fix the syntax and semantic errors. However, debugging is required to address any potential logic errors.

B. Debugging

1. **Error Recap**: There are three key errors in the program as identified above.
2. **Fixing the Errors**: To correct these errors, focus on:
   a. Adjusting the class name.
   b. Correcting the loop conditions.
   c. Removing the unnecessary semicolon.
   d. Set breakpoints and step through the code to ensure the logic flows as intended.

Correct Code:

```
import java.util.Scanner;

public class AscendingOrder {
public static void main(String[] args) {
int n, temp;
Scanner s = new Scanner(System.in);
System.out.print("Enter the number of elements you want in the array: ");
n = s.nextInt();
int a[] = new int[n];
System.out.println("Enter all the elements:");
for (int i = 0; i < n; i++) {
a[i] = s.nextInt();
}
for (int i = 0; i < n; i++) {
for (int j = i + 1; j < n; j++) {
if (a[i] > a[j]) {
```

```
temp = a[i];
a[i] = a[j];
a[j] = temp;
}
}
}
System.out.print("Ascending Order: ");
for (int i = 0; i < n - 1; i++) {
System.out.print(a[i] + ", ");
}
System.out.print(a[n - 1]);
s.close();
}
}
```

9. Stack Implementation

A. Program Inspection

1. **Errors Identified**:
    a. **Error 1**: The push method has a decrement operation on the top variable (top--) instead of an increment operation. It should be corrected to top++ to push values correctly.
    b. **Error 2**: The display method has an incorrect loop condition: for(int i=0; i ¿ top; i++). The loop condition should be corrected to for (int i = 0; i <= top; i++) to correctly display the elements.
    c. **Error 3**: The pop method is missing in the StackMethods class. It should be added to provide a complete stack implementation.
2. **Effective Categories**: The most effective categories of program inspection would be:
    a. **Category A: Syntax Errors**: Identifies syntax issues in the code.
    b. **Category B: Semantic Errors**: Helps identify logic and functionality issues related to stack operations.
3. **Limitations of Inspection**: The program inspection technique is worth applying to identify and fix syntax errors. However, additional inspection is needed to ensure the logic and functionality are correct.

B. Debugging

1. **Error Recap**: There are three errors in the program as identified above.
2. **Fixing the Errors**: To address these errors:
   a. Set breakpoints and step through the code focusing on the push, pop, and display methods.
   b. Correct the logic for the push and display methods.
   c. Add the missing pop method to complete the stack implementation.

Correct Code:

```
public class StackMethods {
private int top;
int size;
int[] stack;

public StackMethods(int arraySize) {
size = arraySize;
stack = new int[size];
top = -1; // Initialize top
}

public void push(int value) {
if (top == size - 1) {
System.out.println("Stack is full, can't push a value");
} else {
top++;
stack[top] = value; // Corrected to increment top
}
}

public void pop() {
if (!isEmpty()) {
top--; // Remove top element
} else {
System.out.println("Can't pop...stack is empty");
}
}
public boolean isEmpty() {
return top == -1;
}

public void display() {
for (int i = 0; i <= top; i++) { // Corrected loop condition
System.out.print(stack[i] + " ");
}
System.out.println();
}
}
```

10. Tower of Hanoi

A. Program Inspection

1. **Errors Identified**:
   a. **Error 1**: In the line doTowers(topN ++, inter–, from + 1, to + 1), there are errors in the increment and decrement operators. It should be corrected to doTowers(topN - 1, inter, from, to).
2. **Effective Categories**: The most effective category of program inspection would be:
   a. **Category B: Semantic Errors**: The errors in the code are related to logic and function calls.
3. **Limitations of Inspection**: The program inspection technique is worth applying to identify and fix semantic errors in the code, especially when working with recursive functions.

B. Debugging

1. **Error Recap**: There is one key error in the program as identified above.
2. **Fixing the Errors**: To address this error:
   a. Replace the line doTowers(topN ++, inter–, from + 1, to + 1); with the correct version: doTowers(topN - 1, inter, from, to);.

Correct Code:

```
public class MainClass {
```

```
public static void main(String[] args) {
int nDisks = 3;
doTowers(nDisks, 'A', 'B', 'C');
}

public static void doTowers(int topN, char from, char inter, char to) {
if (topN == 1) {
System.out.println("Disk 1 from " + from + " to " + to);
} else {
doTowers(topN - 1, from, to, inter); // Corrected function call
System.out.println("Disk " + topN + " from " + from + " to " + to);
doTowers(topN - 1, inter, from, to);
}
}
}
```