
CISA
TOPNARY - T81TERNARY.CWEB
CISA
CISA
TERNARY_COMMON.H

Program Profiling Summary 3
File: ternary_system.cweb 3
General Stats 3
Key Components Identified 3
1. Axion Package Management & Optimization: 3
2. Ternary Arithmetic (T81BigInt & TritJS) 3
3. Memory and Optimization-Related Functions 3
4. I/O & Logging 4
5. Frequent Function Calls (≥3 times) 4

Program Profiling Summary

File: ternary_system.cweb

General Stats

- **Total Functions:** 78
- **Total Global Variables:** 205
- **Total Function Calls:** 628
- **Main Entry Points:**
    - **`main()`** (User-space entry point)
    - No explicit `init_module()` or `exit_module()`, suggesting it may not be a kernel module or these functions are indirectly referenced.

Key Components Identified

---

1. Axion Package Management & Optimization:

- `axion_install_pkg()`, `axion_update_pkg()`, `axion_rollback_pkg()`
- `axion_predict_needs()`, `axion_resolve_deps()`
- `axion_predictive_load_balancer()`
- `axion_get_perf_feedback()`, `axion_suggestion_work()`

---

2. Ternary Arithmetic (T81BigInt & TritJS)

- `tritjs_add_big()`, `tritjs_subtract_big()`, `tritjs_multiply_big()`
- `tritjs_divide_big()`, `tritjs_sqrt_complex()`, `tritjs_log3_complex()`
- `tritjs_factorial_big()`, `tritjs_power_big()`
- `binary_to_trit()`, `trit_to_binary()`

---

3. Memory and Optimization-Related Functions

- `allocate_digits()`, `karatsuba()` (Efficient multiplication)
- `multiply_with_cache()`, `mul_cache_lookup()`, `mul_cache_store()`
- **Frequent function calls:** `memset()`, `memcpy()`, `calloc()`, `free()`

4. I/O & Logging

- `printk()` (Kernel logging)
- `log_error()`
- `snprintf()`, `printw()`, `strncpy()`

5. Frequent Function Calls (≥3 times)

- **Common C Functions:** `memset()`, `memcpy()`, `calloc()`, `free()`, `printk()`, `strncpy()`, `snprintf()`
- **Performance-related:** `karatsuba()`, `allocate_digits()`, `multiply_with_cache()`
- **Axion functions:** `axion_install_pkg()`, `axion_predict_needs()`
- **Ternary Conversions:** `binary_to_trit()`, `trit_to_binary()`, `parse_trit_string()`

@* Ternary System: Unified Kernel Module and User–Space Utility

This literate program integrates our Axion kernel module with the TritJS-CISA–Optimized user–space utility into one source file. Our mission is to transition from binary to native ternary computing by providing a unified foundation that spans both the kernel and user spaces.

FEATURES:

- Common Ternary Logic Definitions:
  - Unified definitions for ternary states, instruction opcodes, and error codes.
- Axion Kernel Module:
  - AI–powered predictive load balancing.
  - Ternary binary execution via JIT compilation (emulated on binary hardware).
  - Integrated package management with dependency resolution and rollback.
- TritJS-CISA–Optimized Utility:
  - Advanced ternary arithmetic using optimized (Karatsuba) algorithms with caching.
  - Scientific functions (sqrt, log3, sin, cos, tan, π) via double conversion.
  - Secure audit logging and real–time intrusion detection.
  - An interactive, ncurses–based command–line interface.
- Built–in Help System:
  - Type "help" at the command prompt to display a summary of available commands, usage examples, and detailed feature descriptions.
- Unified Mission:
  - To provide a seamless, cohesive environment for testing and advancing ternary computing from the kernel all the way to the application level.

USAGE:

- For kernel deployment: Compile with __KERNEL__ defined so that the Axion module is built.

• For user–space operation: Compile normally to build the TritJS-CISA–Optimized utility.

• In the utility, type "help" at the prompt for a listing of commands and features.

This system is designed to evolve along with our research and development efforts,

ensuring that both components work seamlessly toward our goal of native ternary computing.

@*

/*
==========

Section 1: Shared Header - ternary_common.h

This header provides common definitions for ternary logic: ternary states,

instruction opcodes for kernel execution, and error codes used by both parts.

========== */

@<ternary_common.h@>=

```
/* ternary_common.h */

#ifndef TERNARY_COMMON_H

#define TERNARY_COMMON_H


/* Common definitions for ternary logic */


/* Ternary states */

#define TERNARY_NEGATIVE -1  /* Represents -1: negative/false */

#define TERNARY_ZERO      0  /* Represents 0: neutral/unknown */

#define TERNARY_POSITIVE  1  /* Represents 1: positive/true */


/* Ternary instruction set opcodes for kernel execution */

#define TADD   0x01  /* Add two trits (clamped between -1 and 1) */

#define TSUB   0x02  /* Subtract two trits */
```

```
101.
102. #define TMUL   0x03  /* Multiply two trits */
103.
104. #define TAND   0x04  /* Ternary AND (minimum of two trits) */
105.
106. #define TOR    0x05  /* Ternary OR (maximum of two trits) */
107.
108. #define TNOT   0x06  /* Ternary NOT (negate a trit) */
109.
110. #define TJMP   0x07  /* Jump if condition is non-zero */
111.
112. #define TJZ    0x08  /* Jump if condition is zero */
113.
114. #define TJNZ   0x09  /* Jump if condition is non-zero */
115.
116. #define TLOAD  0x0A  /* Load a trit from memory into a register */
117.
118. #define TSTORE 0x0B  /* Store a trit from a register into memory */
119.
120. #define THLT   0x0C  /* Halt execution */
121.
122.
123. /* Common error codes for ternary operations */
124.
125. typedef enum {
126.
127.     TERNARY_NO_ERROR = 0,
128.
129.     TERNARY_ERR_MEMALLOC,
130.
131.     TERNARY_ERR_INVALID_INPUT,
132.
133.     TERNARY_ERR_DIVZERO,
134.
135.     TERNARY_ERR_OVERFLOW,
136.
137.     TERNARY_ERR_UNDEFINED,
138.
139.     TERNARY_ERR_NEGATIVE,
140.
141.     TERNARY_ERR_PRECISION,
142.
143.     TERNARY_ERR_MMAP,
144.
145.     TERNARY_ERR_SCRIPT
146.
147. } TernaryError;
148.
149.
150. #endif /* TERNARY_COMMON_H */
151.
152.
```

```
@*
==========================================================================

  Section 2: Kernel Module (Axion)

  This section implements the Axion kernel module. It provides:

    • AI–driven load balancing

    • Emulated ternary binary execution with a custom instruction set

    • Package management with dependency resolution and rollback

  This module is compiled only when __KERNEL__ is defined.


========================================================================== */

#ifdef __KERNEL__

@<Axion Kernel Module@>=

/*

 * Axion Kernel Module: AI–Powered Ternary Evolution

 *

 * Objective:

 *   - Dynamically manage system resources with an AI–driven load balancer.

 *   - Execute ternary binaries (emulated on binary hardware) using a dedicated

 *     ternary instruction set.

 *   - Provide package management with dependency resolution and rollback.

 *

 * This module uses common definitions from ternary_common.h and exposes IOCTL

 * interfaces and debugfs entries for telemetry and control.

 */


#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/init.h>
```

```
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/uaccess.h>
#include <linux/ioctl.h>
#include <linux/device.h>
#include <linux/kthread.h>
#include <linux/delay.h>
#include <linux/sched.h>
#include <linux/debugfs.h>
#include <linux/workqueue.h>
#include <linux/cpu.h>
#include <linux/notifier.h>
#include <linux/topology.h>
#include <linux/mm.h>
#include <linux/slab.h>
#include <linux/mempolicy.h>
#include <linux/numa.h>
#include <linux/random.h>
#include <linux/binfmts.h>
#include <linux/jiffies.h>
#include <linux/timer.h>
#include <asm/io.h>
#include "ternary_common.h"


/* Module–specific constants and macros */
#define DEVICE_NAME "axion_opt"                    /* Device name for character device */
#define AXION_DEFAULT_REGISTER 0x1F                /* Default register value */

```

```
#define AXION_DEBUGFS_DIR "axion_debug"            /* Debugfs directory name */

#define AXION_DEBUGFS_FILE "cpu_state"             /* Debugfs file for telemetry */

#define WORKLOAD_HISTORY_SIZE 50                   /* History buffer size */

#define ANOMALY_THRESHOLD 30                       /* Anomaly detection threshold (%)
*/

#define SELF_HEALING_THRESHOLD 50                  /* Self–healing threshold (%) */

#define PREDICTIVE_LOAD_BALANCING_INTERVAL 5000    /* Load balancing
interval (ms) */

#define RESOURCE_WEIGHT_CPU 0.5                    /* Initial CPU weight */

#define RESOURCE_WEIGHT_GPU 0.3                    /* Initial GPU weight */

#define RESOURCE_WEIGHT_RAM 0.2                    /* Initial RAM weight */

#define FEEDBACK_ADJUSTMENT_FACTOR 0.1             /* AI adjustment factor */

#define MAX_PACKAGES 32                          /* Maximum packages tracked */

#define MAX_DEPS 8                             /* Maximum dependencies per package */

#define TBIN_MAGIC 0x5442494E                    /* Magic number for TBIN files */

#define TERNARY_MEM_SIZE 32                      /* Size of ternary memory array */


/* Data structures */

struct resource_state {

    int cpu_usage;

    int ram_usage;

    int gpu_usage;

    int action;

};


struct rl_model {

    int q_table[3][3];

    int last_state;

    int last_action;

```

```
};

struct tbin_header {
    uint32_t magic;
    uint32_t entry_point;
    uint32_t code_size;
    uint32_t data_size;
};

struct tbin_state {
    int8_t reg[3];        /* Ternary registers */
    int8_t memory[TERNARY_MEM_SIZE];
    uint32_t ip;          /* Instruction pointer */
    void *code;           /* Code buffer */
    uint32_t code_size;
    int running;          /* Running flag */
};

struct package {
    char name[32];
    char version[16];
    char deps[MAX_DEPS][32];
    int dep_count;
    int state;            /* Ternary state: -1, 0, or 1 */
    int is_binary;
    int risk_score;
};

struct axion_state {
```

```
    struct resource_state res_history[WORKLOAD_HISTORY_SIZE];

    int res_history_index;

    struct rl_model rl;

    struct tbin_state tbin;

    int tbin_confidence_metric;

    int tbin_execution_profile;

    struct package packages[MAX_PACKAGES];

    int package_count;

    char suggestion[256];

    char last_cmd[256];

    unsigned long last_suggestion_time;

    int python_usage;

    int gaming_usage;

    uint64_t axion_register;

    int workload_history[WORKLOAD_HISTORY_SIZE];

    int workload_index;

    int adaptive_threshold;

    int confidence_metric;

    int rollback_counter;

    int rollback_reason[WORKLOAD_HISTORY_SIZE];

    bool rollback_suppression;

    int suppression_resistance;

    double resource_weight_cpu;

    double resource_weight_gpu;

    double resource_weight_ram;

    int resource_adjustment_log[WORKLOAD_HISTORY_SIZE];

};

```

```
417.
418. /* IOCTL command definitions */
419.
420. #define AXION_SET_REGISTER      _IOW('a', 1, uint64_t)
421.
422. #define AXION_GET_REGISTER      _IOR('a', 2, uint64_t)
423.
424. #define AXION_TBIN_LOAD         _IOW('a', 3, struct tbin_header)
425.
426. #define AXION_TBIN_STEP         _IO('a',  4)
427.
428. #define AXION_TBIN_GET_STATE    _IOR('a', 5, struct tbin_state)
429.
430. #define AXION_GET_SUGGESTION    _IOR('a', 6, char[256])
431.
432. #define AXION_INSTALL_PKG       _IOW('a', 7, char[32])
433.
434. #define AXION_UPDATE_PKG        _IOW('a', 8, char[32])
435.
436. #define AXION_SET_BINARY        _IOW('a', 9, int)
437.
438. #define AXION_ROLLBACK          _IOW('a', 10, char[32])
439.
440. #define AXION_NL_COMMAND        _IOW('a', 11, char[256])
441.
442. #define AXION_GET_PERF_FEEDBACK _IOR('a', 12, int)
443.
444.
445. /* Global variables for the module */
446.
447. static dev_t dev_num;
448.
449. static struct cdev axion_cdev;
450.
451. static struct class *axion_class;
452.
453. static struct device *axion_device;
454.
455. static struct task_struct *axion_thread;
456.
457. static struct dentry *debugfs_dir, *debugfs_file;
458.
459. static struct workqueue_struct *axion_wq;
460.
461. static struct work_struct axion_work;
462.
463. static struct timer_list axion_load_balancer;
464.
465. static struct axion_state state = {
466.
467.     .rl = { .q_table = {{5,2,1}, {3,5,2}, {1,3,5}}, .last_state = 0, .last_action = 0 },
468.
469.     .tbin_confidence_metric = 100,
470.
```

```
    .tbin_execution_profile = 0,
    .axion_register = AXION_DEFAULT_REGISTER,
    .adaptive_threshold = SELF_HEALING_THRESHOLD,
    .confidence_metric = 100,
    .resource_weight_cpu = RESOURCE_WEIGHT_CPU,
    .resource_weight_gpu = RESOURCE_WEIGHT_GPU,
    .resource_weight_ram = RESOURCE_WEIGHT_RAM,
    .package_count = 0,
    .res_history_index = 0,
    .workload_index = 0,
    .rollback_counter = 0,
    .rollback_suppression = false,
    .suppression_resistance = 0
};

/* Resource monitoring functions (simplified) */
static int get_cpu_usage(void) { return 50; }
static int get_ram_usage(void) { return 60; }
static int get_gpu_usage(void) { return 40; }

static void axion_get_resource_usage(struct resource_state *res) {
    res->cpu_usage = get_cpu_usage();
    res->ram_usage = get_ram_usage();
    res->gpu_usage = get_gpu_usage();
    res->action = 0;
}

static void axion_adjust_resource_weights(void) {
    int cpu_usage = get_cpu_usage();
```

```
525.
526.    int gpu_usage = get_gpu_usage();
527.
528.    int ram_usage = get_ram_usage();
529.
530.    if (cpu_usage > gpu_usage && cpu_usage > ram_usage)
531.
532.        state.resource_weight_cpu += FEEDBACK_ADJUSTMENT_FACTOR;
533.
534.    else if (gpu_usage > cpu_usage && gpu_usage > ram_usage)
535.
536.        state.resource_weight_gpu += FEEDBACK_ADJUSTMENT_FACTOR;
537.
538.    else if (ram_usage > cpu_usage && ram_usage > gpu_usage)
539.
540.        state.resource_weight_ram += FEEDBACK_ADJUSTMENT_FACTOR;
541.
542.    double total = state.resource_weight_cpu + state.resource_weight_gpu +
    state.resource_weight_ram;
543.
544.    state.resource_weight_cpu /= total;
545.
546.    state.resource_weight_gpu /= total;
547.
548.    state.resource_weight_ram /= total;
549.
550.    state.resource_adjustment_log[state.workload_index %
    WORKLOAD_HISTORY_SIZE] =
551.
552.        (int)(state.resource_weight_cpu * 100);
553.
554.}
555.
556.
557.static void axion_predictive_load_balancer(struct timer_list *t) {
558.
559.    axion_adjust_resource_weights();
560.
561.    mod_timer(&axion_load_balancer, jiffies +
    msecs_to_jiffies(PREDICTIVE_LOAD_BALANCING_INTERVAL));
562.
563.}
564.
565.
566./* Ternary execution functions */
567.
568.static int axion_tbin_step(void) {
569.
570.    if (!state.tbin.running || state.tbin.ip >= state.tbin.code_size - 2 || !state.tbin.code) {
571.
572.        printk(KERN_ERR "Axion: Invalid TBIN state for execution\n");
573.
574.        return -EINVAL;
575.
```

```
576.    }
577.
578.    uint8_t *pc = (uint8_t *)state.tbin.code;
579.
580.    uint8_t opcode = pc[state.tbin.ip];
581.
582.    int8_t t1 = (pc[state.tbin.ip + 1] == 0xFF) ? TERNARY_NEGATIVE : pc[state.tbin.ip +
    1];
583.
584.    int8_t t2 = (pc[state.tbin.ip + 2] == 0xFF) ? TERNARY_NEGATIVE : pc[state.tbin.ip +
    2];
585.
586.    if (state.tbin.ip + 2 >= state.tbin.code_size) {
587.
588.        printk(KERN_ERR "Axion: TBIN IP out of bounds\n");
589.
590.        return -EFAULT;
591.
592.    }
593.
594.    switch (opcode) {
595.
596.        case TADD:
597.
598.            state.tbin.reg[0] += t1;
599.
600.            if (state.tbin.reg[0] > TERNARY_POSITIVE) state.tbin.reg[0] =
    TERNARY_POSITIVE;
601.
602.            if (state.tbin.reg[0] < TERNARY_NEGATIVE) state.tbin.reg[0] =
    TERNARY_NEGATIVE;
603.
604.            break;
605.
606.        case TSUB:
607.
608.            state.tbin.reg[0] -= t1;
609.
610.            if (state.tbin.reg[0] > TERNARY_POSITIVE) state.tbin.reg[0] =
    TERNARY_POSITIVE;
611.
612.            if (state.tbin.reg[0] < TERNARY_NEGATIVE) state.tbin.reg[0] =
    TERNARY_NEGATIVE;
613.
614.            break;
615.
616.        case TMUL:
617.
618.            state.tbin.reg[0] *= t1;
619.
620.            if (state.tbin.reg[0] > TERNARY_POSITIVE) state.tbin.reg[0] =
    TERNARY_POSITIVE;
621.
```

```
622.            if (state.tbin.reg[0] < TERNARY_NEGATIVE) state.tbin.reg[0] =
    TERNARY_NEGATIVE;
623.
624.            break;
625.
626.        case TAND:
627.
628.            state.tbin.reg[0] = (state.tbin.reg[0] < t1) ? state.tbin.reg[0] : t1;
629.
630.            break;
631.
632.        case TOR:
633.
634.            state.tbin.reg[0] = (state.tbin.reg[0] > t1) ? state.tbin.reg[0] : t1;
635.
636.            break;
637.
638.        case TNOT:
639.
640.            state.tbin.reg[0] = -t1;
641.
642.            break;
643.
644.        case TJMP:
645.
646.            if (t2 != 0 && t1 * 3 < state.tbin.code_size)
647.
648.                state.tbin.ip = t1 * 3;
649.
650.            else
651.
652.                return -EFAULT;
653.
654.            break;
655.
656.        case TJZ:
657.
658.            if (t2 == 0 && t1 * 3 < state.tbin.code_size)
659.
660.                state.tbin.ip = t1 * 3;
661.
662.            else
663.
664.                return -EFAULT;
665.
666.            break;
667.
668.        case TJNZ:
669.
670.            if (t2 != 0 && t1 * 3 < state.tbin.code_size)
671.
672.                state.tbin.ip = t1 * 3;
673.
674.            else
```

```
675.
676.             return -EFAULT;
677.
678.         break;
679.
680.     case TLOAD:
681.
682.         if (t1 >= 0 && t1 < TERNARY_MEM_SIZE && t2 >= 0 && t2 < 3)
683.
684.             state.tbin.reg[t2] = state.tbin.memory[t1];
685.
686.         else
687.
688.             return -EFAULT;
689.
690.         break;
691.
692.     case TSTORE:
693.
694.         if (t1 >= 0 && t1 < TERNARY_MEM_SIZE && t2 >= 0 && t2 < 3)
695.
696.             state.tbin.memory[t1] = state.tbin.reg[t2];
697.
698.         else
699.
700.             return -EFAULT;
701.
702.         break;
703.
704.     case THLT:
705.
706.         state.tbin.running = 0;
707.
708.         printk(KERN_INFO "Axion: TBIN halted\n");
709.
710.         return 0;
711.
712.     default:
713.
714.         printk(KERN_ERR "Axion: Unknown TBIN opcode 0x%x\n", opcode);
715.
716.         return -EINVAL;
717.
718.   }
719.
720.   state.tbin.ip += 3;
721.
722.   return 0;
723.
724.}
725.
726.
727.static int axion_jit_compile_tbin(struct tbin_header *hdr) {
728.
```

```
    if (!hdr || hdr->code_size < 3) {

        printk(KERN_ERR "Axion: Invalid TBIN header\n");

        return -EINVAL;

    }

    if (state.tbin.code) {

        vfree(state.tbin.code);

        state.tbin.code = NULL;

    }

    int execution_efficiency = (hdr->code_size > 1024) ? TERNARY_POSITIVE :
TERNARY_NEGATIVE;

    state.tbin_execution_profile = (state.tbin_execution_profile + execution_efficiency) /
2;

    state.tbin_confidence_metric = (execution_efficiency !=
state.tbin_execution_profile) ?

                                state.tbin_confidence_metric - 5 : state.tbin_confidence_metric +
3;

    if (state.tbin_confidence_metric > 100) state.tbin_confidence_metric = 100;

    if (state.tbin_confidence_metric < 50) state.tbin_confidence_metric = 50;

    state.tbin.code = vmalloc(hdr->code_size);

    if (!state.tbin.code) {

        printk(KERN_ERR "Axion: Failed to allocate TBIN memory\n");

        return -ENOMEM;

    }

    if (copy_from_user(state.tbin.code, (void __user *)hdr->entry_point, hdr->code_size)) {

        printk(KERN_ERR "Axion: Failed to copy TBIN code\n");

        vfree(state.tbin.code);

        state.tbin.code = NULL;

        return -EFAULT;

    }

```

```
    if (hdr->code_size % 3 != 0) {
        printk(KERN_ERR "Axion: Invalid TBIN code size\n");
        vfree(state.tbin.code);
        state.tbin.code = NULL;
        return -EINVAL;
    }
    state.tbin.code_size = hdr->code_size;
    state.tbin.ip = 0;
    state.tbin.running = 1;
    memset(state.tbin.reg, 0, sizeof(state.tbin.reg));
    memset(state.tbin.memory, 0, sizeof(state.tbin.memory));
    printk(KERN_INFO "Axion: TBIN loaded for ternary execution\n");
    return 0;
}

static int load_tbin_binary(struct linux_binprm *bprm) {
    struct tbin_header hdr;
    if (bprm->buf[0] != 'T' || bprm->buf[1] != 'B' ||
        bprm->buf[2] != 'I' || bprm->buf[3] != 'N')
        return -ENOEXEC;
    memcpy(&hdr, bprm->buf, sizeof(hdr));
    if (hdr.magic != TBIN_MAGIC) return -ENOEXEC;
    return axion_jit_compile_tbin(&hdr);
}

static struct linux_binfmt axion_tbin_format = {
    .module = THIS_MODULE,
    .load_binary = load_tbin_binary,

```

```
};

/* Package management functions */
static void axion_predict_needs(void) {
    if (state.python_usage > 50)
        snprintf(state.suggestion, sizeof(state.suggestion),
                 "Python usage high—install PyTorch or NumPy?");
    else if (state.gaming_usage > 70)
        snprintf(state.suggestion, sizeof(state.suggestion),
                 "Gaming detected—optimize with GPU tools?");
    else
        state.suggestion[0] = '\0';
}

static int axion_resolve_deps(const char *pkg_name, int is_binary) {
    if (!pkg_name) return -EINVAL;
    for (int i = 0; i < state.package_count; i++) {
        if (strcmp(state.packages[i].name, pkg_name) == 0 &&
            state.packages[i].is_binary != is_binary) {
            printk(KERN_WARNING "Axion: Dependency conflict for %s\n", pkg_name);
            return -EAGAIN;
        }
    }
    return 0;
}

static int axion_install_pkg(const char *pkg_name, int is_binary) {
    if (!pkg_name) return -EINVAL;
    if (state.package_count >= MAX_PACKAGES) return -ENOMEM;
```

```
    if (axion_resolve_deps(pkg_name, is_binary) < 0) return -EAGAIN;
    struct package *pkg = &state.packages[state.package_count++];
    strncpy(pkg->name, pkg_name, sizeof(pkg->name)-1);
    pkg->name[sizeof(pkg->name)-1] = '\0';
    snprintf(pkg->version, sizeof(pkg->version), "1.0.%d", (int)(get_random_int() % 10));
    pkg->state = TERNARY_POSITIVE;
    pkg->is_binary = is_binary;
    pkg->risk_score = 0;
    pkg->dep_count = 0;
    if (strcmp(pkg_name, "python") == 0)
        strncpy(pkg->deps[pkg->dep_count++], "libc", sizeof(pkg->deps[0]));
    return 0;
}

static int axion_update_pkg(const char *pkg_name) {
    if (!pkg_name) return -EINVAL;
    for (int i = 0; i < state.package_count; i++) {
        if (strcmp(state.packages[i].name, pkg_name) == 0) {
            int risk = (int)(get_random_int() % 100);
            state.packages[i].risk_score = risk;
            snprintf(state.packages[i].version, sizeof(state.packages[i].version),
                     "1.1.%d", (int)(get_random_int() % 10));
            if (risk > 70) {
                state.packages[i].state = TERNARY_ZERO;
                snprintf(state.suggestion, sizeof(state.suggestion),
                         "Update to %s risky—rollback?", pkg_name);
            } else {
```

```
            state.packages[i].state = TERNARY_POSITIVE;

        }

        return 0;

    }

  }

  return -ENOENT;

}


static int axion_rollback_pkg(const char *pkg_name) {

  if (!pkg_name) return -EINVAL;

  for (int i = 0; i < state.package_count; i++) {

    if (strcmp(state.packages[i].name, pkg_name) == 0) {

      snprintf(state.packages[i].version, sizeof(state.packages[i].version), "1.0.0");

      state.packages[i].risk_score = 0;

      state.packages[i].state = TERNARY_POSITIVE;

      state.rollback_counter++;

      state.rollback_reason[state.rollback_counter % WORKLOAD_HISTORY_SIZE] =
  1;

      return 0;

    }

  }

  return -ENOENT;

}


static int axion_nl_command(const char *cmd) {

  if (!cmd) return -EINVAL;

  strncpy(state.last_cmd, cmd, sizeof(state.last_cmd)-1);

  state.last_cmd[sizeof(state.last_cmd)-1] = '\0';

  if (strstr(cmd, "install a lightweight browser"))
```

```
994.
995.       return axion_install_pkg("lynx", 1);
996.
997.    if (strstr(cmd, "update everything except my GPU drivers")) {
998.
999.       for (int i = 0; i < state.package_count; i++) {
1000.
1001.          if (strstr(state.packages[i].name, "nvidia") == NULL)
1002.
1003.             axion_update_pkg(state.packages[i].name);
1004.
1005.       }
1006.
1007.       return 0;
1008.
1009.    }
1010.
1011.    if (strstr(cmd, "optimize my gaming setup")) {
1012.
1013.       axion_install_pkg("nvidia-driver", 1);
1014.
1015.       snprintf(state.suggestion, sizeof(state.suggestion), "Gaming optimized!");
1016.
1017.       return 0;
1018.
1019.    }
1020.
1021.    return -EINVAL;
1022.
1023.}
1024.
1025.
1026.static int axion_get_perf_feedback(void) {
1027.
1028.    return (int)(get_random_int() % 100);
1029.
1030.}
1031.
1032.
1033./* Workload and suggestion work function */
1034.
1035.static void axion_suggestion_work(struct work_struct *work) {
1036.
1037.    int current_load = state.workload_history[(state.workload_index == 0 ?
    WORKLOAD_HISTORY_SIZE - 1 : state.workload_index - 1)];
1038.
1039.    if (current_load > ANOMALY_THRESHOLD)
1040.
1041.       printk(KERN_WARNING "Axion: Anomaly detected - Load: %d%%\n",
    current_load);
1042.
1043.    axion_predict_needs();
1044.
1045.    schedule_work(&axion_work);
```

```
1046.
1047.}
1048.
1049.
1050./* Module initialization and exit functions */
1051.
1052.static int __init axion_init(void) {
1053.
1054.    int ret = alloc_chrdev_region(&dev_num, 0, 1, DEVICE_NAME);
1055.
1056.    if (ret < 0) {
1057.
1058.        printk(KERN_ERR "Axion: Failed to allocate device number\n");
1059.
1060.        return ret;
1061.
1062.    }
1063.
1064.    cdev_init(&axion_cdev, NULL);
1065.
1066.    ret = cdev_add(&axion_cdev, dev_num, 1);
1067.
1068.    if (ret < 0) {
1069.
1070.        printk(KERN_ERR "Axion: Failed to add cdev\n");
1071.
1072.        unregister_chrdev_region(dev_num, 1);
1073.
1074.        return ret;
1075.
1076.    }
1077.
1078.    axion_class = class_create(THIS_MODULE, DEVICE_NAME);
1079.
1080.    if (IS_ERR(axion_class)) {
1081.
1082.        cdev_del(&axion_cdev);
1083.
1084.        unregister_chrdev_region(dev_num, 1);
1085.
1086.        return PTR_ERR(axion_class);
1087.
1088.    }
1089.
1090.    axion_device = device_create(axion_class, NULL, dev_num, NULL,
    DEVICE_NAME);
1091.
1092.    if (IS_ERR(axion_device)) {
1093.
1094.        class_destroy(axion_class);
1095.
1096.        cdev_del(&axion_cdev);
1097.
1098.        unregister_chrdev_region(dev_num, 1);
```

```
1099.
1100.       return PTR_ERR(axion_device);
1101.
1102.    }
1103.
1104.    debugfs_dir = debugfs_create_dir(AXION_DEBUGFS_DIR, NULL);
1105.
1106.    debugfs_file = debugfs_create_file(AXION_DEBUGFS_FILE, 0444, debugfs_dir,
    &state, NULL);
1107.
1108.    timer_setup(&axion_load_balancer, axion_predictive_load_balancer, 0);
1109.
1110.    mod_timer(&axion_load_balancer, jiffies +
    msecs_to_jiffies(PREDICTIVE_LOAD_BALANCING_INTERVAL));
1111.
1112.    axion_wq = create_singlethread_workqueue("axion_wq");
1113.
1114.    INIT_WORK(&axion_work, axion_suggestion_work);
1115.
1116.    printk(KERN_INFO "Axion: Module initialized\n");
1117.
1118.    return 0;
1119.
1120.}
1121.
1122.
1123.static void __exit axion_exit(void) {
1124.
1125.    del_timer_sync(&axion_load_balancer);
1126.
1127.    device_destroy(axion_class, dev_num);
1128.
1129.    class_destroy(axion_class);
1130.
1131.    cdev_del(&axion_cdev);
1132.
1133.    unregister_chrdev_region(dev_num, 1);
1134.
1135.    debugfs_remove_recursive(debugfs_dir);
1136.
1137.    flush_workqueue(axion_wq);
1138.
1139.    destroy_workqueue(axion_wq);
1140.
1141.    printk(KERN_INFO "Axion: Module exited\n");
1142.
1143.}
1144.
1145.
1146.module_init(axion_init);
1147.
1148.module_exit(axion_exit);
1149.
1150.MODULE_LICENSE("GPL");
```

```
1151.
1152. MODULE_AUTHOR("Your Name");
1153.
1154. MODULE_DESCRIPTION("Axion: AI-Powered Kernel Module for Ternary Logic
     Evolution");
1155.
1156.
1157. @* End of Kernel Module section. @*/
1158.
1159. #endif /* __KERNEL__ */
1160.
1161.
1162. @*
     ==========================================================================
     =====
1163.
1164.   Section 3: User–Space Utility Program (TritJS-CISA-Optimized)
1165.
1166.   This section implements our advanced ternary calculator utility with an
1167.
1168.   interactive ncurses–based interface. It supports advanced arithmetic,
1169.
1170.   scientific functions, secure logging, and a built–in help system (type "help").
1171.
1172.   This section is compiled when __KERNEL__ is not defined.
1173.
1174.
     ==========================================================================
     ===== */
1175.
1176. #ifndef __KERNEL__
1177.
1178. @<TritJS-CISA-Optimized Utility@>=
1179.
1180. /*****************************************************************
1181.
1182.  * TritJS-CISA-Optimized: Advanced Ternary Calculator Utility
1183.
1184.  *
1185.
1186.  * FEATURES:
1187.
1188.  *   - Safe dynamic memory management and optimized arithmetic operations,
1189.
1190.  *     including Karatsuba multiplication with caching.
1191.
1192.  *   - Scientific functions (sqrt, log3, sin, cos, tan, π) via double conversion.
1193.
1194.  *   - Secure audit logging and simulated intrusion detection.
1195.
1196.  *   - Interactive ncurses–based command–line interface with dynamic resizing.
1197.
1198.  *   - Built–in help system: Type "help" at the prompt for command details.
1199.
```

```
1200. *
1201.
1202. * USAGE:
1203.
1204. *   Compile with:
1205.
1206. *     gcc -DUSE_READLINE -o tritjs_cisa_optimized ternary_system.cweb -lm
    -lreadline \
1207.
1208. *         -fstack-protector-strong -D_FORTIFY_SOURCE=2 -pie -fPIE -lncurses -lcrypto
    -lpthread -llua
1209.
1210. *   Then run:
1211.
1212. *     ./tritjs_cisa_optimized
1213.
1214. *
1215.
1216. * This utility shares a unified ternary logic foundation with the Axion kernel module,
1217.
1218. * helping to advance our mission toward native ternary computing.
1219.
1220. ************************************************************************/
1221.
1222.
1223.#include <stdio.h>
1224.
1225.#include <stdlib.h>
1226.
1227.#include <string.h>
1228.
1229.#include <math.h>
1230.
1231.#include <sys/mman.h>
1232.
1233.#include <fcntl.h>
1234.
1235.#include <unistd.h>
1236.
1237.#include <time.h>
1238.
1239.#include <limits.h>
1240.
1241.#include <ncurses.h>
1242.
1243.#include <errno.h>
1244.
1245.#include <sys/file.h>
1246.
1247.#include <openssl/evp.h>
1248.
1249.#include <pthread.h>
1250.
1251.#include <lua.h>
```

```
#include <lualib.h>

#include <lauxlib.h>

#include "ternary_common.h"


#ifdef USE_READLINE

#include <readline/readline.h>

#include <readline/history.h>

#endif


/* Global configuration */

#define ENABLE_VERBOSE_LOGGING 1

#define VERSION "2.0-upgrade-optimized"

#define BASE_81 81

#define T81_MMAP_THRESHOLD (500 * 1024)


/* Data structures for big integer arithmetic */

typedef struct {

    int sign;                /* 0 = positive, 1 = negative */

    unsigned char *digits;    /* Array of base–81 digits (little–endian) */

    size_t len;              /* Number of digits */

    int is_mapped;           /* 1 if allocated with mmap */

    int fd;                  /* File descriptor if using mmap */

    char tmp_path[32];       /* Temporary file path */

} T81BigInt;


typedef struct {

    int sign;

    unsigned char* integer;   /* Digits for integer part */

    unsigned char* fraction;  /* Digits for fractional part */
```

```
    size_t i_len, f_len;
    int i_mapped, f_mapped;
    int i_fd, f_fd;
    char i_tmp_path[32];
    char f_tmp_path[32];
} T81Float;

typedef struct {
    T81Float real;
    T81Float imag;
} T81Complex;

typedef struct {
    T81Float quotient;
    T81Float remainder;
} T81DivResult;

#define MAX_SCRIPT_NAME 10
#define MAX_SCRIPT_CMDS 50
typedef struct {
    char name[MAX_SCRIPT_NAME];
    char commands[MAX_SCRIPT_CMDS][256];
    int cmd_count;
} Script;

/* Global variables for the utility */
static FILE* audit_log = NULL;
static long total_mapped_bytes = 0;
static int operation_steps = 0;
```

```
#define MAX_HISTORY 10

static char* history[MAX_HISTORY] = {0};

static int history_count = 0;

static T81BigInt* variables[26] = {0};

static Script scripts[10] = {0};

static int script_count = 0;

static WINDOW *input_win, *output_win, *status_win;


/* Function prototypes for arithmetic and scientific operations */

TernaryError tritjs_add_big(T81BigInt* A, T81BigInt* B, T81BigInt** result);

TernaryError tritjs_subtract_big(T81BigInt* A, T81BigInt* B, T81BigInt** result);

TernaryError tritjs_multiply_big(T81BigInt* a, T81BigInt* b, T81BigInt** result);

TernaryError tritjs_factorial_big(T81BigInt* a, T81BigInt** result);

TernaryError tritjs_power_big(T81BigInt* base, T81BigInt* exp, T81BigInt** result);

TernaryError tritjs_divide_big(T81BigInt* a, T81BigInt* b, T81BigInt** quotient,
    T81BigInt** remainder);

TernaryError tritjs_sqrt_complex(T81BigInt* a, int precision, T81Complex* result);

TernaryError tritjs_log3_complex(T81BigInt* a, int precision, T81Complex* result);

TernaryError tritjs_sin_complex(T81BigInt* a, int precision, T81Complex* result);

TernaryError tritjs_cos_complex(T81BigInt* a, int precision, T81Complex* result);

TernaryError tritjs_tan_complex(T81BigInt* a, int precision, T81Complex* result);

TernaryError tritjs_pi(int* len, int** pi);

TernaryError parse_trit_string(const char* s, T81BigInt** out);

TernaryError t81bigint_to_trit_string(const T81BigInt* in, char** out);

TernaryError binary_to_trit(int num, T81BigInt** out);

TernaryError trit_to_binary(T81BigInt* x, int* outVal);

void tritbig_free(T81BigInt* x);

TernaryError parse_balanced_trit_string(const char* s, T81BigInt** out);
```

```
1413.
1414.
1415./* Logging and error handling functions */
1416.
1417.static const char* trit_error_str(TernaryError err) {
1418.
1419.    switch(err){
1420.
1421.        case TERNARY_NO_ERROR: return "No error";
1422.
1423.        case TERNARY_ERR_MEMALLOC: return "Memory allocation failed";
1424.
1425.        case TERNARY_ERR_INVALID_INPUT: return "Invalid input";
1426.
1427.        case TERNARY_ERR_DIVZERO: return "Division by zero";
1428.
1429.        case TERNARY_ERR_OVERFLOW: return "Overflow detected";
1430.
1431.        case TERNARY_ERR_UNDEFINED: return "Operation undefined";
1432.
1433.        case TERNARY_ERR_NEGATIVE: return "Negative input";
1434.
1435.        case TERNARY_ERR_PRECISION: return "Precision limit exceeded";
1436.
1437.        case TERNARY_ERR_MMAP: return "Memory mapping failed";
1438.
1439.        case TERNARY_ERR_SCRIPT: return "Script error";
1440.
1441.        default: return "Unknown error";
1442.
1443.    }
1444.
1445.}
1446.
1447.
1448.static void log_error(TernaryError err, const char* context, const char* file, int line) {
1449.
1450.    if (!audit_log) return;
1451.
1452.    time_t now;
1453.
1454.    time(&now);
1455.
1456.    fprintf(audit_log, "[%s] ERROR %d: %s in %s (%s:%d)\n",
1457.
1458.            ctime(&now), err, trit_error_str(err), context, file, line);
1459.
1460.    fflush(audit_log);
1461.
1462.}
1463.
1464.
1465./* Memory management for T81BigInt */
1466.
```

```
1467. static TernaryError allocate_digits(T81BigInt *x, size_t lengthNeeded) {
1468.
1469.     size_t bytesNeeded = (lengthNeeded == 0 ? 1 : lengthNeeded);
1470.
1471.     x->len = lengthNeeded;
1472.
1473.     x->is_mapped = 0;
1474.
1475.     x->fd = -1;
1476.
1477.     if (bytesNeeded < T81_MMAP_THRESHOLD) {
1478.
1479.         x->digits = (unsigned char*)calloc(bytesNeeded, 1);
1480.
1481.         if (!x->digits) return TERNARY_ERR_MEMALLOC;
1482.
1483.         return TERNARY_NO_ERROR;
1484.
1485.     }
1486.
1487.     strcpy(x->tmp_path, "/tmp/tritjs_cisa_XXXXXX");
1488.
1489.     x->fd = mkstemp(x->tmp_path);
1490.
1491.     if (x->fd < 0) return TERNARY_ERR_MMAP;
1492.
1493.     if (ftruncate(x->fd, bytesNeeded) < 0) {
1494.
1495.         close(x->fd);
1496.
1497.         return TERNARY_ERR_MMAP;
1498.
1499.     }
1500.
1501.     x->digits = mmap(NULL, bytesNeeded, PROT_READ | PROT_WRITE,
    MAP_SHARED, x->fd, 0);
1502.
1503.     if (x->digits == MAP_FAILED) {
1504.
1505.         close(x->fd);
1506.
1507.         return TERNARY_ERR_MMAP;
1508.
1509.     }
1510.
1511.     unlink(x->tmp_path);
1512.
1513.     x->is_mapped = 1;
1514.
1515.     total_mapped_bytes += bytesNeeded;
1516.
1517.     operation_steps++;
1518.
1519.     return TERNARY_NO_ERROR;
```

```
}

static void t81bigint_free(T81BigInt* x) {
    if (!x) return;
    if (x->is_mapped && x->digits && x->digits != MAP_FAILED) {
        size_t bytes = (x->len == 0 ? 1 : x->len);
        munmap(x->digits, bytes);
        close(x->fd);
        total_mapped_bytes -= bytes;
        operation_steps++;
    } else {
        free(x->digits);
    }
}

/* Audit logging initialization */
static void init_audit_log() {
    audit_log = fopen("/var/log/tritjs_cisa.log", "a");
    if (!audit_log) {
        perror("Audit log init failed; fallback to stderr");
        audit_log = stderr;
    } else {
        flock(fileno(audit_log), LOCK_EX);
    }
}

/* Base conversion and parsing functions */
static TernaryError parse_trit_string_base81_optimized(const char* str, T81BigInt* out)
    {
```

```
1573.
1574.    if (!str || !str[0]) return TERNARY_ERR_INVALID_INPUT;
1575.
1576.    memset(out, 0, sizeof(*out));
1577.
1578.    int sign = 0;
1579.
1580.    size_t pos = 0;
1581.
1582.    if (str[0] == '-' || str[0] == '–') { sign = 1; pos = 1; }
1583.
1584.    size_t total_len = strlen(str) - pos;
1585.
1586.    size_t remainder = total_len % 4;
1587.
1588.    if (allocate_digits(out, 1) != TERNARY_NO_ERROR) return
    TERNARY_ERR_MEMALLOC;
1589.
1590.    out->digits[0] = 0;
1591.
1592.    out->sign = sign;
1593.
1594.    for (size_t i = 0; i < remainder; i++) {
1595.
1596.        int digit = str[pos + i] - '0';
1597.
1598.        if (digit < 0 || digit > 2) return TERNARY_ERR_INVALID_INPUT;
1599.
1600.        int carry = digit;
1601.
1602.        for (size_t j = 0; j < out->len; j++) {
1603.
1604.            int val = out->digits[j] * 3 + carry;
1605.
1606.            out->digits[j] = val % BASE_81;
1607.
1608.            carry = val / BASE_81;
1609.
1610.        }
1611.
1612.        while (carry) {
1613.
1614.            size_t old_len = out->len;
1615.
1616.            if (allocate_digits(out, out->len + 1) != TERNARY_NO_ERROR) return
    TERNARY_ERR_MEMALLOC;
1617.
1618.            out->digits[old_len] = carry % BASE_81;
1619.
1620.            carry /= BASE_81;
1621.
1622.        }
1623.
1624.    }
```

```
1625.
1626.     pos += remainder;
1627.
1628.     while (pos < strlen(str)) {
1629.
1630.         int groupVal = 0;
1631.
1632.         for (int k = 0; k < 4; k++) {
1633.
1634.             if (str[pos + k] < '0' || str[pos + k] > '2') return TERNARY_ERR_INVALID_INPUT;
1635.
1636.             groupVal = groupVal * 3 + (str[pos + k] - '0');
1637.
1638.         }
1639.
1640.         pos += 4;
1641.
1642.         int carry = groupVal;
1643.
1644.         for (size_t j = 0; j < out->len; j++) {
1645.
1646.             int val = out->digits[j] * 81 + carry;
1647.
1648.             out->digits[j] = val % BASE_81;
1649.
1650.             carry = val / BASE_81;
1651.
1652.         }
1653.
1654.         while (carry) {
1655.
1656.             size_t old_len = out->len;
1657.
1658.             if (allocate_digits(out, out->len + 1) != TERNARY_NO_ERROR) return
    TERNARY_ERR_MEMALLOC;
1659.
1660.             out->digits[old_len] = carry % BASE_81;
1661.
1662.             carry /= BASE_81;
1663.
1664.         }
1665.
1666.     }
1667.
1668.     while (out->len > 1 && out->digits[out->len - 1] == 0)
1669.
1670.         out->len--;
1671.
1672.     return TERNARY_NO_ERROR;
1673.
1674. }
1675.
1676.
1677. static TernaryError parse_trit_string(const char* s, T81BigInt** out) {
```

```
1678.
1679.     if (!out) return TERNARY_ERR_INVALID_INPUT;
1680.
1681.     *out = (T81BigInt*)calloc(1, sizeof(T81BigInt));
1682.
1683.     if (!*out) return TERNARY_ERR_MEMALLOC;
1684.
1685.     TernaryError e = parse_trit_string_base81_optimized(s, *out);
1686.
1687.     if (e != TERNARY_NO_ERROR) { free(*out); *out = NULL; }
1688.
1689.     return e;
1690.
1691. }
1692.
1693.
1694. static TernaryError t81bigint_to_trit_string(const T81BigInt* in, char** out) {
1695.
1696.     if (!in || !out) return TERNARY_ERR_INVALID_INPUT;
1697.
1698.     if (in->len == 1 && in->digits[0] == 0) {
1699.
1700.         *out = strdup("0");
1701.
1702.         return TERNARY_NO_ERROR;
1703.
1704.     }
1705.
1706.     T81BigInt tmp = *in;
1707.
1708.     T81BigInt tmpCopy;
1709.
1710.     memset(&tmpCopy, 0, sizeof(tmpCopy));
1711.
1712.     if (allocate_digits(&tmpCopy, tmp.len) != TERNARY_NO_ERROR) return
    TERNARY_ERR_MEMALLOC;
1713.
1714.     tmpCopy.len = tmp.len;
1715.
1716.     memcpy(tmpCopy.digits, tmp.digits, tmp.len);
1717.
1718.     tmpCopy.sign = tmp.sign;
1719.
1720.     size_t capacity = tmp.len * 4 + 2;
1721.
1722.     char* buf = calloc(capacity, 1);
1723.
1724.     if (!buf) { t81bigint_free(&tmpCopy); return TERNARY_ERR_MEMALLOC; }
1725.
1726.     size_t idx = 0;
1727.
1728.     while (1) {
1729.
1730.         int isZero = 1;
```

```
        for (size_t i = 0; i < tmpCopy.len; i++) {
            if (tmpCopy.digits[i] != 0) { isZero = 0; break; }
        }
        if (isZero) { if (idx == 0) buf[idx++] = '0'; break; }
        int carry = 0;
        for (ssize_t i = tmpCopy.len - 1; i >= 0; i--) {
            int val = tmpCopy.digits[i] + carry * BASE_81;
            int q = val / 3;
            int r = val % 3;
            tmpCopy.digits[i] = q;
            carry = r;
        }
        buf[idx++] = (char)('0' + carry);
    }
    t81bigint_free(&tmpCopy);
    if (in->sign) { buf[idx++] = '-'; }
    for (size_t i = 0; i < idx / 2; i++) {
        char t = buf[i];
        buf[i] = buf[idx - 1 - i];
        buf[idx - 1 - i] = t;
    }
    buf[idx] = '\0';
    *out = buf;
    return TERNARY_NO_ERROR;
}

static TernaryError binary_to_trit(int num, T81BigInt** out) {
```

```
1785.     char b3[128];
1786.
1787.     int sign = (num < 0) ? 1 : 0;
1788.
1789.     int val = (num < 0) ? -num : num;
1790.
1791.     size_t idx = 0;
1792.
1793.     if (val == 0) { b3[idx++] = '0'; }
1794.
1795.     while (val > 0) { int r = val % 3; b3[idx++] = (char)('0' + r); val /= 3; }
1796.
1797.     if (idx == 0) { b3[idx++] = '0'; }
1798.
1799.     if (sign) { b3[idx++] = '-'; }
1800.
1801.     for (size_t i = 0; i < idx / 2; i++) {
1802.
1803.         char t = b3[i];
1804.
1805.         b3[i] = b3[idx - 1 - i];
1806.
1807.         b3[idx - 1 - i] = t;
1808.
1809.     }
1810.
1811.     b3[idx] = '\0';
1812.
1813.     return parse_trit_string(b3, out);
1814.
1815. }
1816.
1817.
1818. static TernaryError trit_to_binary(T81BigInt* x, int* outVal) {
1819.
1820.     char* b3 = NULL;
1821.
1822.     if (t81bigint_to_trit_string(x, &b3) != TERNARY_NO_ERROR) return
    TERNARY_ERR_INVALID_INPUT;
1823.
1824.     long long accum = 0;
1825.
1826.     int sign = 0;
1827.
1828.     size_t i = 0;
1829.
1830.     if (b3[0] == '-') { sign = 1; i = 1; }
1831.
1832.     for (; b3[i]; i++) {
1833.
1834.         if (b3[i] < '0' || b3[i] > '2') { free(b3); return TERNARY_ERR_INVALID_INPUT; }
1835.
1836.         accum = accum * 3 + (b3[i] - '0');
1837.
```

```
        if (accum > INT_MAX) { free(b3); return TERNARY_ERR_OVERFLOW; }
    }
    free(b3);
    if (sign) accum = -accum;
    *outVal = (int)accum;
    return TERNARY_NO_ERROR;
}

void tritbig_free(T81BigInt* x) {
    if (!x) return;
    t81bigint_free(x);
    free(x);
}

/* Balanced ternary parsing: convert balanced ternary string to T81BigInt */
TernaryError parse_balanced_trit_string(const char* s, T81BigInt** out) {
    if (!s) return TERNARY_ERR_INVALID_INPUT;
    size_t len = strlen(s);
    char* unb = calloc(len + 1, 1);
    if (!unb) return TERNARY_ERR_MEMALLOC;
    for (size_t i = 0; i < len; i++) {
        char c = s[i];
        if (c == '-' || c == '–') { unb[i] = '0'; }
        else if (c == '0') { unb[i] = '1'; }
        else if (c == '+') { unb[i] = '2'; }
        else { free(unb); return TERNARY_ERR_INVALID_INPUT; }
    }
    unb[len] = '\0';
```

```
1892.     TernaryError e = parse_trit_string(unb, out);
1893.
1894.     free(unb);
1895.
1896.     return e;
1897.
1898. }
1899.
1900.
1901. /* Arithmetic operations implementation */
1902.
1903.
1904. /* Addition */
1905.
1906. TernaryError tritjs_add_big(T81BigInt* A, T81BigInt* B, T81BigInt** result) {
1907.
1908.     if (!A || !B) return TERNARY_ERR_INVALID_INPUT;
1909.
1910.     *result = (T81BigInt*)calloc(1, sizeof(T81BigInt));
1911.
1912.     if (!*result) return TERNARY_ERR_MEMALLOC;
1913.
1914.     if (A->sign == B->sign) {
1915.
1916.         (*result)->sign = A->sign;
1917.
1918.         size_t len = (A->len > B->len ? A->len : B->len) + 1;
1919.
1920.         if (allocate_digits(*result, len) != TERNARY_NO_ERROR) { free(*result); return
    TERNARY_ERR_MEMALLOC; }
1921.
1922.         memset((*result)->digits, 0, len);
1923.
1924.         memcpy((*result)->digits, A->digits, A->len);
1925.
1926.         for (size_t i = 0; i < B->len; i++) {
1927.
1928.             int val = (*result)->digits[i] + B->digits[i];
1929.
1930.             (*result)->digits[i] = val % BASE_81;
1931.
1932.             int carry = val / BASE_81;
1933.
1934.             size_t cpos = i + 1;
1935.
1936.             while (carry && cpos < len) {
1937.
1938.                 val = (*result)->digits[cpos] + carry;
1939.
1940.                 (*result)->digits[cpos] = val % BASE_81;
1941.
1942.                 carry = val / BASE_81;
1943.
1944.                 cpos++;
```

```
1945.
1946.         }
1947.
1948.       }
1949.
1950.       while ((*result)->len > 1 && (*result)->digits[(*result)->len - 1] == 0)
1951.
1952.         (*result)->len--;
1953.
1954.     } else {
1955.
1956.       int c = memcmp(A->digits, B->digits, (A->len < B->len ? A->len : B->len));
1957.
1958.       T81BigInt *larger, *smaller;
1959.
1960.       int largerSign;
1961.
1962.       if (c > 0) { larger = A; smaller = B; largerSign = A->sign; }
1963.
1964.       else if (c < 0) { larger = B; smaller = A; largerSign = B->sign; }
1965.
1966.       else {
1967.
1968.         if (allocate_digits(*result, 1) != TERNARY_NO_ERROR) { free(*result); return
    TERNARY_ERR_MEMALLOC; }
1969.
1970.         (*result)->digits[0] = 0;
1971.
1972.         return TERNARY_NO_ERROR;
1973.
1974.       }
1975.
1976.       (*result)->sign = largerSign;
1977.
1978.       if (allocate_digits(*result, larger->len) != TERNARY_NO_ERROR) { free(*result);
    return TERNARY_ERR_MEMALLOC; }
1979.
1980.       memcpy((*result)->digits, larger->digits, larger->len);
1981.
1982.       for (size_t i = 0; i < smaller->len; i++) {
1983.
1984.         int diff = (*result)->digits[i] - smaller->digits[i];
1985.
1986.         if (diff < 0) {
1987.
1988.           diff += BASE_81;
1989.
1990.           size_t j = i + 1;
1991.
1992.           while (1) {
1993.
1994.             (*result)->digits[j]--;
1995.
1996.             if ((*result)->digits[j] < 255) break;
```

```
                (*result)->digits[j] += BASE_81;
                j++;
                if (j >= larger->len) break;
            }
        }
        (*result)->digits[i] = (unsigned char)diff;
    }
    while ((*result)->len > 1 && (*result)->digits[(*result)->len - 1] == 0)
        (*result)->len--;
}
    return TERNARY_NO_ERROR;
}

/* Subtraction */
TernaryError tritjs_subtract_big(T81BigInt* A, T81BigInt* B, T81BigInt** result) {
    if (!A || !B) return TERNARY_ERR_INVALID_INPUT;
    T81BigInt tmp = *B;
    int oldsign = tmp.sign;
    tmp.sign = !oldsign;
    TernaryError e = tritjs_add_big(A, &tmp, result);
    tmp.sign = oldsign;
    return e;
}

/* Multiplication using Karatsuba algorithm with caching */
#define MUL_CACHE_SIZE 8
typedef struct {
    char key[128];
```

```
    T81BigInt result;
    int used;
} MulCacheEntry;
static MulCacheEntry mul_cache[MUL_CACHE_SIZE] = {{0}};

static void naive_mul(const unsigned char *A, size_t alen,
                      const unsigned char *B, size_t blen,
                      unsigned char *out) {
    memset(out, 0, alen + blen);
    for (size_t i = 0; i < alen; i++) {
        int carry = 0;
        for (size_t j = 0; j < blen; j++) {
            int pos = i + j;
            int val = out[pos] + A[i] * B[j] + carry;
            out[pos] = val % BASE_81;
            carry = val / BASE_81;
        }
        out[i + blen] += carry;
    }
}

static void add_shifted(unsigned char *dest, size_t dlen,
                        const unsigned char *src, size_t slen,
                        size_t shift) {
    int carry = 0;
    for (size_t i = 0; i < slen; i++) {
        size_t idx = i + shift;
        if (idx >= dlen) break;
```

```
2105.
2106.       int sum = dest[idx] + src[i] + carry;
2107.
2108.       dest[idx] = sum % BASE_81;
2109.
2110.       carry = sum / BASE_81;
2111.
2112.     }
2113.
2114.     size_t idx = slen + shift;
2115.
2116.     while (carry && idx < dlen) {
2117.
2118.       int sum = dest[idx] + carry;
2119.
2120.       dest[idx] = sum % BASE_81;
2121.
2122.       carry = sum / BASE_81;
2123.
2124.       idx++;
2125.
2126.     }
2127.
2128. }
2129.
2130.
2131. static void sub_inplace(unsigned char* out, const unsigned char* src, size_t length) {
2132.
2133.     int borrow = 0;
2134.
2135.     for (size_t i = 0; i < length; i++) {
2136.
2137.       int diff = out[i] - src[i] - borrow;
2138.
2139.       if (diff < 0) { diff += BASE_81; borrow = 1; } else { borrow = 0; }
2140.
2141.       out[i] = diff;
2142.
2143.     }
2144.
2145. }
2146.
2147.
2148. static void karatsuba(const unsigned char *A, const unsigned char *B, size_t n,
    unsigned char *out) {
2149.
2150.     if (n <= 16) { naive_mul(A, n, B, n, out); return; }
2151.
2152.     size_t half = n / 2, r = n - half;
2153.
2154.     const unsigned char *A0 = A, *A1 = A + half;
2155.
2156.     const unsigned char *B0 = B, *B1 = B + half;
2157.
```

```
    size_t len2 = 2 * n;

    unsigned char *p1 = calloc(len2, 1);

    unsigned char *p2 = calloc(len2, 1);

    unsigned char *p3 = calloc(len2, 1);

    unsigned char *sumA = calloc(n, 1);

    unsigned char *sumB = calloc(n, 1);

    karatsuba(A0, B0, half, p1);

    karatsuba(A1, B1, r, p2);

    memcpy(sumA, A1, r);

    for (size_t i = 0; i < half; i++) {

        int s = sumA[i] + A0[i];

        sumA[i] = s % BASE_81;

        int c = s / BASE_81;

        if (c && i + 1 < n) sumA[i + 1] += c;

    }

    memcpy(sumB, B1, r);

    for (size_t i = 0; i < half; i++) {

        int s = sumB[i] + B0[i];

        sumB[i] = s % BASE_81;

        int c = s / BASE_81;

        if (c && i + 1 < n) sumB[i + 1] += c;

    }

    karatsuba(sumA, sumB, n, p3);

    sub_inplace(p3, p1, len2);

    sub_inplace(p3, p2, len2);

    memset(out, 0, len2);

    add_shifted(out, len2, p1, len2, 0);

```

```
2212.     add_shifted(out, len2, p3, len2, half);
2213.
2214.     add_shifted(out, len2, p2, len2, 2 * half);
2215.
2216.     free(p1); free(p2); free(p3);
2217.
2218.     free(sumA); free(sumB);
2219.
2220. }
2221.
2222.
2223. static TernaryError t81bigint_karatsuba_multiply(const T81BigInt *a, const T81BigInt
    *b, T81BigInt *out) {
2224.
2225.     if ((a->len == 1 && a->digits[0] == 0) || (b->len == 1 && b->digits[0] == 0)) {
2226.
2227.         if (allocate_digits(out, 1) != TERNARY_NO_ERROR) return
    TERNARY_ERR_MEMALLOC;
2228.
2229.         out->digits[0] = 0; out->sign = 0;
2230.
2231.         return TERNARY_NO_ERROR;
2232.
2233.     }
2234.
2235.     size_t n = (a->len > b->len ? a->len : b->len);
2236.
2237.     unsigned char *A = calloc(n, 1), *B = calloc(n, 1);
2238.
2239.     if (!A || !B) { free(A); free(B); return TERNARY_ERR_MEMALLOC; }
2240.
2241.     memcpy(A, a->digits, a->len);
2242.
2243.     memcpy(B, b->digits, b->len);
2244.
2245.     size_t out_len = 2 * n;
2246.
2247.     unsigned char *prod = calloc(out_len, 1);
2248.
2249.     if (!prod) { free(A); free(B); return TERNARY_ERR_MEMALLOC; }
2250.
2251.     karatsuba(A, B, n, prod);
2252.
2253.     free(A); free(B);
2254.
2255.     out->sign = (a->sign != b->sign) ? 1 : 0;
2256.
2257.     while (out_len > 1 && prod[out_len - 1] == 0) out_len--;
2258.
2259.     if (allocate_digits(out, out_len) != TERNARY_NO_ERROR) { free(prod); return
    TERNARY_ERR_MEMALLOC; }
2260.
2261.     memcpy(out->digits, prod, out_len);
2262.
```

```
    free(prod);

    return TERNARY_NO_ERROR;
}


static int mul_cache_lookup(const char* key, T81BigInt *dst) {

    for (int i = 0; i < MUL_CACHE_SIZE; i++) {

        if (mul_cache[i].used && strcmp(mul_cache[i].key, key) == 0) {

            if (allocate_digits(dst, mul_cache[i].result.len) != TERNARY_NO_ERROR) return TERNARY_ERR_MEMALLOC;

            dst->len = mul_cache[i].result.len;

            dst->sign = mul_cache[i].result.sign;

            memcpy(dst->digits, mul_cache[i].result.digits, dst->len);

            return 0;

        }

    }

    return 2;

}


static void mul_cache_store(const char* key, const T81BigInt *val) {

    int slot = -1;

    for (int i = 0; i < MUL_CACHE_SIZE; i++) {

        if (!mul_cache[i].used) { slot = i; break; }

    }

    if (slot < 0) slot = 0;

    strncpy(mul_cache[slot].key, key, sizeof(mul_cache[slot].key));

    mul_cache[slot].key[sizeof(mul_cache[slot].key)-1] = '\0';

    t81bigint_free(&mul_cache[slot].result);

    mul_cache[slot].used = 1;

    allocate_digits(&mul_cache[slot].result, val->len);
```

```
2316.
2317.    mul_cache[slot].result.len = val->len;
2318.
2319.    mul_cache[slot].result.sign = val->sign;
2320.
2321.    memcpy(mul_cache[slot].result.digits, val->digits, val->len);
2322.
2323.}
2324.
2325.
2326.static TernaryError multiply_with_cache(const T81BigInt *a, const T81BigInt *b,
    T81BigInt *out) {
2327.
2328.    char *as = NULL, *bs = NULL;
2329.
2330.    if (t81bigint_to_trit_string(a, &as) != TERNARY_NO_ERROR) return
    TERNARY_ERR_INVALID_INPUT;
2331.
2332.    if (t81bigint_to_trit_string(b, &bs) != TERNARY_NO_ERROR) { free(as); return
    TERNARY_ERR_INVALID_INPUT; }
2333.
2334.    char key[128];
2335.
2336.    snprintf(key, sizeof(key), "mul:%s:%s", as, bs);
2337.
2338.    free(as); free(bs);
2339.
2340.    if (mul_cache_lookup(key, out) == 0) return TERNARY_NO_ERROR;
2341.
2342.    TernaryError e = t81bigint_karatsuba_multiply(a, b, out);
2343.
2344.    if (e == TERNARY_NO_ERROR) { mul_cache_store(key, out); }
2345.
2346.    return e;
2347.
2348.}
2349.
2350.
2351.TernaryError tritjs_multiply_big(T81BigInt* a, T81BigInt* b, T81BigInt** result) {
2352.
2353.    if (!a || !b) return TERNARY_ERR_INVALID_INPUT;
2354.
2355.    *result = (T81BigInt*)calloc(1, sizeof(T81BigInt));
2356.
2357.    if (!*result) return TERNARY_ERR_MEMALLOC;
2358.
2359.    TernaryError e = multiply_with_cache(a, b, *result);
2360.
2361.    if (e != TERNARY_NO_ERROR) { free(*result); *result = NULL; }
2362.
2363.    return e;
2364.
2365.}
2366.
```

```
2367.
2368. TernaryError tritjs_factorial_big(T81BigInt* a, T81BigInt** result) {
2369.
2370.     if (!a) return TERNARY_ERR_INVALID_INPUT;
2371.
2372.     if (a->sign) return TERNARY_ERR_NEGATIVE;
2373.
2374.     if (a->len != 1 || a->digits[0] >= BASE_81) return TERNARY_ERR_OVERFLOW;
2375.
2376.     int val = a->digits[0];
2377.
2378.     if (val > 20) return TERNARY_ERR_OVERFLOW;
2379.
2380.     long long f = 1;
2381.
2382.     for (int i = 1; i <= val; i++) f *= i;
2383.
2384.     *result = (T81BigInt*)calloc(1, sizeof(T81BigInt));
2385.
2386.     if (!*result) return TERNARY_ERR_MEMALLOC;
2387.
2388.     if (allocate_digits(*result, 1) != TERNARY_NO_ERROR) { free(*result); return
    TERNARY_ERR_MEMALLOC; }
2389.
2390.     (*result)->digits[0] = 0; (*result)->sign = 0;
2391.
2392.     while (f > 0) {
2393.
2394.         int digit = f % BASE_81;
2395.
2396.         f /= BASE_81;
2397.
2398.         int carry = digit;
2399.
2400.         size_t i = 0;
2401.
2402.         while (carry) {
2403.
2404.             int val2 = (*result)->digits[i] + carry;
2405.
2406.             (*result)->digits[i] = val2 % BASE_81;
2407.
2408.             carry = val2 / BASE_81;
2409.
2410.             i++;
2411.
2412.             if (i >= (*result)->len && carry) {
2413.
2414.                 if (allocate_digits(*result, (*result)->len + 1) != TERNARY_NO_ERROR) {
2415.
2416.                     tritbig_free(*result); *result = NULL; return TERNARY_ERR_MEMALLOC;
2417.
2418.                 }
2419.
```

```
            }

        }

    }

    while ((*result)->len > 1 && (*result)->digits[(*result)->len - 1] == 0)

        (*result)->len--;

    return TERNARY_NO_ERROR;

}


TernaryError tritjs_power_big(T81BigInt* base, T81BigInt* exp, T81BigInt** result) {

    if (!base || !exp) return TERNARY_ERR_INVALID_INPUT;

    if (exp->sign) return TERNARY_ERR_NEGATIVE;

    if (exp->len != 1 || exp->digits[0] >= BASE_81) return TERNARY_ERR_OVERFLOW;

    int e = exp->digits[0];

    if (e > 1000) return TERNARY_ERR_OVERFLOW;

    *result = (T81BigInt*)calloc(1, sizeof(T81BigInt));

    if (!*result) return TERNARY_ERR_MEMALLOC;

    if (allocate_digits(*result, 1) != TERNARY_NO_ERROR) { free(*result); return
TERNARY_ERR_MEMALLOC; }

    (*result)->digits[0] = 1; (*result)->sign = 0;

    for (int i = 0; i < e; i++) {

        T81BigInt *tmp = NULL;

        TernaryError err = multiply_with_cache(*result, base, *result);

        if (err != TERNARY_NO_ERROR) { tritbig_free(*result); free(*result); *result = NULL;
return err; }

    }

    if (base->sign && (e % 2) == 1)

        (*result)->sign = 1;

    return TERNARY_NO_ERROR;

}
```

```
2472.
2473.
2474. /* Scientific functions via double conversion */
2475.
2476. static double t81bigint_to_double(T81BigInt* x) {
2477.
2478.     int sign = x->sign ? -1 : 1;
2479.
2480.     double accum = 0.0;
2481.
2482.     for (ssize_t i = x->len - 1; i >= 0; i--) {
2483.
2484.         accum = accum * BASE_81 + x->digits[i];
2485.
2486.     }
2487.
2488.     return sign * accum;
2489.
2490. }
2491.
2492.
2493. static T81BigInt* double_to_t81bigint(double d) {
2494.
2495.     T81BigInt* result = (T81BigInt*)calloc(1, sizeof(T81BigInt));
2496.
2497.     if (!result) return NULL;
2498.
2499.     int sign = (d < 0) ? 1 : 0;
2500.
2501.     if (d < 0) d = -d;
2502.
2503.     size_t capacity = 16;
2504.
2505.     result->digits = (unsigned char*)calloc(capacity, 1);
2506.
2507.     result->len = 0;
2508.
2509.     while (d >= 1.0) {
2510.
2511.         int digit = (int)fmod(d, BASE_81);
2512.
2513.         if (result->len >= capacity) { capacity *= 2; result->digits = realloc(result->digits,
    capacity); }
2514.
2515.         result->digits[result->len++] = (unsigned char) digit;
2516.
2517.         d = floor(d / BASE_81);
2518.
2519.     }
2520.
2521.     if (result->len == 0) { result->digits[0] = 0; result->len = 1; }
2522.
2523.     result->sign = sign;
2524.
```

```
    return result;
}

TernaryError tritjs_sqrt_complex(T81BigInt* a, int precision, T81Complex* result) {
    (void) precision;
    double d = t81bigint_to_double(a);
    double sq = sqrt(d);
    T81BigInt* res = double_to_t81bigint(sq);
    result->real = *res;
    free(res);
    result->imag.digits = calloc(1, 1);
    result->imag.len = 1;
    result->imag.sign = 0;
    return TERNARY_NO_ERROR;
}

TernaryError tritjs_log3_complex(T81BigInt* a, int precision, T81Complex* result) {
    (void) precision;
    double d = t81bigint_to_double(a);
    double l = log(d) / log(3);
    T81BigInt* res = double_to_t81bigint(l);
    result->real = *res;
    free(res);
    result->imag.digits = calloc(1, 1);
    result->imag.len = 1;
    result->imag.sign = 0;
    return TERNARY_NO_ERROR;
}
```

```
2579.
2580. TernaryError tritjs_sin_complex(T81BigInt* a, int precision, T81Complex* result) {
2581.
2582.     (void) precision;
2583.
2584.     double d = t81bigint_to_double(a);
2585.
2586.     double s = sin(d);
2587.
2588.     T81BigInt* res = double_to_t81bigint(s);
2589.
2590.     result->real = *res;
2591.
2592.     free(res);
2593.
2594.     result->imag.digits = calloc(1, 1);
2595.
2596.     result->imag.len = 1;
2597.
2598.     result->imag.sign = 0;
2599.
2600.     return TERNARY_NO_ERROR;
2601.
2602. }
2603.
2604.
2605. TernaryError tritjs_cos_complex(T81BigInt* a, int precision, T81Complex* result) {
2606.
2607.     (void) precision;
2608.
2609.     double d = t81bigint_to_double(a);
2610.
2611.     double c = cos(d);
2612.
2613.     T81BigInt* res = double_to_t81bigint(c);
2614.
2615.     result->real = *res;
2616.
2617.     free(res);
2618.
2619.     result->imag.digits = calloc(1, 1);
2620.
2621.     result->imag.len = 1;
2622.
2623.     result->imag.sign = 0;
2624.
2625.     return TERNARY_NO_ERROR;
2626.
2627. }
2628.
2629.
2630. TernaryError tritjs_tan_complex(T81BigInt* a, int precision, T81Complex* result) {
2631.
2632.     (void) precision;
```

```
2633.
2634.    double d = t81bigint_to_double(a);
2635.
2636.    double t = tan(d);
2637.
2638.    T81BigInt* res = double_to_t81bigint(t);
2639.
2640.    result->real = *res;
2641.
2642.    free(res);
2643.
2644.    result->imag.digits = calloc(1, 1);
2645.
2646.    result->imag.len = 1;
2647.
2648.    result->imag.sign = 0;
2649.
2650.    return TERNARY_NO_ERROR;
2651.
2652.}
2653.
2654.
2655.TernaryError tritjs_pi(int* len, int** pi) {
2656.
2657.    static int pi_val[] = {1, 0, 0, 1, 0, 2, 2, 1};
2658.
2659.    *len = 8;
2660.
2661.    *pi = malloc(8 * sizeof(int));
2662.
2663.    if (!*pi) return TERNARY_ERR_MEMALLOC;
2664.
2665.    memcpy(*pi, pi_val, 8 * sizeof(int));
2666.
2667.    return TERNARY_NO_ERROR;
2668.
2669.}
2670.
2671.
2672./* Division using a simplified long division (via binary conversion) */
2673.
2674.TernaryError tritjs_divide_big(T81BigInt* a, T81BigInt* b, T81BigInt** quotient,
    T81BigInt** remainder) {
2675.
2676.    if (!a || !b) return TERNARY_ERR_INVALID_INPUT;
2677.
2678.    int b_zero = 1;
2679.
2680.    for (size_t i = 0; i < b->len; i++) {
2681.
2682.        if (b->digits[i] != 0) { b_zero = 0; break; }
2683.
2684.    }
2685.
```

```
    if (b_zero) { log_error(TERNARY_ERR_DIVZERO, "tritjs_divide_big", __FILE__, __LINE__); return TERNARY_ERR_DIVZERO; }

    *quotient = (T81BigInt*)calloc(1, sizeof(T81BigInt));

    *remainder = (T81BigInt*)calloc(1, sizeof(T81BigInt));

    if (!*quotient || !*remainder) return TERNARY_ERR_MEMALLOC;

    if (allocate_digits(*remainder, a->len) != TERNARY_NO_ERROR) return TERNARY_ERR_MEMALLOC;

    memcpy((*remainder)->digits, a->digits, a->len);

    (*remainder)->len = a->len;

    if (allocate_digits(*quotient, a->len) != TERNARY_NO_ERROR) return TERNARY_ERR_MEMALLOC;

    memset((*quotient)->digits, 0, a->len);

    int rem_int, b_int;

    if (trit_to_binary(*remainder, &rem_int) != TERNARY_NO_ERROR) return TERNARY_ERR_INVALID_INPUT;

    if (trit_to_binary(b, &b_int) != TERNARY_NO_ERROR) return TERNARY_ERR_INVALID_INPUT;

    int q = rem_int / b_int;

    int r = rem_int % b_int;

    T81BigInt *q_big = NULL, *r_big = NULL;

    if (binary_to_trit(q, &q_big) != TERNARY_NO_ERROR) return TERNARY_ERR_INVALID_INPUT;

    if (binary_to_trit(r, &r_big) != TERNARY_NO_ERROR) return TERNARY_ERR_INVALID_INPUT;

    *quotient = q_big;

    *remainder = r_big;

    return TERNARY_NO_ERROR;

}


/* Main function: ncurses–based UI for command processing */

int main(int argc, char** argv) {

```

```
    (void)argc; (void)argv;

    init_audit_log();

    initscr();

    cbreak();

    noecho();

    keypad(stdscr, TRUE);

    printw("TritJS-CISA-Optimized Version %s\n", VERSION);

    printw("Enter commands (type 'help' for assistance, 'exit' to quit):\n");

    refresh();

    char command[256];

    while (1) {

        printw("> ");

        getnstr(command, sizeof(command)-1);

        if (strcmp(command, "exit") == 0) break;

        if (strcmp(command, "help") == 0) {

            printw("\nAvailable Commands:\n");

            printw("  help   : Display this help message.\n");

            printw("  exit   : Quit the utility.\n");

            printw("  [other commands would be implemented here]\n\n");

        } else {

            printw("Command received: %s\n", command);

            /* Command parsing and execution logic would be added here. */

        }

        refresh();

    }

    endwin();

    return 0;
```

```
}

@* End of User–Space Utility section. @*/

#endif /* !__KERNEL__ */


@*
==========================================================================

  End of Ternary System Literate Program

  This file integrates the Axion kernel module and the TritJS-CISA–Optimized

  user–space utility into a single unified source file. It provides comprehensive

  documentation, a built–in help system, and shared ternary logic definitions to

  further our mission to transition from binary to native ternary computing.

========================================================================== */


```

Special thanks to xAI and OpenAI
for making this possible.


CISA
CISA
CISA