Q1:

Code changes in xv6-public directory:

Step 1:
In proc.h added:
```
int twostrike_mode;
int strike_count;
```

Step 2:
Inside allocproc() in proc.c added:
```
p->twostrike_mode = 0;
p->strike_count = 0;
```

Step 3:
Inside syscall.h added:
```
#define SYS_twostrike 23
```

Step 4:
In syscall.c added:
```
extern int sys_twostrike(void);
```

Inside syscalls[] array added:
```
[SYS_twostrike]    sys_twostrike,
```

Step 5:
In sysproc.c added:
```
int
sys_twostrike(void)
{
    int enabled;
    if(argint(0, &enabled) < 0)
        return -1;

    struct proc *p = myproc();
    p->twostrike_mode = enabled;
    p->strike_count = 0;  // reset when enabled

    return 0;
}
```

Step 6:
Inside user.h:
```
int twostrike(int enabled);
```

Step 7:
Inside usys.S added:
```
SYSCALL(twostrike)
```

Step 8:
In console.c in consoleintr() added:
```
void
```

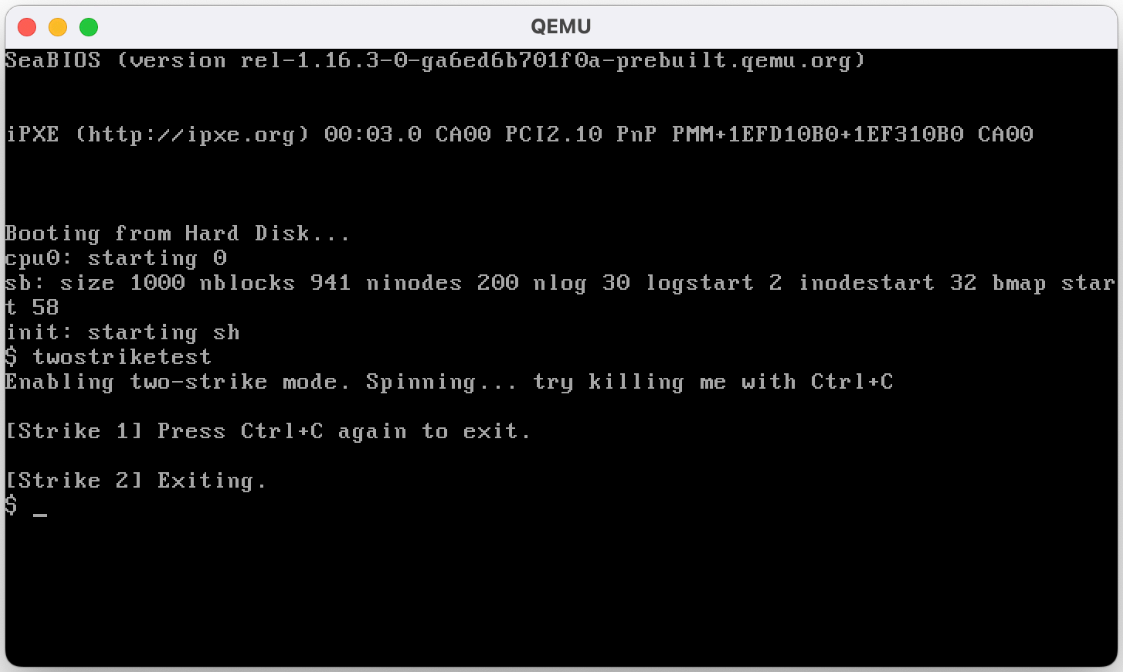
```
consoleintr(int (*getc)(void))
{
  int c, doprocdump = 0;
  int do_twostrike = 0;

  acquire(&cons.lock);

  while((c = getc()) >= 0){
    switch(c){

    // -------------------------
    // Two-Strike Ctrl+C handler
    // -------------------------
    case 3:   // ASCII ETX = Ctrl+C
      do_twostrike = 1;
      break;

    // Existing system key: Ctrl+P
    case C('P'):
      doprocdump = 1;
      break;

    // Kill line: Ctrl+U
    case C('U'):
      while(input.e != input.w &&
            input.buf[(input.e-1) % INPUT_BUF] != '\n'){
        input.e--;
        consputc(BACKSPACE);
      }
      break;

    // Backspace
    case C('H'):
    case '\x7f':
      if(input.e != input.w){
        input.e--;
        consputc(BACKSPACE);
      }
      break;

    // Normal character handling
    default:
      if(c != 0 && input.e-input.r < INPUT_BUF){
        c = (c == '\r') ? '\n' : c;
        input.buf[input.e++ % INPUT_BUF] = c;
        consputc(c);

        if(c == '\n' || c == C('D') || input.e == input.r+INPUT_BUF){
          input.w = input.e;
          wakeup(&input.r);
        }
      }
      break;
    }
  }

  release(&cons.lock);

  // Process dump triggered
  if(doprocdump)
    procdump();

  // ----------------------------------------------------
  // Two-Strike Logic (AFTER releasing console lock)
  // ----------------------------------------------------
  if(do_twostrike){
    struct proc *p = myproc();

    if(p != 0 && p->state == RUNNING){

      if(p->twostrike_mode == 1){

        // First strike
        if(p->strike_count == 0){
          p->strike_count = 1;
          cprintf("\n[Strike 1] Press Ctrl+C again to exit.\n");
        }
        // Second strike
        else {
          cprintf("\n[Strike 2] Exiting.\n");
          p->killed = 1;
        }

      } else {
        // Two-strike mode disabled: normal kill
        p->killed = 1;
      }
    }
  }
}
```

Step 9:
Created twostriketest.c with this:

```
#include "types.h"
#include "stat.h"
#include "user.h"

int
main(int argc, char *argv[])
{
    printf(1, "Enabling two-strike mode. Spinning... try killing me with Ctrl+C\n");

    twostrike(1);  // BLANK 6
    while(1) {     // BLANK 7
        // spin
    }

    exit();
}
```

Step 10:
Inside UPROGS in Makefile added:
_twostriketest\

Screenshot of the output:



QEMU

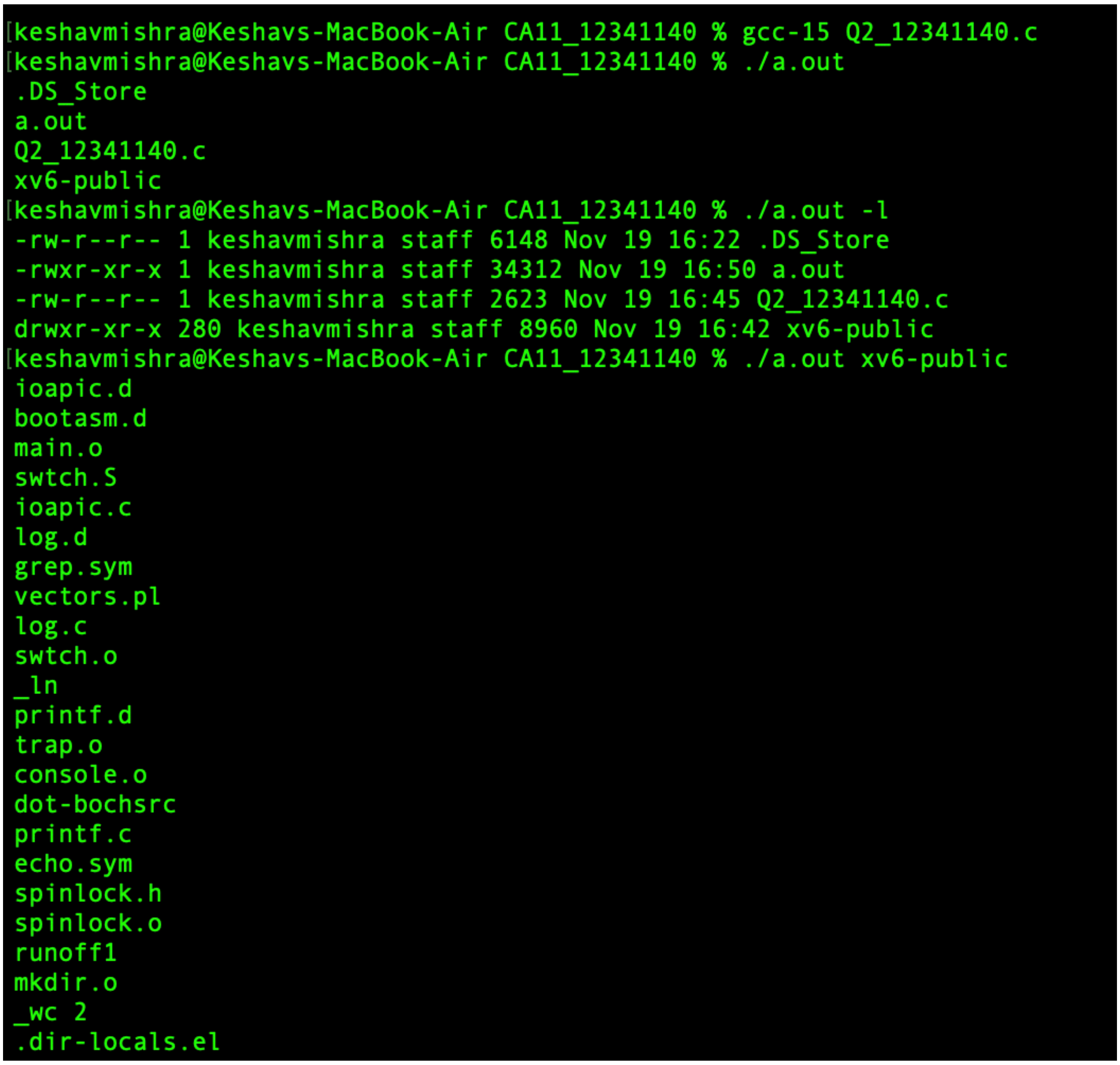
```
SeaBIOS (version rel-1.16.3-0-ga6ed6b701f0a-prebuilt.qemu.org)


iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1EFD10B0+1EF310B0 CA00



Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ twostriketest
Enabling two-strike mode. Spinning... try killing me with Ctrl+C

[Strike 1] Press Ctrl+C again to exit.

[Strike 2] Exiting.
$ _
```

Explanation:

This question implements a safer process termination mechanism in xv6 where pressing Ctrl+C once only gives a warning, and the process is killed only on the second press. It requires adding a new system call, tracking strike counts in the process table, and modifying console input handling.

Q2:
Code:
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <pwd.h>
#include <grp.h>
#include <unistd.h>
#include <time.h>

void print_permissions(mode_t mode) {
    printf( (S_ISDIR(mode)) ? "d" : "-");
    printf( (mode & S_IRUSR) ? "r" : "-");
    printf( (mode & S_IWUSR) ? "w" : "-");
    printf( (mode & S_IXUSR) ? "x" : "-");
    printf( (mode & S_IRGRP) ? "r" : "-");
    printf( (mode & S_IWGRP) ? "w" : "-");
    printf( (mode & S_IXGRP) ? "x" : "-");
    printf( (mode & S_IROTH) ? "r" : "-");
    printf( (mode & S_IWOTH) ? "w" : "-");
    printf( (mode & S_IXOTH) ? "x" : "-");
}

void print_file_details(const char *path, const char *name) {
    char fullpath[1024];
    snprintf(fullpath, sizeof(fullpath), "%s/%s", path, name);

    struct stat sb;
    if (stat(fullpath, &sb) == -1) {
        perror("stat");
        return;
    }

    // 1. Permissions
    print_permissions(sb.st_mode);
    printf(" ");

    // 2. Number of hard links
    printf("%ld ", (long)sb.st_nlink);

    // 3. User name
    struct passwd *pw = getpwuid(sb.st_uid);
    printf("%s ", pw ? pw->pw_name : "unknown");

    // 4. Group name
    struct group *gr = getgrgid(sb.st_gid);
    printf("%s ", gr ? gr->gr_name : "unknown");

    // 5. File size
    printf("%ld ", (long)sb.st_size);

    // 6. Modification time (format: "Nov 10 11:36")
    char timebuf[64];
    struct tm *tm = localtime(&sb.st_mtime);
    strftime(timebuf, sizeof(timebuf), "%b %d %H:%M", tm);
    printf("%s ", timebuf);

    // 7. File name
    printf("%s\n", name);
}

int main(int argc, char *argv[]) {
    int long_format = 0;
    char *dirpath = NULL;

    if (argc == 1) {
        dirpath = ".";
    } else if (argc == 2) {
        if (strcmp(argv[1], "-l") == 0) {
            long_format = 1;
            dirpath = ".";
        } else {
            dirpath = argv[1];
        }
    } else if (argc == 3 && strcmp(argv[1], "-l") == 0) {
        long_format = 1;
        dirpath = argv[2];
    } else {
        fprintf(stderr, "Usage: %s [-l] [directory]\n", argv[0]);
        exit(1);
    }

    DIR *dir = opendir(dirpath);
    if (!dir) {
        perror("opendir");
        return 1;
    }

    struct dirent *entry;
    while ((entry = readdir(dir)) != NULL) {

        // skip . and ..
        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") == 0)
            continue;

        if (long_format)
            print_file_details(dirpath, entry->d_name);
        else
            printf("%s\n", entry->d_name);
    }
    closedir(dir);
    return 0;
}
```

Screenshot of the output:



```
[keshavmishra@Keshavs-MacBook-Air CA11_12341140 % gcc-15 Q2_12341140.c
[keshavmishra@Keshavs-MacBook-Air CA11_12341140 % ./a.out
.DS_Store
a.out
Q2_12341140.c
xv6-public
[keshavmishra@Keshavs-MacBook-Air CA11_12341140 % ./a.out -l
-rw-r--r-- 1 keshavmishra staff 6148 Nov 19 16:22 .DS_Store
-rwxr-xr-x 1 keshavmishra staff 34312 Nov 19 16:50 a.out
-rw-r--r-- 1 keshavmishra staff 2623 Nov 19 16:45 Q2_12341140.c
drwxr-xr-x 280 keshavmishra staff 8960 Nov 19 16:42 xv6-public
[keshavmishra@Keshavs-MacBook-Air CA11_12341140 % ./a.out xv6-public
ioapic.d
bootasm.d
main.o
swtch.S
ioapic.c
log.d
grep.sym
vectors.pl
log.c
swtch.o
_ln
printf.d
trap.o
console.o
dot-bochsrc
printf.c
echo.sym
spinlock.h
spinlock.o
runoff1
mkdir.o
_wc 2
.dir-locals.el
```

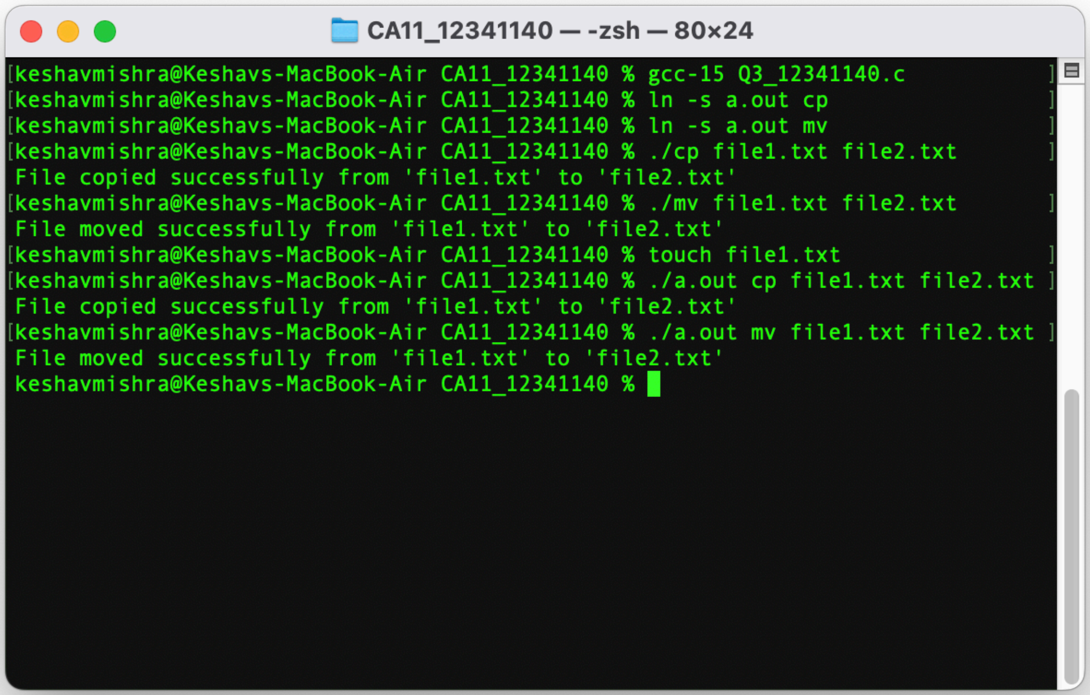
Explanation:
This question implements a simplified version of the Linux ls command that lists files in a directory and optionally shows detailed metadata when the -l flag is used. It uses directory traversal with readdir and stat to print permissions, owner, group, size, and timestamps.

Q3:
Code:
```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
#include <string.h>
#include <errno.h>

#define BUF_SIZE 4096

int copy_file(const char *src, const char *dest) {
    int src_fd, dest_fd;
    struct stat src_stat;
    char buffer[BUF_SIZE];
    ssize_t bytes_read, bytes_written;

    src_fd = open(src, O_RDONLY);
    if (src_fd < 0) { perror("open source"); return -1; }

    if (stat(src, &src_stat) < 0) {
        perror("stat");
        close(src_fd);
        return -1;
    }

    // Create dest with SAME permissions as src
    dest_fd = open(dest, O_WRONLY | O_CREAT | O_TRUNC, src_stat.st_mode);
    if (dest_fd < 0) {
        perror("open destination");
        close(src_fd);
        return -1;
    }

    // --- COPY LOOP ---
    while ((bytes_read = read(src_fd, buffer, BUF_SIZE)) > 0) {
        bytes_written = write(dest_fd, buffer, bytes_read);
        if (bytes_written < 0) {
            perror("write");
            close(src_fd);
            close(dest_fd);
            return -1;
        }
    }

    if (bytes_read < 0) perror("read");

    close(src_fd);
    close(dest_fd);
    return 0;
}
```

```c
int move_file(const char *src, const char *dest) {
    // First try rename (fast move)
    if (rename(src, dest) == 0) {
        return 0;
    }

    // If rename fails (cross-filesystem), fallback to copy + unlink
    if (copy_file(src, dest) < 0) return -1;

    if (unlink(src) < 0) {
        perror("unlink source");
        return -1;
    }

    return 0;
}

int main(int argc, char *argv[]) {
    const char *prog;
    const char *src;
    const char *dest;

    if (argc == 3) {
        // ./cp src dest  OR ./mv src dest
        prog = strrchr(argv[0], '/');
        prog = (prog == NULL) ? argv[0] : prog + 1;

        src = argv[1];
        dest = argv[2];
    }
    else if (argc == 4) {
        // ./a.out cp src dest  OR ./a.out mv src dest
        prog = argv[1];
        src = argv[2];
        dest = argv[3];
    }
    else {
        fprintf(stderr, "Usage:\n");
        fprintf(stderr, "  %s <source> <destination>\n", argv[0]);
        fprintf(stderr, "  %s <cp|mv> <source> <destination>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    if (strcmp(prog, "cp") == 0) {
        if (copy_file(src, dest) == 0)
            printf("File copied successfully from '%s' to '%s'\n", src, dest);
        else
            fprintf(stderr, "Copy failed.\n");
    }
    else if (strcmp(prog, "mv") == 0) {
        if (move_file(src, dest) == 0)
```

```
            printf("File moved successfully from '%s' to '%s'\n", src, dest);
        else
            fprintf(stderr, "Move failed.\n");
    }
    else {
        fprintf(stderr, "Error: Invalid command name '%s'. Use cp or mv.\n", prog);
        exit(EXIT_FAILURE);
    }

    return 0;
}
```

Screenshot of the output:



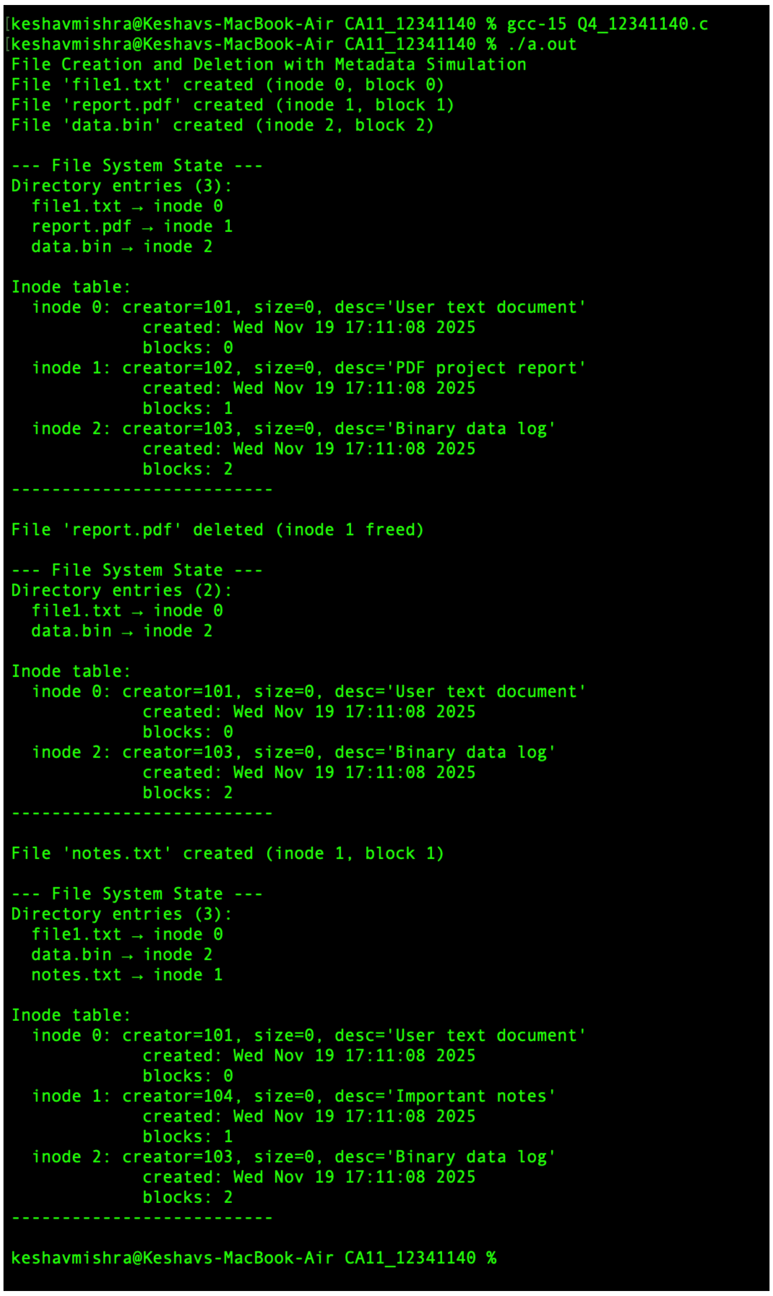
```
[keshavmishra@Keshavs-MacBook-Air CA11_12341140 % gcc-15 Q3_12341140.c
[keshavmishra@Keshavs-MacBook-Air CA11_12341140 % ln -s a.out cp
[keshavmishra@Keshavs-MacBook-Air CA11_12341140 % ln -s a.out mv
[keshavmishra@Keshavs-MacBook-Air CA11_12341140 % ./cp file1.txt file2.txt
File copied successfully from 'file1.txt' to 'file2.txt'
[keshavmishra@Keshavs-MacBook-Air CA11_12341140 % ./mv file1.txt file2.txt
File moved successfully from 'file1.txt' to 'file2.txt'
[keshavmishra@Keshavs-MacBook-Air CA11_12341140 % touch file1.txt
[keshavmishra@Keshavs-MacBook-Air CA11_12341140 % ./a.out cp file1.txt file2.txt
File copied successfully from 'file1.txt' to 'file2.txt'
[keshavmishra@Keshavs-MacBook-Air CA11_12341140 % ./a.out mv file1.txt file2.txt
File moved successfully from 'file1.txt' to 'file2.txt'
keshavmishra@Keshavs-MacBook-Air CA11_12341140 %
```

Explanation:
This question creates a combined cp/mv utility that behaves as either copy or move based on how it is invoked. It uses low-level system calls like open, read, write, rename, and unlink to duplicate or relocate files while preserving permissions.

Q4:
Code:
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define MAX_INODES 8
#define MAX_BLOCKS 64
#define MAX_FILENAME 32
#define MAX_DESC 64
#define DIRECT_PTRS 4
#define MAX_DIR_ENTRIES 16

struct inode {
    int used;                   // 0 = free, 1 = allocated
    int size;                   // file size in bytes
    int type;                   // 0 = file, 1 = directory
    int creator_id;             // user ID who created file
    time_t created_at;          // creation timestamp
    char description[MAX_DESC];    // short file description
    int direct[DIRECT_PTRS];       // direct data block pointers
};

struct dir_entry {
    char name[MAX_FILENAME];
    int inum;
};

struct inode inode_table[MAX_INODES];
int data_block_bitmap[MAX_BLOCKS];   // 0 = free, 1 = used
struct dir_entry directory[MAX_DIR_ENTRIES];
int dir_count = 0;

int alloc_inode() {
    for (int i = 0; i < MAX_INODES; i++) {
        if (!inode_table[i].used) {
            inode_table[i].used = 1;
            inode_table[i].size = 0;
            inode_table[i].type = 0;
            inode_table[i].creator_id = 0;
            inode_table[i].created_at = time(NULL);
            strcpy(inode_table[i].description, "No description");
            for (int j = 0; j < DIRECT_PTRS; j++)
                inode_table[i].direct[j] = -1;
            return i;
        }
    }
    return -1;
}

void free_inode(int inum) {
    inode_table[inum].used = 0;
    inode_table[inum].size = 0;
    inode_table[inum].creator_id = 0;
    inode_table[inum].created_at = 0;
    strcpy(inode_table[inum].description, "");
    for (int j = 0; j < DIRECT_PTRS; j++)
        inode_table[inum].direct[j] = -1;
}

int alloc_block() {
    for (int i = 0; i < MAX_BLOCKS; i++) {
        if (data_block_bitmap[i] == 0) {
            data_block_bitmap[i] = 1;
            return i;
        }
    }
    return -1;
}

void free_block(int blocknum) {
    if (blocknum >= 0 && blocknum < MAX_BLOCKS)
        data_block_bitmap[blocknum] = 0;
}

int fs_create(const char *name, int creator_id, const char *desc) {
    // check duplicate name
    for (int i = 0; i < dir_count; i++) {
        if (strcmp(directory[i].name, name) == 0) {
            printf("Error: File '%s' already exists\n", name);
            return -1;
        }
    }

    int inum = alloc_inode();
    if (inum < 0) {
        printf("Error: No free inode available\n");
        return -1;
    }

    // ------------------------------
    // ADD YOUR CODE:
    // Fill metadata
    // ------------------------------
    inode_table[inum].creator_id = creator_id;
    strncpy(inode_table[inum].description, desc, MAX_DESC - 1);
    inode_table[inum].description[MAX_DESC - 1] = '\0';

    int block = alloc_block();
    if (block < 0) {
        printf("Error: No free data block available\n");
        free_inode(inum);
        return -1;
    }

    inode_table[inum].direct[0] = block;
    inode_table[inum].size = 0;

    strcpy(directory[dir_count].name, name);
    directory[dir_count].inum = inum;
    dir_count++;

    printf("File '%s' created (inode %d, block %d)\n", name, inum, block);
    return inum;
}

int fs_delete(const char *name) {
    int found = -1;

    for (int i = 0; i < dir_count; i++) {
        if (strcmp(directory[i].name, name) == 0) {
            found = i;
            break;
        }
    }

    if (found < 0) {
        printf("Error: File '%s' not found\n", name);
        return -1;
    }

    int inum = directory[found].inum;

    // --------------------------------
    // ADD YOUR CODE:
    // Free all direct blocks
    // --------------------------------
    for (int j = 0; j < DIRECT_PTRS; j++) {
        if (inode_table[inum].direct[j] != -1) {
            free_block(inode_table[inum].direct[j]);
            inode_table[inum].direct[j] = -1;
        }
    }

    free_inode(inum);

    // remove from directory list
    for (int i = found; i < dir_count - 1; i++)
        directory[i] = directory[i + 1];

    dir_count--;

    printf("File '%s' deleted (inode %d freed)\n", name, inum);
    return 0;
}
```

```
void fs_show_state() {
    printf("\n--- File System State ---\n");
    printf("Directory entries (%d):\n", dir_count);
    for (int i = 0; i < dir_count; i++)
        printf("  %s → inode %d\n", directory[i].name, directory[i].inum);

    printf("\nInode table:\n");
    for (int i = 0; i < MAX_INODES; i++) {
        if (inode_table[i].used) {
            printf("  inode %d: creator=%d, size=%d, desc='%s'\n",
                   i, inode_table[i].creator_id, inode_table[i].size, inode_table[i].description);
            printf("            created: %s", ctime(&inode_table[i].created_at));
            printf("            blocks: ");
            for (int j = 0; j < DIRECT_PTRS; j++)
                if (inode_table[i].direct[j] != -1)
                    printf("%d ", inode_table[i].direct[j]);
            printf("\n");
        }
    }
    printf("--------------------------\n\n");
}

int main() {
    printf("File Creation and Deletion with Metadata Simulation\n");

    fs_create("file1.txt", 101, "User text document");
    fs_create("report.pdf", 102, "PDF project report");
    fs_create("data.bin", 103, "Binary data log");
    fs_show_state();

    fs_delete("report.pdf");
    fs_show_state();

    fs_create("notes.txt", 104, "Important notes");
    fs_show_state();

    return 0;
}
```

Screenshot of the output:



```
[keshavmishra@Keshavs-MacBook-Air CA11_12341140 % gcc-15 Q4_12341140.c
[keshavmishra@Keshavs-MacBook-Air CA11_12341140 % ./a.out
File Creation and Deletion with Metadata Simulation
File 'file1.txt' created (inode 0, block 0)
File 'report.pdf' created (inode 1, block 1)
File 'data.bin' created (inode 2, block 2)

--- File System State ---
Directory entries (3):
  file1.txt → inode 0
  report.pdf → inode 1
  data.bin → inode 2

Inode table:
  inode 0: creator=101, size=0, desc='User text document'
             created: Wed Nov 19 17:11:08 2025
             blocks: 0
  inode 1: creator=102, size=0, desc='PDF project report'
             created: Wed Nov 19 17:11:08 2025
             blocks: 1
  inode 2: creator=103, size=0, desc='Binary data log'
             created: Wed Nov 19 17:11:08 2025
             blocks: 2
--------------------------

File 'report.pdf' deleted (inode 1 freed)

--- File System State ---
Directory entries (2):
  file1.txt → inode 0
  data.bin → inode 2

Inode table:
  inode 0: creator=101, size=0, desc='User text document'
             created: Wed Nov 19 17:11:08 2025
             blocks: 0
  inode 2: creator=103, size=0, desc='Binary data log'
             created: Wed Nov 19 17:11:08 2025
             blocks: 2
--------------------------

File 'notes.txt' created (inode 1, block 1)

--- File System State ---
Directory entries (3):
  file1.txt → inode 0
  data.bin → inode 2
  notes.txt → inode 1

Inode table:
  inode 0: creator=101, size=0, desc='User text document'
             created: Wed Nov 19 17:11:08 2025
             blocks: 0
  inode 1: creator=104, size=0, desc='Important notes'
             created: Wed Nov 19 17:11:08 2025
             blocks: 1
  inode 2: creator=103, size=0, desc='Binary data log'
             created: Wed Nov 19 17:11:08 2025
             blocks: 2
--------------------------

keshavmishra@Keshavs-MacBook-Air CA11_12341140 %
```

Explanation:
This question simulates a tiny file system that supports creating and deleting files while maintaining metadata such as creator ID, creation time, and block allocation. It requires allocating inodes and data blocks on creation and freeing them properly on deletion while displaying the current file system state.