CSL301

12340220

Assignment: CA11

Question 1. The ”Double-Tap” Exit (Two-Strike Signal)

Proc.h (in proc struct)

```
 int twostrike_enabled;
 int strike_count;
```

sysproc.c

```
int
sys_twostrike(void)
{
 int enabled;
 if(argint(0, &enabled) < 0)
   return -1;
 myproc()->twostrike_enabled = enabled;
 if(enabled == 0)
   myproc()->strike_count = 0;
 return 0;
}
```

proc.c (in allocproc function)

```
 p->twostrike_enabled = 0;
 p->strike_count = 0;
```

syscall.c

```

extern int sys_twostrike(void);

static int (*syscalls[])(void) = {
..
[SYS_twostrike] sys_twostrike,
};
```

user.h

```
int twostrike(int enabled);
```

usys.S

```
SYSCALL(twostrike)
```

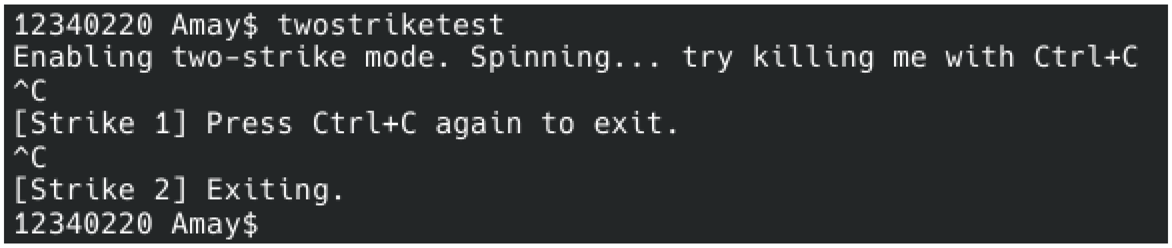
console.c

```
void
consoleintr(int (*getc)(void))
{
 int c, doprocdump = 0, do_twostrike = 0;
 acquire(&cons.lock);
 while((c = getc()) >= 0){
   switch(c){
   // BLANK 1 filled:
   case C('C'):
     do_twostrike = 1;
     break;

   case C('P'):
     doprocdump = 1;
     break;

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

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

   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);
 if(doprocdump) {
   procdump();
 }

 // --- Two Strike Logic ---
 if(do_twostrike) {
   struct proc *p = myproc();
   // BLANK 2 filled:
   if(p != 0 && p->state == RUNNING) {
     if(p->twostrike_enabled == 1) {
       // BLANK 3 filled:
       if(p->strike_count == 0) {
         // BLANK 4 filled:
         p->strike_count = 1;
         cprintf("\n[Strike 1] Press Ctrl+C again to exit.\n");
       } else {
         cprintf("\n[Strike 2] Exiting.\n");
         // BLANK 5 filled:
         p->killed = 1;
       }
     } else {
       p->killed = 1;
     }
   }
 }
}
```

twostriketest.c

```
#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
   // Busy loop
 }

 exit();
}
```

Output:



```
12340220 Amay$ twostriketest
Enabling two-strike mode. Spinning... try killing me with Ctrl+C
^C
[Strike 1] Press Ctrl+C again to exit.
^C
[Strike 2] Exiting.
12340220 Amay$
```

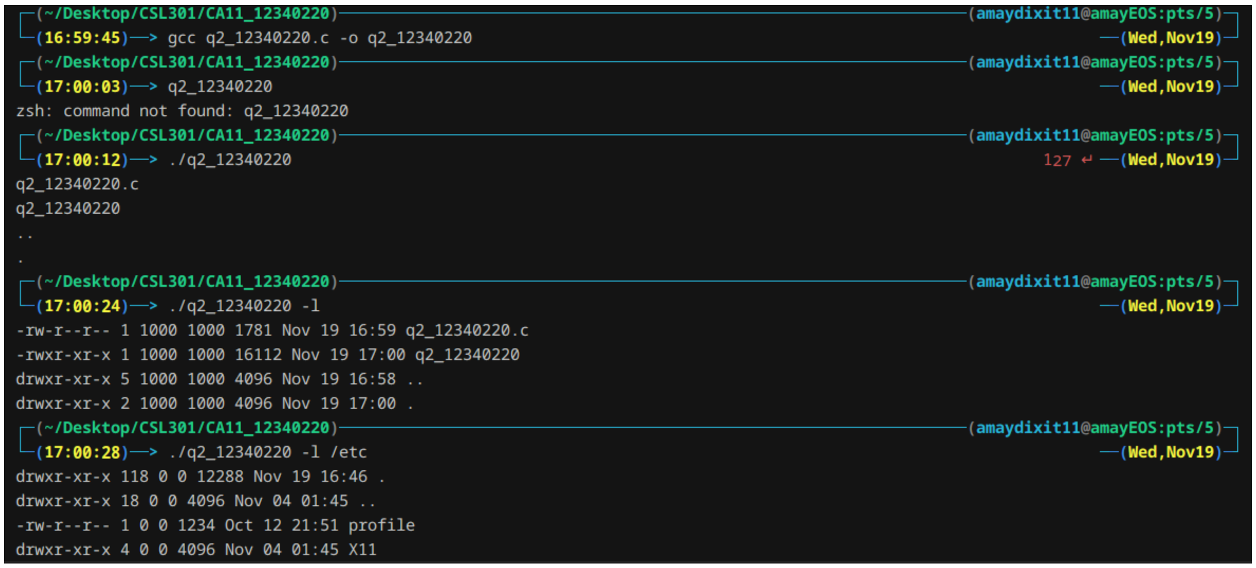
Explanation:

To implement the Double-Tap Exit (Two-Strike Signal) mechanism, two new fields (twostrike_enabled and strike_count) were added to the proc structure in proc.h and initialized in allocproc() to keep track of the signal mode and consecutive Ctrl+C presses. A new system call twostrike(int enabled) was implemented in sysproc.c and properly registered in syscall.c, user.h, and usys.S to allow user processes to enable the feature. The core logic was added inside consoleintr() in console.c, where pressing Ctrl+C first increments the strike counter and displays a warning instead of killing the process immediately. Only upon the second consecutive Ctrl+C press is the process marked as killed. A user-level test program (twostriketest.c) enables the feature and enters a spin loop, allowing interactive verification of the double-strike behavior. This enhances safe termination by preventing accidental process exits on the first interrupt signal.

Question 2. Implement a Simplified ls Command

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include <unistd.h>
#include <time.h>
#include <limits.h>

void print_permissions(mode_t m) {
    char type = S_ISDIR(m) ? 'd' : '-';
    printf("%c", type);
    printf((m & S_IRUSR) ? "r" : "-");
    printf((m & S_IWUSR) ? "w" : "-");
    printf((m & S_IXUSR) ? "x" : "-");
    printf((m & S_IRGRP) ? "r" : "-");
    printf((m & S_IWGRP) ? "w" : "-");
    printf((m & S_IXGRP) ? "x" : "-");
    printf((m & S_IROTH) ? "r" : "-");
    printf((m & S_IWOTH) ? "w" : "-");
    printf((m & S_IXOTH) ? "x" : "-");
}

void print_long(const char *path, const char *name) {
    char full[PATH_MAX];
    snprintf(full, sizeof(full), "%s/%s", path, name);

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

    print_permissions(st.st_mode);
    printf(" %ld %d %d %lld ",
        (long)st.st_nlink,
        st.st_uid, st.st_gid,
        (long long)st.st_size);

    char tbuf[64];
    struct tm *tm = localtime(&st.st_mtime);
    strftime(tbuf, sizeof(tbuf), "%b %d %H:%M", tm);

    printf("%s %s\n", tbuf, name);
}

int main(int argc, char *argv[]) {
    int long_flag = 0;
    char *path = ".";

    if (argc >= 2) {
        if (strcmp(argv[1], "-l") == 0) {
            long_flag = 1;
            if (argc >= 3) path = argv[2];
        } else {
            path = argv[1];
        }
    }

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

    struct dirent *entry;
    while ((entry = readdir(dir)) != NULL) {
        if (!long_flag)
            printf("%s\n", entry->d_name);
        else
            print_long(path, entry->d_name);
    }
    closedir(dir);
    return 0;
}
```

Output:



```
┌(~/Desktop/CSL301/CA11_12340220)──────────────────────────────(amaydixit11@amayEOS:pts/5)┐
└(16:59:45)──> gcc q2_12340220.c -o q2_12340220                                 ─(Wed,Nov19)┘
┌(~/Desktop/CSL301/CA11_12340220)──────────────────────────────(amaydixit11@amayEOS:pts/5)┐
└(17:00:03)──> q2_12340220                                                      ─(Wed,Nov19)┘
zsh: command not found: q2_12340220
┌(~/Desktop/CSL301/CA11_12340220)──────────────────────────────(amaydixit11@amayEOS:pts/5)┐
└(17:00:12)──> ./q2_12340220                                            127 ↵ ─(Wed,Nov19)┘
q2_12340220.c
q2_12340220
..
.
┌(~/Desktop/CSL301/CA11_12340220)──────────────────────────────(amaydixit11@amayEOS:pts/5)┐
└(17:00:24)──> ./q2_12340220 -l                                                 ─(Wed,Nov19)┘
-rw-r--r-- 1 1000 1000 1781 Nov 19 16:59 q2_12340220.c
-rwxr-xr-x 1 1000 1000 16112 Nov 19 17:00 q2_12340220
drwxr-xr-x 5 1000 1000 4096 Nov 19 16:58 ..
drwxr-xr-x 2 1000 1000 4096 Nov 19 17:00 .
┌(~/Desktop/CSL301/CA11_12340220)──────────────────────────────(amaydixit11@amayEOS:pts/5)┐
└(17:00:28)──> ./q2_12340220 -l /etc                                            ─(Wed,Nov19)┘
drwxr-xr-x 118 0 0 12288 Nov 19 16:46 .
drwxr-xr-x 18 0 0 4096 Nov 04 01:45 ..
-rw-r--r-- 1 0 0 1234 Oct 12 21:51 profile
drwxr-xr-x 4 0 0 4096 Nov 04 01:45 X11
```

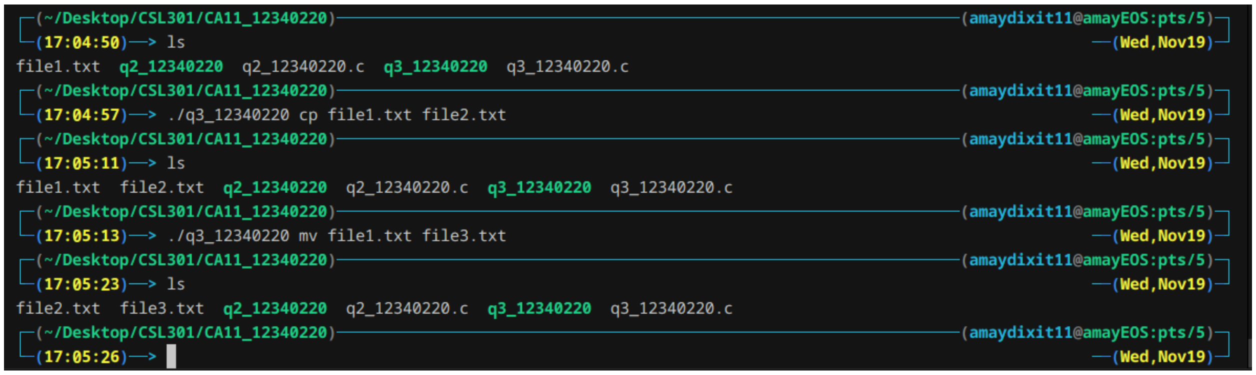
Explanation:

This program replicates the basic functionality of the Linux ls command using system directory access APIs. It lists all files inside a specified directory and supports an optional -l flag to show detailed metadata such as permissions, file size, owner IDs, number of links, and last modification time. The program uses opendir(), readdir(), and closedir() to access directory entries, while stat() retrieves each file's metadata. It formats and displays this information similar to the standard ls -l output. This assignment demonstrates working with file metadata and directory traversal in Unix-like systems.

Question 3. Combined Copy/Move File Utility

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

int copy_file(const char *src, const char *dst) {
    int in_fd, out_fd;
    struct stat st;

    if (stat(src, &st) == -1) {
        perror("stat");
        return -1;
    }

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

    out_fd = open(dst, O_WRONLY | O_CREAT | O_TRUNC, st.st_mode & 0777);
    if (out_fd < 0) {
        perror("open dst");
        close(in_fd);
        return -1;
    }

    char buf[4096];
    ssize_t n;
    while ((n = read(in_fd, buf, sizeof(buf))) > 0) {
        if (write(out_fd, buf, n) != n) {
            perror("write");
            close(in_fd);
            close(out_fd);
            return -1;
        }
    }

    close(in_fd);
    close(out_fd);
    return (n < 0) ? -1 : 0;
}

int do_mv(const char *src, const char *dst) {
    if (rename(src, dst) == 0) return 0;

    if (errno == EXDEV) {
        if (copy_file(src, dst) == 0) {
            if (unlink(src) == -1) perror("unlink");
            return 0;
        }
    }
    perror("mv");
    return -1;
}

char *basename(char *path) {
    char *b = strrchr(path, '/');
    return b ? b + 1 : path;
    }

    int main(int argc, char *argv[]) {
        if (argc < 3) {
            fprintf(stderr,"Usage: ./cp file1 file2  OR  ./mv file1 file2  OR  ./a.out cp
file1 file2\n");
            return 1;
        }

        char *cmd;
        int offset = 0;

        char *prog = basename(argv[0]);
        if (strcmp(prog, "cp") == 0 || strcmp(prog, "mv") == 0) {
            cmd = prog;
        } else {
            cmd = argv[1];
            offset = 1;
        }

        if (argc - offset != 3) {
            fprintf(stderr,"Invalid args.\n");
            return 1;
        }

        char *src = argv[1 + offset];
        char *dst = argv[2 + offset];

        if (strcmp(cmd, "cp") == 0) {
            return copy_file(src,dst);
        } else if (strcmp(cmd, "mv") == 0) {
            return do_mv(src,dst);
        } else {
            fprintf(stderr,"Unknown command.\n");
            return 1;
        }
    }
```

Output:



```
┌(~/Desktop/CSL301/CA11_12340220)──────────────────────────────────────(amaydixit11@amayEOS:pts/5)─┐
└(17:04:50)──> ls                                                                         ──(Wed,Nov19)─┘
file1.txt  q2_12340220  q2_12340220.c  q3_12340220  q3_12340220.c
┌(~/Desktop/CSL301/CA11_12340220)──────────────────────────────────────(amaydixit11@amayEOS:pts/5)─┐
└(17:04:57)──> ./q3_12340220 cp file1.txt file2.txt                                       ──(Wed,Nov19)─┘
┌(~/Desktop/CSL301/CA11_12340220)──────────────────────────────────────(amaydixit11@amayEOS:pts/5)─┐
└(17:05:11)──> ls                                                                         ──(Wed,Nov19)─┘
file1.txt  file2.txt  q2_12340220  q2_12340220.c  q3_12340220  q3_12340220.c
┌(~/Desktop/CSL301/CA11_12340220)──────────────────────────────────────(amaydixit11@amayEOS:pts/5)─┐
└(17:05:13)──> ./q3_12340220 mv file1.txt file3.txt                                       ──(Wed,Nov19)─┘
┌(~/Desktop/CSL301/CA11_12340220)──────────────────────────────────────(amaydixit11@amayEOS:pts/5)─┐
└(17:05:23)──> ls                                                                         ──(Wed,Nov19)─┘
file2.txt  file3.txt  q2_12340220  q2_12340220.c  q3_12340220  q3_12340220.c
┌(~/Desktop/CSL301/CA11_12340220)──────────────────────────────────────(amaydixit11@amayEOS:pts/5)─┐
└(17:05:26)──>                                                                            ──(Wed,Nov19)─┘
```

Explanation:

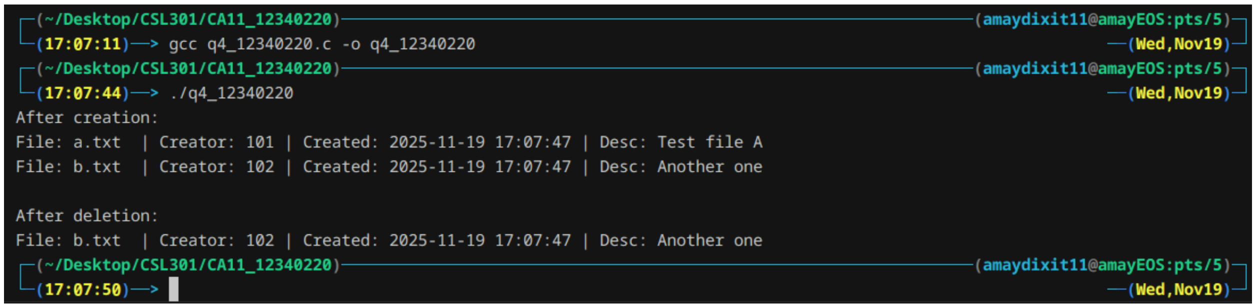
This utility program behaves like both the cp and mv commands depending on how it is executed. It allows two modes of usage: either by naming the executable cp or mv, or by specifying the command name (cp or mv) as the first argument. The program performs file copying using only low-level system calls such as open(), read(), write(), and close(), while preserving file permissions with stat(). For moving, it first tries rename(), and falls back to copy-and-delete (unlink()) for cross-device operations. This demonstrates handling file operations and permissions with core POSIX APIs.

Question 4. File Creation and Deletion with Metadata

```c
#include <stdio.h>
#include <string.h>
#include <time.h>

#define MAX_INODES 10
#define MAX_BLOCKS 10
#define DESC_LEN 64

struct inode {
   int in_use;
   int creator_id;
   time_t created_at;
   char description[DESC_LEN];
   int block;
};

struct inode inode_table[MAX_INODES];
int block_used[MAX_BLOCKS] = {0};
char filenames[MAX_INODES][32];

int allocate_inode() {
   for (int i = 0; i < MAX_INODES; i++)
       if (!inode_table[i].in_use) return i;
   return -1;
}

int allocate_block() {
   for (int i = 0; i < MAX_BLOCKS; i++)
```

```c
        if (!block_used[i]) return i;
    return -1;
}

int find_inode(const char *name) {
    for (int i = 0; i < MAX_INODES; i++)
        if (inode_table[i].in_use && strcmp(filenames[i], name) == 0)
            return i;
    return -1;
}

int create_file(const char *name, int creator_id, const char *desc) {
    int ino = allocate_inode();
    int blk = allocate_block();
    if (ino < 0 || blk < 0) return -1;

    inode_table[ino].in_use = 1;
    inode_table[ino].creator_id = creator_id;
    inode_table[ino].block = blk;
    time(&inode_table[ino].created_at);
    strncpy(inode_table[ino].description, desc, DESC_LEN - 1);

    block_used[blk] = 1;
    strcpy(filenames[ino], name);

    return 0;
}

int delete_file(const char *name) {
    int ino = find_inode(name);
    if (ino < 0) return -1;

    block_used[inode_table[ino].block] = 0;
    inode_table[ino].in_use = 0;
    memset(filenames[ino], 0, 32);

    return 0;
}

void show_inodes() {
    for (int i = 0; i < MAX_INODES; i++) {
        if (inode_table[i].in_use) {
            char tbuf[64];
            struct tm *tm = localtime(&inode_table[i].created_at);
            strftime(tbuf, sizeof(tbuf), "%Y-%m-%d %H:%M:%S", tm);

            printf("File: %s  | Creator: %d | Created: %s | Desc: %s\n",
                filenames[i],
                inode_table[i].creator_id,
                tbuf,
                inode_table[i].description);
        }
    }
}

int main() {
    create_file("a.txt", 101, "Test file A");
    create_file("b.txt", 102, "Another one");

    printf("After creation:\n");
    show_inodes();

    delete_file("a.txt");

    printf("\nAfter deletion:\n");
    show_inodes();

    return 0;
}
```

Output:



```
┌(~/Desktop/CSL301/CA11_12340220)──────────────────────────────(amaydixit11@amayEOS:pts/5)┐
└(17:07:11)──> gcc q4_12340220.c -o q4_12340220                               ──(Wed,Nov19)┘
┌(~/Desktop/CSL301/CA11_12340220)──────────────────────────────(amaydixit11@amayEOS:pts/5)┐
└(17:07:44)──> ./q4_12340220                                                  ──(Wed,Nov19)┘
After creation:
File: a.txt  | Creator: 101 | Created: 2025-11-19 17:07:47 | Desc: Test file A
File: b.txt  | Creator: 102 | Created: 2025-11-19 17:07:47 | Desc: Another one

After deletion:
File: b.txt  | Creator: 102 | Created: 2025-11-19 17:07:47 | Desc: Another one
┌(~/Desktop/CSL301/CA11_12340220)──────────────────────────────(amaydixit11@amayEOS:pts/5)┐
└(17:07:50)──>                                                                ──(Wed,Nov19)┘
```

Explanation:

This program simulates a simplified file system that uses inodes to manage files and their metadata. Each inode stores information such as creator ID, creation timestamp, and a short description. When a new file is created, the program allocates a free inode and data block, fills in metadata, and links the filename to the inode. Deleting a file frees both its inode and associated data block. The program includes functionality to display current inode information for all active files. This assignment reinforces the concept of inodes, resource allocation, and internal file system organization.