1) open_software.c

- **File Description:**
  This C program is designed to open a software application when a specific signal (SIGUSR3) is received. It uses the system function to execute the specified software executable.

- **Included Headers:**
  1) stdio.h: Provides functions for input/output operations, such as printf.
  2)stdlib.h: Provides general utility functions, such as system.
  3)string.h: Provides string manipulation functions.
  4)stdbool.h: Defines the bool data type and associated values (true and false).

- **Macros:**
  EXECUTABLE_PATH: This macro defines the path to the software executable that will be launched.

- **Functions:**
  1)openSoftware(void):
    - This function prints a message to the console indicating that the software is being executed.
    - It then calls the system function, passing the EXECUTABLE_PATH as an argument, which effectively runs the specified software executable.

  2)signal_handler(int signum):
    - This function is a signal handler that is called when a specific signal (SIGUSR3) is received.
    - If the received signal is SIGUSR3, it calls the openSoftware function to execute the software.

  3)main(void):
    - The main function is the entry point of the program.
    - It sets up the signal_handler function to handle the SIGUSR3 signal using the signal function.
    - It then enters an infinite loop with a sleep call to keep the program running indefinitely.

2) split_terminal.c

- **File Description:**
  This C program is designed to simulate the "Ctrl + Shift + 5" keyboard shortcut, which is typically used for splitting the terminal or creating a new terminal window in various Linux desktop environments. The program uses the X11 library to send the necessary key events to the X server, effectively triggering the shortcut.

- **Included Headers:**
  1)stdio.h : Provides functions for input/output operations, such as printf and fprintf.
  2)stdlib.h: Provides general utility functions, such as exit.
  3)X11/Xlib.h : Provides the core X11 library functions for interacting with the X Window System.
  4)X11/keysym.h : Defines keysym values for representing keyboard keys.
  5)X11/extensions/XTest.h: Provides functions for generating synthetic input events, including keyboard and mouse events.

- **Functions:**
  1)splitTerminal(void):
    - This function simulates the "Ctrl + Shift + 5" keyboard shortcut by sending the necessary key events to the X server
    - It opens a connection to the X server using XOpenDisplay.
    - It then sends the following key events using XTestFakeKeyEevent:
      - Press and release the Control(Ctrl) key.
      - Press and release the Shift key.
      - Press and release the “5” key.
    - After sending the key events, it flushes the output buffer and synchronises the X server using XFlush and XSync.
    - Finally, it closes the connection to the X server using XCloseDisplay

  2)signal_handler(int signum):
    - This function is a signal handler that is called when a specific signal(SIGUSR2) is received.
    - If the received signal is SIGUSR2, it calls the splitTerminal function to simulate the "Ctrl + Shift + 5" keyboard shortcut.

  3)main(void):
    - It sets up the signal_handler function to handle the SIGUSR2 signal using the signal function.
    - It prints a message indicating that the "Ctrl + Shift + 5" keyboard shortcut is being executed.
    - It calls the splitTerminal function to simulate the keyboard shortcut.
    - It prints another message indicating that the keyboard shortcut has been executed.
    - It then enters an infinite loop with a sleep call to keep the program running indefinitely.

3) compile_execute.c

- **File Description:**
  This C program is designed to compile and execute a C source file from the command line. It takes the filename as a command-line argument, compiles the file using the GCC compiler, and then runs the resulting executable.

- **Included Headers:**
  1)unistd.h: Provides access to POSIX operating system APIs, including system.
  Rest all headers are the same as used in the previous two .c files.

- **Macros:**
  1)MAX_COMMAND_LENGTH: Defines the maximum length of the compilation and execution commands (set to 1024 characters).
  2)MAX_FILENAME_LENGTH: Defines the maximum length of the filename (set to 256 characters).

- **Functions:**
  1)compileExecute(const char *filename):
    - This function takes a filename as input and compiles and executes the corresponding C source file.
    - It first checks if the provided filename is valid (not NULL and not an empty string).
    - It constructs the command to compile the C program using sprintf and the gcc compiler. The compiled executable is named <filename>.out.
    - It executes the compilation command using system.
    - If the compilation is successful, it constructs the command to run the compiled executable using sprintf.
    - It executes the run command using system.
    - If either the compilation or execution fails, it prints an error message and exits the program using exit(1).

  2)main(int argc, char *argv[]):
    - It checks if the program was called with at least one command-line argument (the filename).
    - If no filename is provided, it prints a usage message and exits the program using exit(1).
    - It calls the compileExecute function with the provided filename argument (argv[1]).

4) call_for_compile_execute.c

- **File Description:**
  This C program is designed to prompt the user to enter a filename of a C program, and then it calls another program (compile_execute) to compile and execute the specified C file. The program listens for the SIGUSR1 signal and triggers the compilation and execution process when that signal is received.

- **Included Headers:**
  1)fcntl.h: Provides functions for file control operations, although not used in this code.
  Rest all headers are the same as used in the previous three .c files.

5) copyPaste.c

- **Description:**
  The copyPaste function retrieves text data from the primary selection (clipboard) using the xclip command. It then appends the retrieved text to a file named “Stored_data.txt”. If the file does not exist, it creates a new one.
  Function Behavior:
  1. Invokes the xclip utility to retrieve text data from the primary selection (clipboard).
  2. Appends the retrieved text to the “Stored_data.txt” file.
  3. If the file does not exist, creates a new file named “Stored_data.txt”.
  4. Adds a newline character to separate entries within the file.
- **Usage Notes:**
  - Ensure that the xclip utility is installed on your system.
  - The “Stored_data.txt” file should be in the same directory as this program.
  - Modify the code as needed for your specific use ca

6)saver[1].c

- **Description:**
  The saveOnActiveWindow function is designed to simulate the action of saving a file in an active window. It achieves this by sending a specific key combination (Ctrl+S) to the currently focused window. This functionality is useful for automating the process of saving files within an application.
  Included Headers:
  - <stdio.h>: Provides functions for input/output operations, such as printf.
  - <unistd.h>: Provides access to the POSIX operating system API, including the sleep function.
  - <X11/Xlib.h>: Contains declarations for X Window System functions and data types.
  - <X11/Xutil.h>: Provides additional utility functions for X Window System programming.
- **Macros:**
  - EXECUTABLE_PATH: This macro defines the path to the software executable that will be launched.
- **Function Behavior:**
  1. Opens an X display connection using XOpenDisplay.
  2. Waits for a brief period (2 seconds) to allow the user to switch to an active window where Ctrl+S triggers the save action.
  3. Retrieves the currently focused window using XGetInputFocus.
  4. Simulates the “Save” action by sending the Ctrl+S key combination to the focused window using XSendEvent.
  5. Closes the X display connection using XCloseDisplay.
- **Usage Notes:**
  - Ensure that the X11 library is linked when compiling the program (e.g., gcc saver2.c -lX11).
  - The focusedWindow represents the active window where the save action will be triggered.
  - Modify the code as needed to adapt it to your specific use case.

7) driver.c

The driver function is part of a Linux kernel module for a custom USB macro pad device. It is called when the macro pad is connected to the system. This function initializes the input device, sets up its capabilities, and registers it with the kernel. The macro pad is designed to trigger specific actions based on key presses.

- **Included Headers:**
  - <linux/module.h>: Provides module-related macros and functions.
  - <linux/input.h>: Contains definitions for input event handling.
  - <linux/usb.h>: Includes USB-related structures and functions.
  - <linux/kernel.h>: Provides kernel-level macros and functions.
  - <linux/init.h>: Contains initialization and cleanup macros.

- **Macros:**
  - MACROPAD_VENDOR_ID: Defines the vendor ID of the macro pad device.
  - MACROPAD_PRODUCT_ID: Defines the product ID of the macro pad device.
  - MACROPAD_KEY_A, MACROPAD_KEY_B, MACROPAD_KEY_C: Custom key addresses to map in kernel space for specific keys on the macro pad.
  - compile_SIGNAL, terminal_SIGNAL, software_SIGNAL: Custom signal numbers used for communication with user space.

- **Function Behavior:**

  1. Initializes an input device structure (input_dev) for the macro pad.
  2. Sets the input device properties:
     - name: Specifies the name of the input device (e.g., “Macropad”).
     - phys: Specifies the physical location of the device (e.g., “/macropad/input0”).
     - id: Defines the bus type, vendor ID, product ID, and version.
     - Capabilities: Sets up key capabilities for specific keys (A, B, C) using input_set_capability.
     - event: Associates the macropad_event function as the event handler.
  3. Registers the input device with the kernel using input_register_device.
  4. Associates the input device with the USB interface using usb_set_intfdata.
  5. Initializes an input device structure (input_dev) for the macro pad.
  6. Sets the input device properties:
     - name: Specifies the name of the input device (e.g., “Macropad”).
     - phys: Specifies the physical location of the device (e.g., “/macropad/input0”).
     - id: Defines the bus type, vendor ID, product ID, and version.
     - Capabilities: Sets up key capabilities for specific keys (A, B, C) using input_set_capability.
     - event: Associates the macropad_event function as the event handler.
  7. Registers the input device with the kernel using input_register_device.
  8. Associates the input device with the USB interface using usb_set_intfdata.

- **Usage Notes:**

- Ensure that the X11 library is linked when compiling the program (e.g., gcc saver2.c -lX11).
- Modify the key codes and associated actions as needed for your specific macro pad configuration.