Project 4: **Button-Activated Periodic Tasks Using Real-Time Scheduling**

Demo Day: December 17, 2024 @ 4:00 PM

By: Htet Hnin Su Wai, Martin Azabo, Manuela Miranda

Components: Jumper Wires x6
Button
GPIO Extension Board
Bread Board
Resistor (220 Ω)
S8050 Transistor
40 Pin GPIO Cable

Design Application:

1. **Task 1**: **Button Monitoring Task**
   - Detects button presses and prints "Button Pressed!" when the button state changes
   - Monitors a button connected to **GPIO Pin 26**
2. **Task 2**: **Periodic Logging Task**
   - Periodically logs a timestamp to the terminal every **2 seconds**.
   - Demonstrates a periodic real-time task with a predefined reservation (budget and period).

The tasks are integrated with **real-time system calls** (set_rsv, wait_until_next_period, and cancel_rsv) developed as part of Project 3. These system calls ensure proper time reservation, periodic execution, and resource management for both tasks.

Challenges Faced:

1. Real-Time System Call Integration

- **Challenge**: Ensuring the system calls (sys_set_rsv, wait_until_next_period) worked correctly with both tasks.
- **Solution**: Debugged syscalls.c logic, ensuring that tasks periodically wait and reset their reservation without error. Debug prints were added to confirm success.

2. GPIO Button Setup

- **Challenge**: Properly configuring GPIO pins and reading button state using /sys/class/gpio.
- **Solution**: Added checks to ensure GPIO export and direction setup worked correctly. Debug prints were included to display GPIO values during testing.

3. Program Termination

- **Challenge**: Allowing tasks to terminate gracefully after a fixed number of button presses or using CTRL+C.
- **Solution**: Added signal handling (SIGUSR1) and a loop counter in the button monitoring task to terminate after a specified number of presses

.4. Debugging Wait Period System Call

- **Challenge**: The program initially froze due to improper behavior of wait_until_next_period.
- **Solution**: Fixed syscalls.c logic and ensured the periodic behavior worked by testing with debug prints.

5. **GPIO Button Setup**

- **Challenge**: The button was initially not detected due to hardware wiring issues and incorrect GPIO configuration. The program did not print the expected "Button Pressed!" output.
- **Solution**:
  - Double-checked **GPIO wiring** to ensure that the button was properly connected to **GPIO Pin 26** and Ground.
  - Verified the correct physical pin on the Raspberry Pi GPIO header (Pin 37 for GPIO 26).
  - Tested the button functionality using a **Python GPIO script** to confirm that the pin reads high/low values accurately.
  - Fixed the read_gpio_value() function to ensure proper reading from /sys/class/gpio/gpio26/value.
  - Debug prints were added to check the raw GPIO input values during execution.

Individual Contributions:

Team Member	Contributions
Htet Hnin Su Wai	Developed the test_project4.c program.
	Implemented button monitoring task (Task 1).
	An integrated system calls for real-time scheduling.
	Debugged and improved syscalls.c for proper task behavior.
Martina Azabo	Verified GPIO setup and hardware connections.
	Tested the program on Raspberry Pi.

	Implemented periodic logging task (Task 2).
	Added graceful termination logic.
Manuela Miranda	Research

Testing Process:

1. Verified the GPIO button functionality with a Python script.
2. Compiled and ran the test_project4.c program on the Raspberry Pi.
3. Confirmed:
   - "Button Pressed!" is printed when the button is pressed.
   - Periodic logging outputs timestamps every **2 seconds**.
4. Ensured tasks terminate after a set number of button presses.

Expected Output:

When the button is pressed:

```
Task 1: Button monitoring started...
Task 2: Logging task started...
Task 2: Periodic logging task executed at 1734391334 seconds
Task 2: Periodic logging task executed at 1734391336 seconds
...
Task 2: Periodic logging task executed at 1734391340 seconds
```

Both tasks were successful.