Multithreading with Shared Memory and Signal Handling Documentation

---

Design Decisions:

1. **Multithreading Approach:**
    - Utilized pthread library for creating and managing threads.
    - Implemented two concurrent threads that increment a shared counter alternately.
    - Used a mutex to ensure thread safety and prevent data races while accessing shared resources.
2. **Shared Memory Management:**
    - Employed System V shared memory for inter-process communication.
    - Defined a shared memory segment with a specific key and size to store an integer counter.
    - Utilized `shmget` to create or retrieve the shared memory segment.
    - Attached the shared memory segment to the process address space using `shmat`.
    - Cleaned up shared memory resources using `shmdt` and `shmctl`.
3. **Thread Synchronization:**
    - Implemented thread synchronization using a mutex.
    - Used `pthread_mutex_lock` and `pthread_mutex_unlock` to ensure exclusive access to shared resources, specifically the shared counter.
4. **Signal Handling:**
    - Implemented signal handling for the SIGINT signal (Ctrl+C) using the `signal` function.
    - Registered a signal handler function (`sigint_handler`) to gracefully handle SIGINT.
    - Upon receiving SIGINT, the program cleans up allocated resources and exits gracefully.

Approach to Challenges:

1. **Thread Synchronization:**
    - To ensure thread safety and prevent race conditions, I used a mutex lock to synchronize access to the shared counter.
    - Each thread acquires the lock before accessing the shared counter, ensuring that only one thread can modify it at a time.
    - Once the operation is complete, the lock is released to allow other threads to access the shared counter.
2. **Shared Memory Management:**
    - Utilized System V shared memory to facilitate communication between threads.

- Defined a specific key and size for the shared memory segment to ensure uniqueness and appropriate allocation.
- Employed `shmat` to attach the shared memory segment to the process address space, allowing threads to access the shared counter.
- Implemented proper cleanup procedures (`shmdt` and `shmctl`) to release shared memory resources upon program termination.

3. **Signal Handling:**

   - Implemented a signal handler function (`sigint_handler`) to handle the SIGINT signal (Ctrl+C).
   - Registered the signal handler using the `signal` function to intercept SIGINT.
   - Upon receiving SIGINT, the program gracefully exits by calling the cleanup function, which releases allocated resources and terminates the program.

Summary:

The design of the multithreading with shared memory and signal handling application focused on ensuring thread safety, efficient resource management, and graceful termination. By employing mutex locks for synchronization, System V shared memory for inter-thread communication, and proper signal handling mechanisms, the application achieves robustness and reliability. These design decisions enable the application to increment a shared counter safely, manage shared memory efficiently, and handle SIGINT signals gracefully, thereby demonstrating proficiency in system programming and automation.