sem_waiter.c

```c
/*****************************************************************************
 * Copyright (C) 2023 by Parth Thakkar
 *
 * Redistribution, modification or use of this software in source or binary
 * forms is permitted as long as the files maintain this copyright. Users are
 * permitted to modify this and use it to learn about the field of embedded
 * software. Parth Thakkar and the University of Colorado are not liable for
 * any misuse of this material.
 * ***************************************************************************/

/**
 * @file    sem_waiter.c
 * @brief   This program demonstrates the use of POSIX semaphores for inter-process
 *          communication and synchronization. It waits on a semaphore until it's
 *          posted by another process, allowing for coordinated execution.
 *
 *          This could be used in scenarios where it's necessary to ensure that
 *          certain resources are not accessed by multiple processes simultaneously
 *          or to synchronize the execution order of processes.
 *
 *
 * @author  Parth Thakkar
 *
 */

#include <stdio.h>
#include <fcntl.h>    // For O_* constants
#include <sys/stat.h> // For mode constants
#include <semaphore.h>
#include <stdlib.h>
#include <unistd.h>

#define SEM_NAME "/semaphore_custom"

int main()
{
    printf("Process with PID %d waiting on semaphore...\n", getpid());

    // Open or create the semaphore
    /*
     * 0644 These are the permissions for the new semaphore if it is created. This is a octal number
     * representing the semaphore's permissions in a Unix/Linux environment. The first digit is always
     * zero, the second digit represents permissions for the owner (read and write), the third digit
     * represents permissions for the owner's group (read), and the fourth digit represents permissions for
     * others (read). This means the owner can read and modify the semaphore, while others can only read
     * its value.
     *
     * O_CREAT: This flag indicates that the semaphore should be created if it does not already exist.
     * sem_open can take multiple flags, combined using the bitwise OR operator (|), but in this case, only
     * O_CREAT is used. Other flags could include O_EXCL, which, when used with O_CREAT, will make sem_open
     * fail if the semaphore already exists, ensuring that the semaphore is newly created.
     *
     * 0: This is the initial value for the semaphore if it is being created. In this case, the semaphore is initialized to 0. This
     * value can be used to control access to a resource by having threads or processes wait until the semaphore's value is greater
     * than zero.
     */
    sem_t *sem = sem_open(SEM_NAME, O_CREAT, 0644, 0);
    if (sem == SEM_FAILED)
    {
        perror("sem_open failed");
        exit(EXIT_FAILURE);
    }

    // Wait on the semaphore
    if (sem_wait(sem) < 0)
    {
        perror("sem_wait failed");
        exit(EXIT_FAILURE);
    }

    printf("Semaphore posted, process %d continuing...\n", getpid());
    // Close the semaphore to release resources. This does not remove the semaphore,
    // but it detaches it from the process that called sem_close.
    sem_close(sem);

    // Unlink the semaphore, removing its name from the system. This is necessary
    // to clean up and ensure that the semaphore does not persist after all
    // processes using it have terminated.
    sem_unlink(SEM_NAME);

    return EXIT_SUCCESS;
}
```