Project 2

Wait Queue

A **Wait Queue** is a synchronization mechanism in the Linux kernel used to put processes to sleep while waiting for a specific condition to be met. Once the condition is satisfied, the processes are awakened. It is commonly used to avoid busy-waiting, thereby improving system efficiency.

Under normal circumstances, a function using a wait queue must know the name or the pointer of the wait queue. This is because a wait queue is represented as a variable (usually of type `wait_queue_head_t`), and the function needs the address of this variable to perform operations.

> In this lab, you need to implement a custom wait queue-like functionality in kernel space, allowing user applications to operate through the system call.

The target output is as follows: threads exit the wait queue in FIFO order.

```
enter wait queue thread_id: 0
enter wait queue thread_id: 1
enter wait queue thread_id: 2
enter wait queue thread_id: 6
enter wait queue thread_id: 7
enter wait queue thread_id: 8
enter wait queue thread_id: 9
enter wait queue thread_id: 5
enter wait queue thread_id: 3
enter wait queue thread_id: 4
start clean queue ...
exit wait queue thread_id: 0
exit wait queue thread_id: 1
exit wait queue thread_id: 2
exit wait queue thread_id: 6
```

```
exit wait queue thread_id: 7
exit wait queue thread_id: 8
exit wait queue thread_id: 9
exit wait queue thread_id: 5
exit wait queue thread_id: 3
exit wait queue thread_id: 4
```

Sample Code (User Space)

Except for `syscall(xxx, 1);` and `syscall(yyy, 2);`, please do not modify any other parts of the code.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/syscall.h>

#define NUM_THREADS 10

void *enter_wait_queue(void *thread_id)
{
    fprintf(stderr, "enter wait queue thread_id: %d\n",
        *(int *)thread_id);

    /*
        // your syscall here
        syscall( xxx , 1);
    */

    fprintf(stderr, "exit wait queue thread_id: %d\n",
        *(int *)thread_id);
}

void *clean_wait_queue()
{
    /*
        // your syscall here
```

```
        syscall( xxx , 2);
    */
}

int main()
{
    void *ret;
    pthread_t id[NUM_THREADS];
    int thread_args[NUM_THREADS];

    for (int i = 0; i < NUM_THREADS; i++)
    {
        thread_args[i] = i;
        pthread_create(&id[i], NULL, enter_wait_queue,
            (void *)&thread_args[i]);
    }

    sleep(1);
    fprintf(stderr,  "start clean queue ...\n");
    clean_wait_queue();

    for (int i = 0; i < NUM_THREADS; i++)
    {
        pthread_join(id[i], &ret);
    }
    return 0;
}
```

Sample Code (Kernel Space)

Here is a program example. You are free to modify the contents of this file as needed.

```
static int enter_wait_queue(void)
{
    return 0;
}
```

```
static int clean_wait_queue(void)
{
    return 0;
}

SYSCALL_DEFINE1(call_wait_queue, int, id)
{
    switch (id){
        case 1:
            enter_wait_queue()
            break;
        case 2:
            clean_wait_queue();
            break;
    }

    return 0;
}
```

Todo

> Implement a system call `call_my_wait_queue(int id)`.
> This function takes an argument
> `id` to determine which of the following two operations to perform:
>
> 1. **Add a thread to the wait queue to wait**
>
>   If an error occurs, return `0`; on success, return `1`.
>
> 2. **Remove threads from the wait queue, allowing them to exit**
>
>   If an error occurs, return `0`; on success, return `1`.
>   The removal order must follow the
>   **FIFO (First In, First Out)** principle.

The target output is as follows: threads exit the wait queue in FIFO order.

Hints

- You can use kernel-provided wait queue related functions to implement this functionality.
- You need to declare a `my_wait_queue` in kernel space.

Project Submission

- Due time: **23:55 29th Dec.**
- The demo will be held from **30th Dec.**
- Please fill out this form to choose your demo time before **29th Dec.**
- An on-site demo of this project is required.
- During the on-site demo, the TAs will execute several programs they wrote to check the correctness of your system calls.
- When demonstrating your projects, the TAs will ask you some questions regarding your projects. Part of your project grade is determined by your answers to the questions.
- Report Content:
  - Remember to write the names and student IDs of all members of your team.
  - Your report should be in **hackmd** document form and contain:
    - Your source code
    - the execution results
  - Submit the URL of your **hacked** document to the new-eeclass.
- Late submissions will **NOT** be accepted.