Lab08 - Assignment

> 姓名：周星达 学号：21373339

1. 用两个线程交替打印0-99，期望输出如下：

```
thread1: 0
thread2: 1
thread1: 2
thread2: 3
.
.
.
thread1: 98
thread2: 99
```

```
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include <stdio.h>
int count = 0;
pthread_mutex_t mutex;
pthread_cond_t cond1, cond2;

void *thread1(void *arg)
{
    while (1)
    {
        pthread_mutex_lock(&mutex);
        if (count >= 100)
        {
            exit(1);
        }
        printf("thread1: %d \n", count);
        count++;
        pthread_cond_signal(&cond2);
        pthread_cond_wait(&cond1, &mutex);
        pthread_mutex_unlock(&mutex);
        sleep(1);
    }
}
void *thread2(void *arg)
{
    while (1)
    {
        sleep(1); //保证是线程1先打印
        pthread_mutex_lock(&mutex);
        printf("thread2: %d \n", count);
        count++;
        pthread_cond_signal(&cond1);
        pthread_cond_wait(&cond2, &mutex);
        pthread_mutex_unlock(&mutex);
    }
}
int main()
{
    pthread_t pid1, pid2;
    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&cond1, NULL);
    pthread_cond_init(&cond2, NULL);

    pthread_create(&pid1, NULL, thread1, NULL);
    pthread_create(&pid2, NULL, thread2, NULL);
    pthread_join(pid1, NULL);
    pthread_join(pid2, NULL);

    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&cond1);
    pthread_cond_destroy(&cond2);
    return 0;
}
```

2. 修改以下代码，使counter最终值是 `5000000`。

修改后：

```
#include <stdio.h>
#include <pthread.h>

#define thread_num 5

int counter;

void *add(void *arg)
{
    for (int i = 0; i < 1000000; i++)
    {
        counter++;
    }
}

int main()
{
    pthread_t tids[thread_num];
    for (int i = 0; i < thread_num; i++)
    {
        pthread_create(&tids[i], NULL, add, NULL);
        pthread_join(tids[i], NULL);
    }
    printf("counter=%d\n", counter);
}
```

3. 用C语言完成Leetcode 1117 H2O 生成，给出通过的代码。

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <stdbool.h>
#include <semaphore.h>

typedef struct
{
    volatile int oc;
    volatile int hc;
    sem_t os;
    sem_t hs;
    sem_t rs;
} H2O;

H2O *h2oCreate()
{
    H2O *obj = (H2O *)malloc(sizeof(H2O));

    obj->oc = 0;
    obj->hc = 0;

    sem_init(&obj->os, 0, 1);
    sem_init(&obj->hs, 0, 2);
    sem_init(&obj->rs, 0, 1);

    return obj;
}

bool reset(H2O *obj)
{
    return obj->oc == 1 && obj->hc == 2;
}
void check_reset(H2O *obj)
{
    sem_wait(&obj->rs);
    if (reset(obj))
    {
        obj->hc = 0;
        obj->oc = 0;
        sem_post(&obj->os);
        sem_post(&obj->hs);
        sem_post(&obj->hs);
    }
    sem_post(&obj->rs);
}
```

```
void hydrogen(H2O *obj)
{
    // releaseHydrogen() outputs "H". Do not change or remove this line.
    check_reset(obj);
    while (0 != sem_trywait(&obj->hs))
    {
        usleep(300);
    }
    releaseHydrogen();
    obj->hc += 1;
    check_reset(obj);
}

void oxygen(H2O *obj)
{
    check_reset(obj);
    while (0 != sem_trywait(&obj->os))
    {
        usleep(300);
    }
    // releaseOxygen() outputs "O". Do not change or remove this line.
    releaseOxygen();
    obj->oc += 1;
    check_reset(obj);
}

void h2oFree(H2O *obj)
{
    // User defined data may be cleaned up here.
    sem_destroy(&obj->rs);
    sem_destroy(&obj->hs);
    sem_destroy(&obj->os);
    free(obj);
}
```

4. 生产者消费者模型

生产者消费者模型是条件变量最经典的使用场景之一，该问题描述了共享固定大小缓冲区的两个线程——即所谓的“生产者”和“消费者”——在实际运行时会发生的问题。生产者的主要作用是生成一定量的数据放到缓冲区中，然后重复此过程。与此同时，消费者也在缓冲区消耗这些数据。该问题的关键就是要保证生产者不会在缓冲区满时加入数据，消费者也不会在缓冲区中空时消耗数据。

生产者消费者问题主要要注意以下三点：

- 在缓冲区为空时，消费者不能再进行消费
- 在缓冲区为满时，生产者不能再进行生产
- 在一个线程进行生产或消费时，其余线程不能再进行生产或消费等操作，即保持线程间的同步

假设缓冲区上限为 20，生产者和消费者线程各 10 个，请编写程序实现一个生产者消费者模型。**在每次生产、消费时将当前动作类型（produce/consume）与缓冲区内容量输出到屏幕，给出代码和操作步骤。**

```c
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <unistd.h>

int current = 0; //生产者运行加一，消费者运行减一
int buf[20];     //缓冲区
int in = 0, out = 0;
int items = 0, spaces = 20;
int flag;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t notfull = PTHREAD_COND_INITIALIZER;
pthread_cond_t notempty = PTHREAD_COND_INITIALIZER;

void *producer(void *arg)
{
    while (flag != 0)
    {
        pthread_mutex_lock(&mutex);
        while (!spaces)
        {
            pthread_cond_wait(&notfull, &mutex);
        }
        buf[in] = current++;
        in = (in + 1) % 20;
        items++;
        spaces--;
        printf("producer %zu , current = %d\n", pthread_self(), current);
        for (int i = 0; i < 20; i++)
        {
            if (buf[i] != -1)
            {
                printf("%-4d", buf[i]);
            }
        }
        printf("\n\n");

        pthread_cond_signal(&notempty);
        pthread_mutex_unlock(&mutex);
    }
    pthread_exit(NULL);
}

void *consumer(void *arg)
{
    while (flag != 0)
    {
        pthread_mutex_lock(&mutex);
        while (!items)
        {
            pthread_cond_wait(&notempty, &mutex);
        }
        buf[out] = -1;
        out = (out + 1) % 20;
        current--;
        items--;
        spaces++;
        printf("consumer %zu ,current = %d\n", pthread_self(), current);
        for (int i = 0; i < 20; i++)
        {
            if (buf[i] != -1)
            {
                printf("%-4d", buf[i]);
            }
        }
        printf("\n\n");

        pthread_cond_signal(&notfull);
        pthread_mutex_unlock(&mutex);
    }
    pthread_exit(NULL);
}

int main()
{
    memset(buf, -1, sizeof(buf));
    flag = 1;
    pthread_t pro[10], con[10];
    int i = 0;
    for (int i = 0; i < 10; i++)
    {
        pthread_create(&pro[i], NULL, producer, NULL);
        pthread_create(&con[i], NULL, consumer, NULL);
    }
    sleep(1);
    flag = 0;
    for (int i = 0; i < 10; i++)
    {
        pthread_join(pro[i], NULL);
        pthread_join(con[i], NULL);
    }
    return 0;
}
```