线程池的设计

1. 为什么需要线程池

为了并行开发，利用多CPU的优势，开启一个线程去干某件事情。

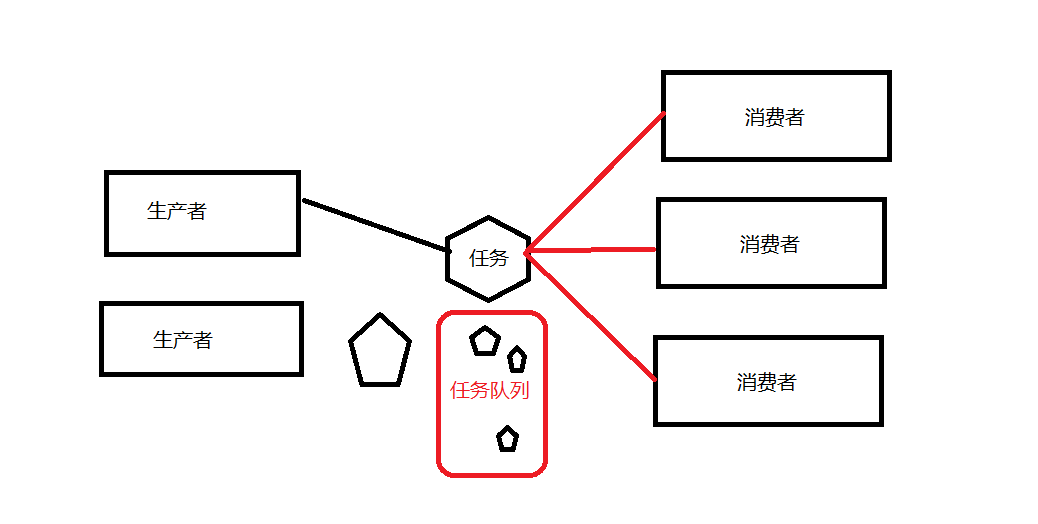
任务，然后创建一个线程去执行这个任务，任务执行完了之后，就把线程给退出掉。

我就先创建一大堆的线程，放在这个池子里，他们在程序运行过程中，一般都不销毁。

1. 线程池如何去设计。

生产和消费的关系：

有线程去生成 任务 有线程去消费



5 typedef struct queue

6 {

7 int header;

8 int tail;

9 int size;

10 int capcity;

11 void \*\*\_buf;

12 } queue\_t;

buf他是一个动态的数组，header是读指针，tail是写指针

如果采用链表，需要经常去malloc和free（链表的节点）。

typedef struct async\_queue

{

pthread\_mutex\_t mutex;

pthread\_cond\_t cond;

int waiting\_threads;

queue\_t\* queue;

int quit; // 0 表示不退出 1 表示退出

/\* 调试变量 \*/

long long tasked; // 已经处理完的任务数量

} async\_queue\_t;

pthread\_cond\_t cond; 条件变量

int pthread\_cond\_wait(pthread\_cond\_t \*restrict *cond***,** pthread\_mutex\_t \*restrict *mutex***);**

等待某个条件满足：

pthread\_mutex\_lock(mutex);

cond就是一个信号，

1. 条件是否满足，
2. 如果不满足，则解锁mutex
3. 然后等待cond信号， 挂起了

pthread\_cond\_wait(cond, mutex );

pthread\_cond\_timedwait(&(q->cond), &(q->mutex), &outtime);

这个等待可以通过

pthread\_cond\_signal(&(q->cond));去唤醒线程结束等待。 只会给一个线程发送唤醒的信号。

pthread\_cond\_broadcast(&(q->cond)); 他会给所有的线程发送信号。

第二种线程池的设计方案：

1. eventfd + epoll

eventfd，他是由内核实现的线程和进程之间通信的一种方案，8个字节内存，初始值就是initval，0,1，任意值。flags

可以在某个地方去write，也可以在另一个地方去read这个数据。

write：在eventfd的现有的值上面 加上这个写入的值，追加写。

read : 就是把这个值读出来，读完后内核对应的值就变成了初始值。

eventfd就是内核维护的一个计数器而已。

int eventfd(unsigned int initval, int flags);

从队列里去任务：

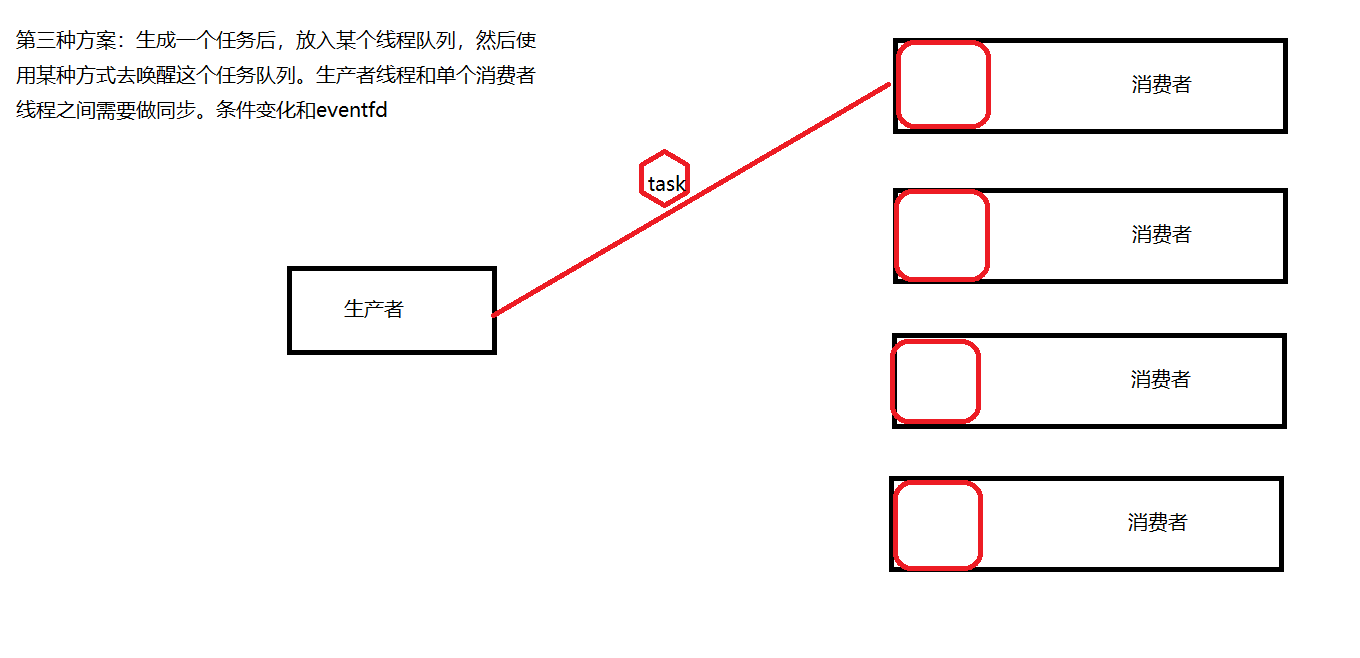
epoll\_wait因为我们在插入任务的时候写入了数据，所以会返回。

1. async\_queue\_create

创建了一个eventfd，初始值为0.

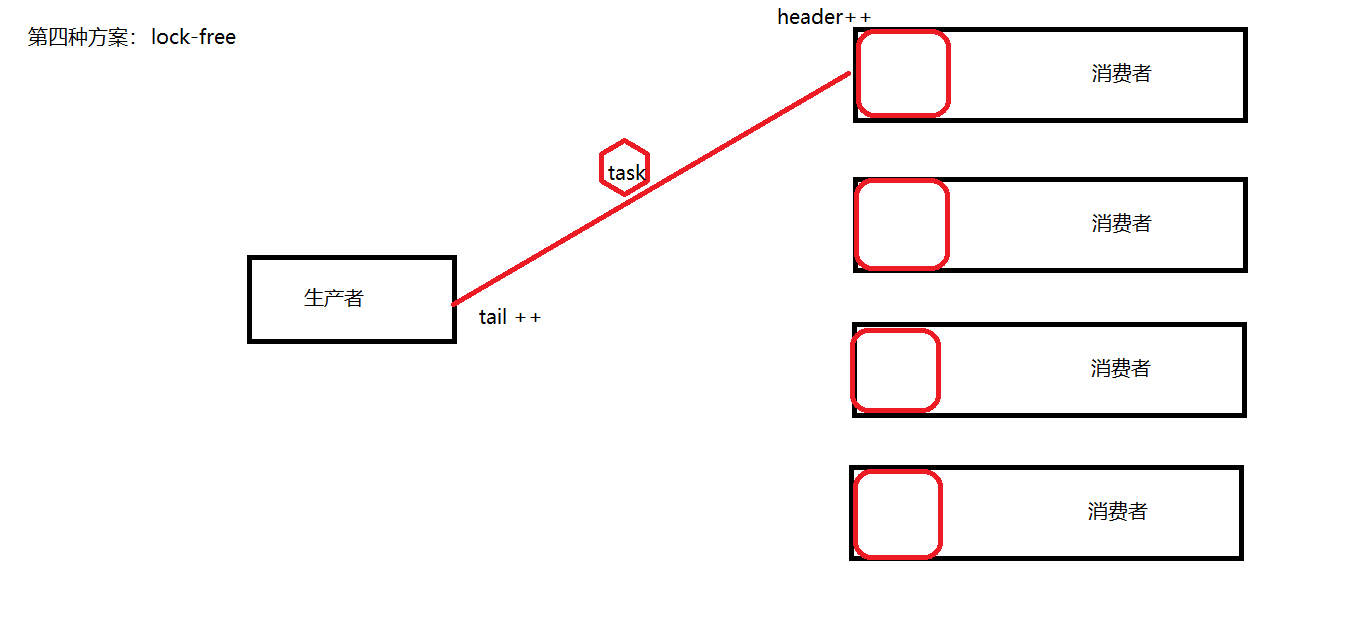
1. async\_queue\_push\_tail的时候，我就往这个eventfd里写入1。 > 1，插入了多个任务，
2. async\_queue\_pop\_head的时候，就read这个eventfd，返回一个task。read只触发一次，返回一个任务。

第三种设计方案：



第四种方案：

LOCK-FREE + 线程死循环



static void \*threadpool\_thread(void \*threadpool)

{

threadpool\_t \*pool = (threadpool\_t \*)threadpool;

for(;;)

{

threadpool\_task\_t\* task = (threadpool\_task\_t\*)**async\_queue\_pop\_head\_nowait**(pool->queue, 50);

if (task != NULL)

{

(\*(task->run))(task->argument);

free(task);

}

if(((pool->shutdown == immediate\_shutdown) || (pool->shutdown == graceful\_shutdown) )

&& (async\_queue\_is\_empty(pool->queue)))

{

//printf("--- thread %d is exit.\n", pthread\_self());

break;

}

}

return(NULL);

}

brks接下来的工程开发设计：

