Lab-5

1. 实验：了解并实现 Stride Scheduling 调度算法，为不同线程设置不同优先级，使得其获得与优先级成正比的运行时间。

   用一句话来概括Stride算法就是：让每个线程分到的时间和优先级成正比。这在现代操作系统中，是更为合理的（相比rr算法）

   实现流程，参考了ucore的教程：

   1. 为每个runnable的进程设置一个当前状态stride，表示该进程当前的调度权。另外定义其对应的pass值，表示对应进程在调度后，stride 需要进行的累加值。
   2. 每次需要调度时，从当前 runnable 态的进程中选择 stride最小的进程调度。
   3. 对于获得调度的进程P，将对应的stride加上其对应的步长pass（只与进程的优先权有关系）。
   4. 在一段固定的时间之后，回到 2.步骤，重新调度当前stride最小的进程。可以证明，如果令 P.pass =BigStride / P.priority 其中 P.priority 表示进程的优先权（大于 1），而 BigStride 表示一个预先定义的大常数，则该调度方案为每个进程分配的时间将与其优先级成正比。

   以上是ucore教程中的，但我认为stride翻译为步长更合适，因此我把pass和stride的意义调换了

   于是我要做的有

   1. 挑选合适的数据结构

      这里我使用alloc::collections::BinaryHeap，是一个优先队列

   2. 选择BigStride

      满足STRIDE_MAX – STRIDE_MIN <= BigStride即可，所以我使用usize::MAX

   3. 完成trait中的各个函数

      有了合适的数据结构，这几个函数都很好实现了，add_thread只需把thread包裹起来并push到优先队列就好了，get_next则pop一个元素，如果不为None，则修改pass，并push回优先队列，remove_thread使用retain函数就可以解决，set_priority函数则找出线程并修改priority和stride。

4. 处理溢出，ucore教程中已经讲了如何处理溢出，于是我就照着处理了，但是rust对于溢出的运算会报错，所以要把pass的计算都用Warpping包裹起来。

```
thread 3
thread 2
thread 1
thread 4
thread 3
thread 4
thread 2
thread 3
thread 4
thread 4
thread 2
thread 3
thread 1
thread 4
thread 3
thread 4
thread 2
thread 3
thread 4
thread 4
thread 2
thread 3
thread 1
```

1. 分析：

   - 在 Stride Scheduling 算法下，如果一个线程进入了一段时间的等待（例如等待输入，此时它不会被运行），会发生什么？

     由于没有被运行，那么pass就不会增加，如果此时有别的线程被运行了，那么该线程相比其他线程pass值就比较小，唤醒后很容易长时间占据CPU，不过也有可能出现所有线程都在等待io这种情况。

   - 对于两个优先级分别为 9 和 1 的线程，连续 10 个时间片中，前者的运行次数一定更多吗？

     不一定，因为没有说明两者的pass值，也没有说明BigStride的大小，所以情况有很多种。

     而且按照ucore的教程，优先级是必须大于1的。

     经过实验发现，优先级为9和1的两个线程，调度算法只会让优先级为1的线程运行。

     如果是优先级为9和2的两个线程，一起运行，确实是优先级为9的线程运行次数多

- 你认为 Stride Scheduling 算法有什么不合理之处？可以怎样改进？

如果线程进入漫长的等待状态，长时间没有被运行，pass就会很小，唤醒后便会长时间占据CPU资源，容易导致其他线程饥饿。

可以在线程被唤醒后，修改它的pass值为所有可运行状态下线程的最小值，或是修改为最小值的一半，比例可以视需求而定。