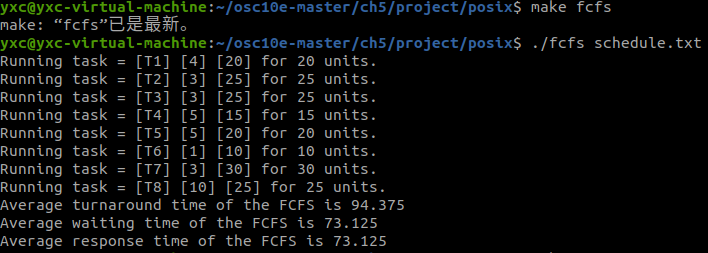
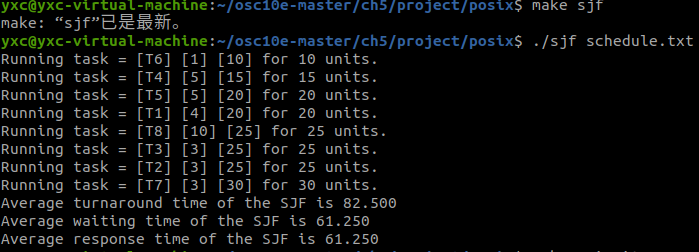
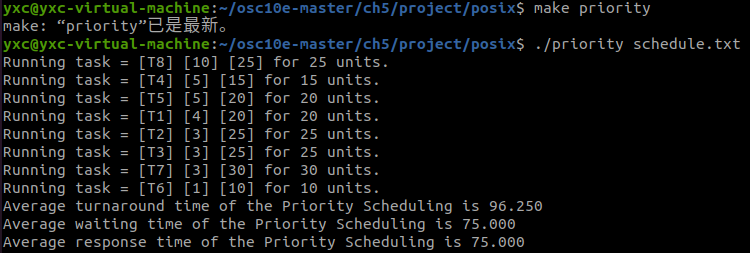
FCFS：首先定义add()函数，除了对于链表头head\_first\_in\_head有特殊处理外，其余任务在调用add()时，创建对应的task实例，并赋予唯一的tid，并将其加入链表中，同时number\_of\_tasks加1，用来记录任务数量。注意，该函数在任意调度算法中相同。

然后定义调度函数schedule\_single\_task()，FCFS调度算法中，该函数按顺序调度任务即可。需要注意的是，根据链表的插入规则，第一个任务出于队尾。需要从队尾遍历链表执行任务。

在过程中记录当前时间，计算总周转时间、总等待时间、总相应时间，在最后的schedule（）中计算平均值并输出。结果如图

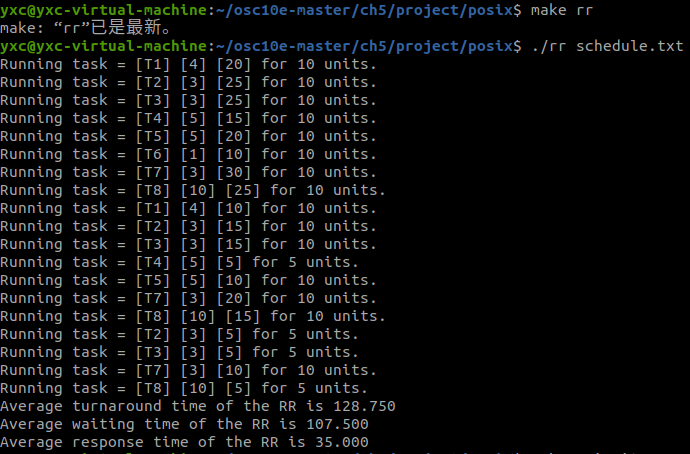
SJF：在SJF算法中，schedule\_single\_task()函数需要做出调整，每次均需遍历链表，找到时间最短的任务，将其执行并从链表中删除。如此，达成了短任务先执行的条件。需要在临界条件上（如当链表中元素只剩一个时）多加关注，以免造成错误。

在过程中记录当前时间，计算总周转时间、总等待时间、总相应时间，在最后的schedule（）中计算平均值并输出。结果如图

Priority Scheduling：在该算法中，我们先调用resort\_priority()函数对于链表根据优先级进行重排，然后再与FCFS算法相似地直接顺序调用任务，即可达成优先级高的任务先执行的条件。其余均相同。如图

RR：在RR调度算法中，首先是顺序调用任务，判断其执行时间burt是否大于轮转的时间切片QUANTUM，若大于，则令该任务先执行QUANTUM时长（即burst减QUANTUM），然后将该任务放到任务链表的队尾，另其可以在下一次被调用执行；若不大于，则可以将该任务执行完，并将其从链表中删除。即可达成轮转调度。

需要注意的是，为了计算总响应时间，我们需判断何时为该任务第一次被调用，因此我在task结构体中增加了first\_ask的整形变量，其等于1时是该任务第一次被调用，在调用后将其改为0，则之后无需计算其响应时间。结果如图



Priority\_RR：基于优先级的RR算法与RR算法的唯一区别在于进程的初始调度是根据优先级的。因此，只需要如Priority算法中一样，调用resort\_priority()函数对于链表根据优先级进行重排，然后再与RR算法相似的方式对于任务进行调度执行，其余相同。结果如图