实现方法

1. 创建队列：创建多个队列，每个队列对应一个不同的优先级。通常会有三个队列，但实际情况可以有更多。

2. 初始化任务：初始化任务，包括任务的名字、优先级、和估计的执行时间。

3. 任务入队：将任务添加到初始的队列中，通常是根据任务的初始优先级。

4. 调度：从队列中选择任务来执行，通常选择最高优先级的任务执行。任务执行一段时间后，如果没有完成，将其放回到队列的尾部，以便后续调度。

5. 降低优先级：如果任务在当前队列执行了一段时间仍未完成，其优先级可能会降低，将其移到较低优先级的队列中等待执行。

6. 任务完成检查：检查任务是否已经执行完成，如果完成则将其从队列中移除。

7. 循环调度：重复执行上述步骤，直到所有任务都完成。

源代码：

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  // 定义任务结构  typedef struct Task {  char name;  int priority;  int burst\_time;  } Task;  // 定义任务队列  typedef struct Queue {  Task\* tasks;  int front;  int rear;  int size;  } Queue;  // 初始化队列  Queue\* initQueue(int size) {  Queue\* queue = (Queue\*)malloc(sizeof(Queue));  queue->tasks = (Task\*)malloc(size \* sizeof(Task));  queue->front = 0;  queue->rear = -1;  queue->size = size;  return queue;  }  // 入队  void enqueue(Queue\* queue, Task task) {  if (queue->rear == queue->size - 1) {  printf("Queue is full\n");  } else {  queue->tasks[++queue->rear] = task;  }  }  // 出队  Task dequeue(Queue\* queue) {  if (queue->front > queue->rear) {  printf("Queue is empty\n");  Task null\_task = {'\0', -1, -1};  return null\_task;  }  return queue->tasks[queue->front++];  }  // 调度函数  void schedule(Queue\* queues[], int num\_queues, int time\_slice) {  int current\_time = 0;  int current\_queue = 0;  int quantum = time\_slice;  while (1) {  if (queues[current\_queue]->front <= queues[current\_queue]->rear) {  Task current\_task = dequeue(queues[current\_queue]);  printf("Executing task %c from queue %d for time slice %d\n", current\_task.name, current\_queue, quantum);  current\_time += quantum;  current\_task.burst\_time -= quantum;  if (current\_task.burst\_time > 0) {  if (current\_queue < num\_queues - 1) {  current\_queue++;  }  enqueue(queues[current\_queue], current\_task);  } else {  printf("Task %c completed\n", current\_task.name);  }  }  if (queues[current\_queue]->front > queues[current\_queue]->rear) {  if (current\_queue < num\_queues - 1) {  current\_queue++;  }  }  if (current\_queue == 0 && queues[0]->front > queues[0]->rear)  break;  }  }  int main() {  int num\_queues = 3;  Queue\* queues[3];  for (int i = 0; i < num\_queues; i++) {  queues[i] = initQueue(10);  }  Task tasks[] = {  {'A', 0, 5},  {'B', 1, 10},  {'C', 2, 3},  {'D', 0, 7},  {'E', 1, 2},  };  for (int i = 0; i < sizeof(tasks) / sizeof(Task); i++) {  enqueue(queues[0], tasks[i]);  }  int time\_slice = 2;  schedule(queues, num\_queues, time\_slice);  return 0;  } |

测试结果：

