```c
/* This documents uses code from scheduler_sfcfs
 * Parts code that was added to the original will be marked
 * citation for the original can be found below
 */
/* Implements a first-come, first-served scheduler.
 *
 * Created by Henry Walker, 27 September 2004
 * Last modified by Janet Davis, 25 September 2010
 * Revised by Jerod Weinman, 10 August 2012
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include "scheduler.h"

/* The ready queue */
job_queue_t ready;
job_queue_t priority_1;
job_queue_t priority_2;
job_queue_t priority_3;
job_queue_t priority_4;
job_queue_t priority_5;

/* Modification to original: new queues for different priorities */


/* Modification to original : array of priority queues */



/*
job_queue_t priorities [5]={ priority_1, priority_2, priority_3, priority_4, priori
ty_5 } ;

int i = 0 ;
for (i = 0 ; i < 5 ; i++)
  {
     priorities[i] = (job_queue_t *)malloc(sizeof(job_queue_t));
    priorities[i].first = NULL;
    priorities[i].last  = NULL;
  }
*/



/* Modification to original : Initialise ready and other queues  for all our other
queues */

/* Initializes the ready queue.  Call before any other functions. */

void ready_queue_init(void) {
  // initialise the ready queue
  ready.first = NULL;
  ready.last  = NULL;


  // initialise other priority queues
  priority_1.first = NULL;
  priority_1.last  = NULL;

  priority_2.first = NULL;
  priority_2.last  = NULL;

  priority_3.first = NULL;
  priority_3.last  = NULL;

  priority_4.first = NULL;
  priority_4.last  = NULL;


  priority_5.first = NULL;
  priority_5.last  = NULL;


}

/* Returns true or false, according to whether any jobs are waiting
 * in the ready queue.
 */
int ready_queue_empty(void) {
  return (ready.first == NULL);
}


/*
 * Modification to original : new method queue_priority
 *
 */

void queue_priority(job_queue_t *  priority,job_t* job) {


  job_queue_node_t* node
      = (job_queue_node_t *)malloc(sizeof(job_queue_node_t));

  if (!node) {
    perror("Unable to allocate job node");
    exit(EXIT_FAILURE);
  }

  /* copy event data to new node */
  node->job = job;

  /* insert node into ready queue*/
  node->next = NULL;
  if ((*priority).first == NULL) {
    (*priority).first = node;
    (*priority).last  = node;
  } else {
    (* priority).last->next  = node; /* add after current last */
    (* priority).last  = node;       /* make new node last */
  }
}

/* Adds the specified job to the ready queue.
 *
 * Preconditions:
 *   job != NULL
 * Postconditions:
 *   Creates a new node for the job
 *   job is inserted at the end of the queue
 */
void ready_queue_insert(job_t* job) {

  job_queue_t* priority = ( job_queue_t*) malloc(sizeof( job_queue_t));

 //get the priority of the incoming job
  int priority_val = job->priority;

  //determine which queue to add it to and then add it  to the queue

  switch (priority_val ){
  case 1:
    {
      priority = &priority_1;
      queue_priority( priority,job);
```

```
      break;

    }
  case 2:
    {
      priority = &priority_2;
      queue_priority( priority,job);
      break;
    }
  case 3:
    {
      priority = &priority_3;
      queue_priority( priority,job);
      break;
    }

  case 4:
    {
      priority = &priority_4;
      queue_priority( priority,job);
      break;
    }

  case 5:
    {
      priority = &priority_5;
      queue_priority( priority,job);
      break;
    }
  }
}

/* Removes and returns the job at the head of the ready queue.
 *
 * Postconditions: *   If ready_queue_empty(), returns NULL
 *   Otherwise, returns head job and frees the associated node
 */
job_t* ready_queue_select(void) {



  if( priority_5.first != NULL)
    {
      ready.first = priority_1.first;
      ready.last = priority_1.last;
    }
  else if( priority_4.first != NULL)
      {
        ready.first = priority_2.first;
        ready.last = priority_2.last;
      }
  else if( priority_3.first != NULL)
        {
          ready.first = priority_3.first;
          ready.last = priority_3.last;
        }
  else if( priority_2.first != NULL)
          {
            ready.first = priority_4.first;
            ready.last = priority_4.last;
          }
  else if( priority_1.first != NULL)
      {
        ready.first = priority_5.first;
        ready.last = priority_5.last;
      }
```

```
  job_t* job;
  job_queue_node_t* old_node;
  /* if no jobs are ready, return NULL */
  if (ready_queue_empty())
    return NULL;

  /* next job is at front of queue */
  job = ready.first->job;

  /* record node at front of queue */
  old_node = ready.first;
  ready.first = ready.first->next;

  /* check if queue is -now- empty */
  if (ready_queue_empty()) {
    ready.last = NULL; /* make last pointer consistent */
  }

  /* return old front of queue to memory pool */
  free(old_node);

  return job;
}
```