```c

/*
 *
 * Simulation_Run of A Single Server Queueing System
 *
 * Copyright (C) 2014 Terence D. Todd Hamilton, Ontario, CANADA,
 * todd@mcmaster.ca
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 3 of the License, or (at your option)
 * any later version.
 *
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
 * more details.
 *
 * You should have received a copy of the GNU General Public License along with
 * this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 */

/*******************************************************************************/

#include <math.h>
#include <stdio.h>
#include "main.h"
#include "packet_transmission.h"
#include "packet_arrival.h"

/*******************************************************************************/

/*
 * This function will schedule a packet arrival at a time given by
 * event_time. At that time the function "packet_arrival" (located in
 * packet_arrival.c) is executed. An object can be attached to the event and
 * can be recovered in packet_arrival.c.
 */

long int schedule_packet_arrival_event(Simulation_Run_Ptr simulation_run,
,                              double event_time)
{
  Event event;

  event.description = "Data Packet Arrival";
  event.function = data_packet_arrival_event;
  event.attachment = (void *) NULL;

  return simulation_run_schedule_event(simulation_run, event, event_time);
}

long int schedule_voice_packet_arrival_event(Simulation_Run_Ptr simulation_run,
,                              double event_time)
```

```c
{
  Event event;

  event.description = "Voice Packet Arrival";
  event.function = voice_packet_arrival_event;
  event.attachment = (void *) NULL;

  return simulation_run_schedule_event(simulation_run, event, event_time);
}

/*******************************************************************************/

/*
 * This is the event function which is executed when a packet arrival event
 * occurs. It creates a new packet object and places it in either the fifo
 * queue if the server is busy. Otherwise it starts the transmission of the
 * packet. It then schedules the next packet arrival event.
 */

void data_packet_arrival_event(Simulation_Run_Ptr simulation_run, void * ptr)
{
  Simulation_Run_Data_Ptr data;
  Packet_Ptr new_packet;

  data = (Simulation_Run_Data_Ptr) simulation_run_data(simulation_run);
  data->arrival_count++;

  new_packet = (Packet_Ptr) xmalloc(sizeof(Packet));
  new_packet->arrive_time = simulation_run_get_time(simulation_run);
  new_packet->service_time = get_packet_transmission_time();
  new_packet->status = WAITING;
  new_packet->source_id = DATA_PACKET;

  /*
   * Start transmission if the data link is free and there are no voice packets
queued.
   * Otherwise put the packet into the buffer.
   */
  if((server_state(data->link) == BUSY) ||
    (fifoqueue_size(data->voice_packet_buffer) > 0)) {
    fifoqueue_put(data->data_packet_buffer, (void*) new_packet);
  } else {
    start_transmission_on_link(simulation_run, new_packet, data->link);
  }

  /*
   * Schedule the next packet arrival. Independent, exponentially distributed
   * interarrival times gives us Poisson process arrivals.
   */

  schedule_packet_arrival_event(simulation_run,
                                simulation_run_get_time(simulation_run) +
                                exponential_generator((double)
1/DATA_PACKET_ARRIVAL_RATE));
}
```

```c

void voice_packet_arrival_event(Simulation_Run_Ptr simulation_run, void * ptr)
{
  Simulation_Run_Data_Ptr data;
  Packet_Ptr new_packet;

  data = (Simulation_Run_Data_Ptr) simulation_run_data(simulation_run);
  data->arrival_count++;

  new_packet = (Packet_Ptr) xmalloc(sizeof(Packet));
  new_packet->arrive_time = simulation_run_get_time(simulation_run);
  new_packet->service_time = get_voice_packet_transmission_time();
  new_packet->status = WAITING;
  new_packet->source_id = VOICE_PACKET;

  /*
   * Start transmission if the data link is free. Otherwise put the packet into
   * the buffer.
   */

  if(server_state(data->link) == BUSY) {
    fifoqueue_put(data->voice_packet_buffer, (void*) new_packet);
  } else {
    start_transmission_on_link(simulation_run, new_packet, data->link);
  }

  /*
   * Schedule the next packet arrival. Independent, exponentially distributed
   * interarrival times gives us Poisson process arrivals.
   */
  schedule_voice_packet_arrival_event(simulation_run,
                      simulation_run_get_time(simulation_run) +
                      (double) VOICE_PACKET_ARRIVAL_RATE);
}
```