```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)
```

```
55 {
56   Event event;
57
58   event.description = "Voice Packet Arrival";
59   event.function = voice_packet_arrival_event;
60   event.attachment = (void *) NULL;
61
62   return simulation_run_schedule_event(simulation_run, event, event_time);
63 }
64
65 /*************************************************************************/
66
67 /*
68  * This is the event function which is executed when a packet arrival event
69  * occurs. It creates a new packet object and places it in either the fifo
70  * queue if the server is busy. Otherwise it starts the transmission of the
71  * packet. It then schedules the next packet arrival event.
72  */
73
74 void data_packet_arrival_event(Simulation_Run_Ptr simulation_run, void * ptr)
75 {
76   Simulation_Run_Data_Ptr data;
77   Packet_Ptr new_packet;
78
79   data = (Simulation_Run_Data_Ptr) simulation_run_data(simulation_run);
80   data->arrival_count++;
81
82   new_packet = (Packet_Ptr) xmalloc(sizeof(Packet));
83   new_packet->arrive_time = simulation_run_get_time(simulation_run);
84   new_packet->service_time = get_packet_transmission_time();
85   new_packet->status = WAITING;
86   new_packet->source_id = DATA_PACKET;
87
88   /*
89    * Start transmission if the data link is free. Otherwise put the packet into
90    * the buffer.
91    */
92
93   if(server_state(data->link) == BUSY) {
94     fifoqueue_put(data->buffer, (void*) new_packet);
95   } else {
96     start_transmission_on_link(simulation_run, new_packet, data->link);
97   }
98
99   /*
100   * Schedule the next packet arrival. Independent, exponentially distributed
101   * interarrival times gives us Poisson process arrivals.
102   */
103
104  schedule_packet_arrival_event(simulation_run,
105                          simulation_run_get_time(simulation_run) +
106                          exponential_generator((double)
1/DATA_PACKET_ARRIVAL_RATE));
107 }
108
```

```
109
110 void voice_packet_arrival_event(Simulation_Run_Ptr simulation_run, void * ptr)
111 {
112   Simulation_Run_Data_Ptr data;
113   Packet_Ptr new_packet;
114
115   data = (Simulation_Run_Data_Ptr) simulation_run_data(simulation_run);
116   data->arrival_count++;
117
118   new_packet = (Packet_Ptr) xmalloc(sizeof(Packet));
119   new_packet->arrive_time = simulation_run_get_time(simulation_run);
120   new_packet->service_time = get_voice_packet_transmission_time();
121   new_packet->status = WAITING;
122   new_packet->source_id = VOICE_PACKET;
123
124   /*
125    * Start transmission if the data link is free. Otherwise put the packet into
126    * the buffer.
127    */
128
129   if(server_state(data->link) == BUSY) {
130     fifoqueue_put(data->buffer, (void*) new_packet);
131   } else {
132     start_transmission_on_link(simulation_run, new_packet, data->link);
133   }
134
135   /*
136    * Schedule the next packet arrival. Independent, exponentially distributed
137    * interarrival times gives us Poisson process arrivals.
138    */
139
140   schedule_voice_packet_arrival_event(simulation_run,
141                       simulation_run_get_time(simulation_run) +
142                       (double) VOICE_PACKET_ARRIVAL_RATE);
143 }
```