

```
1
2  /*
3   *
4   * Simulation_Run of A Single Server Queueing System
5   *
6   * Copyright (C) 2014 Terence D. Todd Hamilton, Ontario, CANADA,
7   * todd@mcmaster.ca
8   *
9   * This program is free software; you can redistribute it and/or modify it
10  * under the terms of the GNU General Public License as published by the Free
11  * Software Foundation; either version 3 of the License, or (at your option)
12  * any later version.
13  *
14  * This program is distributed in the hope that it will be useful, but WITHOUT
15  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
16  * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
17  * more details.
18  *
19  * You should have received a copy of the GNU General Public License along with
20  * this program. If not, see <http://www.gnu.org/licenses/>.
21  *
22  */
23
24  /*****
25
26  #include <math.h>
27  #include <stdio.h>
28  #include "main.h"
29  #include "packet_transmission.h"
30  #include "packet_arrival.h"
31
32  *****/
33
34  /*
35   * This function will schedule a packet arrival at a time given by
36   * event_time. At that time the function "packet_arrival" (located in
37   * packet_arrival.c) is executed. An object can be attached to the event and
38   * can be recovered in packet_arrival.c.
39   */
40
41  long int schedule_packet_arrival_event(Simulation_Run_Ptr simulation_run,
42                                         double event_time)
43  {
44      Event event;
45
46      event.description = "Data Packet Arrival";
47      event.function = data_packet_arrival_event;
48      event.attachment = (void *) NULL;
49
50      return simulation_run_schedule_event(simulation_run, event, event_time);
51  }
52
53  long int schedule_voice_packet_arrival_event(Simulation_Run_Ptr simulation_run,
54                                                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 and there are no voice packets
90      * queued.
91      * Otherwise put the packet into the buffer.
92      */
93     if((server_state(data->link) == BUSY) ||
94        (fifoqueue_size(data->voice_packet_buffer) > 0)) {
95         fifoqueue_put(data->data_packet_buffer, (void*) new_packet);
96     } else {
97         start_transmission_on_link(simulation_run, new_packet, data->link);
98     }
99
100    /*
101     * Schedule the next packet arrival. Independent, exponentially distributed
102     * interarrival times gives us Poisson process arrivals.
103     */
104
105    schedule_packet_arrival_event(simulation_run,
106                                   simulation_run_get_time(simulation_run) +
107                                   exponential_generator((double)
1081/DATA_PACKET_ARRIVAL_RATE));
109 }
```

```
108
109 void voice_packet_arrival_event(Simulation_Run_Ptr simulation_run, void * ptr)
110 {
111     Simulation_Run_Data_Ptr data;
112     Packet_Ptr new_packet;
113
114     data = (Simulation_Run_Data_Ptr) simulation_run_data(simulation_run);
115     data->arrival_count++;
116
117     new_packet = (Packet_Ptr) xmalloc(sizeof(Packet));
118     new_packet->arrive_time = simulation_run_get_time(simulation_run);
119     new_packet->service_time = get_voice_packet_transmission_time();
120     new_packet->status = WAITING;
121     new_packet->source_id = VOICE_PACKET;
122
123     /*
124      * Start transmission if the data link is free. Otherwise put the packet into
125      * the buffer.
126      */
127
128     if(server_state(data->link) == BUSY) {
129         fifoqueue_put(data->voice_packet_buffer, (void*) new_packet);
130     } else {
131         start_transmission_on_link(simulation_run, new_packet, data->link);
132     }
133
134     /*
135      * Schedule the next packet arrival. Independent, exponentially distributed
136      * interarrival times gives us Poisson process arrivals.
137      */
138     schedule_voice_packet_arrival_event(simulation_run,
139                                         simulation_run_get_time(simulation_run) +
140                                         (double) VOICE_PACKET_ARRIVAL_RATE);
141 }
```