Jason Petrov and Matthew Yaswinski

Dr. Kimm

CPSC 542

23 November 2021

Queen and Messengers

The task of this program is to use Promela in order to have one critical section handle multiple messages at once. There are several messengers that want to share a message with a queen, and each of these messages has a priority. Once all messages are created, they are all sent to the critical section at once in hopes of being read by the queen. The queen then looks for the message with the lowest priority (meaning a priority of 1 would be kept over a priority of 2) and consumes that message. Afterwards, the message is discarded and not read again by the queen. This process continues until all of the messages have been consumed.

The development began by trying to use embedded C using [2], [3], and [4] to read a file of messengers, each with a priority and a two-character message. We were able to read the message, but were unable to implement the data collected from C into Promela code. So, we decided to write the program purely in Promela.

The program begins with creating a few variables, including threadCounter that counts the number of messengers, bufferLength for the amount of messages to be stored, loopCounter for counting the amount of times the queen has read a message, and criticalSection for signaling that the critical section of the program is active. In addition, there is a msg structure that holds the ID of the message, the priority of the message, and the message itself. There is also an mtype for holding the available text that a message can have (from “AA” to “ZZ”) [5], an array of messages for storing each of them, and a channel called data for sending the messages to the queen [1].

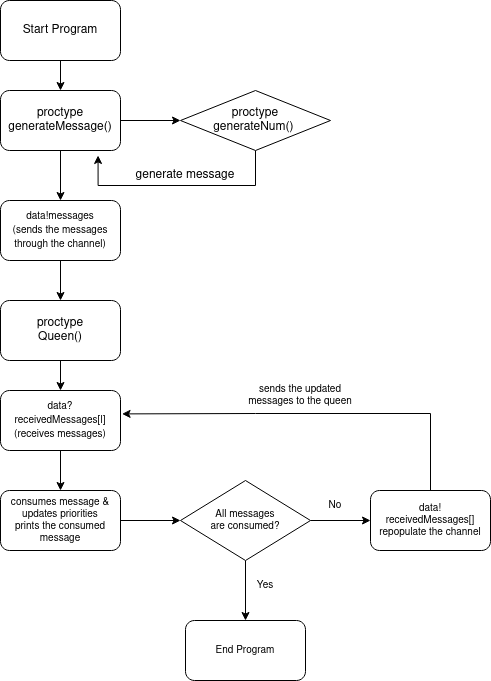
First, the program runs a method called generateMessage(), which runs another method called generateNum(). The generateNum() method randomly selects a number between 1 and 26, inclusively, and assigns it to a global byte called randNum. Back in generateMessage(), the ID of the message is set to the value of threadCounter (starting at 0), the priority is set to randNum, then generateNum() is run again to randomly select message text for a particular message. These steps are run within a loop that iterates from 0 to bufferLength – 1 (where bufferLength is set to 26), threadCounter is incremented for each iteration, and each of these messages is stored in the global array messages. Once these messages are created, they are sent to the queen through global channel data [7]. To do this, a loop is created that sends each of the elements of the array into the channel without any sorting being done in either the array or in the channel.

Once the messages are sent, the queen process checks if the loopCounter is equal to the bufferLength, and if it is not, it will check if the critical section is active, and if it not, then it will set the value for critical section to be equal to 1 to signal that the critical section is active. Then, the queen receives the message from the data channel and puts the messages into an array called receivedMessages [6]. It then goes through the entire array and compares each of the elements’ priorities. If an message with a lower priority is found in the array, it will reset the lowest priority to the priority of that message and set a variable indexRedMsg to the position in the array that the message is stored. After it finds the message with the lowest priority, the priorities of all messages that have not been read are decremented by 1 if their priorities are between 1 and 101 (which is the new priority set for read messages). After this, the critical section ends by setting the criticalSection variable to 0. In the remainder section, all messages, message ID’s, and their original priorities are printed. In addition, the message that the queen consumed is printed.

When all messages are printed and the queen consumes a message, the priority of the consumed message is set to 101. This prevents the program from reading the same exact message again, preventing starvation. In addition, the queen sends the messages with the updated priorities back into the data channel [7], and the loopCounter variable is incremented by 1. The queen processes continues until loopCounter is equal to 26, which means that all messages have been consumed.

The synchronization methods of the program include the use of the criticalSection variable and if statements, as well as updating priorities. The criticalSection variable being set to 1 and 0 allows for the program to be blocked once all messages have been sent to the queen. By using if statements to ensure that the critical section is not active (when criticalSection is equal to 0), this ensures that data cannot be modified once the critical section has begun, resulting in messages being consumed that should not be. In addition, updating priorities in several ways ensures that there is no starvation during the synchronization process. The message that has been consumed has its priority changed from its original value to 101, which ensures that the queen will not read it again due to its priority being higher than the maximum value that can be assigned. Also, changing the priorities of all unread messages allows for each message to eventually have a low enough priority to be read by the queen.

The flow chart below describes the steps of the program:



This chart shows that the program begins by generating all messages in a loop, then sends them in a channel. Once it does this, the queen reads the messages from the channel and stores them in an array. It finds the lowest priority, consumes the message with it, then checks if all of the messages are consumed. If they are not, the messages with updated priorities are sent to the channel again, and the process repeats until all of the messages are consumed.

In conclusion, the program that we have created allows for multiple messages to be sent to a queen. Once there, the program finds the message with the lowest priority to consume, changes its priority and lowers the priority of all other messages in order to prevent starvation, and it continues this until all messages have been consumed.

**References**

[1] *Promela reference -- chan(2)*. [Online]. Available: https://spinroot.com/spin/Man/chan.html. [Accessed: 23-Nov-2021].

[2] *Promela V4 reference -- c\_code(7)*. [Online]. Available: https://spinroot.com/spin/Man/c\_code.html. [Accessed: 23-Nov-2021].

[3] *Promela V4 reference -- c\_decl(7)*. [Online]. Available: https://spinroot.com/spin/Man/c\_decl.html. [Accessed: 23-Nov-2021].

[4] *Promela V4 reference -- c\_expr(7)*. [Online]. Available: https://spinroot.com/spin/Man/c\_expr.html. [Accessed: 23-Nov-2021].

[5] *Promela reference -- mtype(2)*. [Online]. Available: https://spinroot.com/spin/Man/mtype.html. [Accessed: 23-Nov-2021].

[6] *Promela reference -- receive(4)*. [Online]. Available: https://spinroot.com/spin/Man/receive.html. [Accessed: 23-Nov-2021].

[7] *Promela reference -- send(4)*. [Online]. Available: https://spinroot.com/spin/Man/send.html. [Accessed: 23-Nov-2021].