Documentation for exercise 3

In this exercise, we are supposed to implement the flow control in order not to overflow the Spread buffer. This is because the speed of various processes differs a lot. We need to ensure that even the slowest process won't overflow its buffer.

1. Strategy

   The strategy I used is to somehow measure the speed of various processes such that the faster processes will wait for the slower ones to catch up. Basically, I recorded the number of packets received from all the processes. Even the processes which doesn't need to send useful packets still need to send alert packets in order to let other processes know how fast the process is. I differentiate useful packets from alert packets by using message type supported by the Spread. 1 stands for the useful packet, 2 stands for the alert packets. I used an array data structure to record the number of packets received. The sign of the integer in the array is important. The positive integer stands for the number of useful packets received. The absolute value of the negative integer also stands for the number of packets, but it includes both the useful packets and alert packets. Whenever the process received an alert packet, it means all the useful packets have been received and the following packets are alert packets. If all the integers in the array are negative, it means all the processes have sent out the useful packets and the program could be safely terminated.

   In order not to overflow the buffers, at the same time, keep the buffers from empty all the time, I initiated the program by letting each process send WINDOW_SIZE number of packets. Whenever the process received a packet, it incremented the counter in the array and find out the process which has the least number of packets received by comparing the counter in the array. The faster process needs to wait for the slower process to send more packets until the faster one becomes the slower one. However, the slower process cannot send more packets immediately because there may still be a lot of packets in the buffer. The slower process is allowed to send more packets once it has received half of the number of packets it has already sent.

2. Performance

   The program is experimented with 8 machines where 6 of them send 160000 packets each, for a total of 960000 packets, and rest machines send 0 packets. The time cost is around 15 seconds.

3. How to run the program

   To run the program, use the command "./mcast <num_of_messages> <process_index> <num_of_processes>". Then input "j" to join the default group. Once all the processes are in the default group, the multicast will automatically start and terminate when the multicast is over.