Exercise 3 Design & Performance Discussion

Distributed System 600.437

Huabin Liu hliu76@jhu.edu
Qianyu Luo qluo3@jhu.edu

The mcast program mainly consists of two parts, one is to process membership management messages, the other is to process regular messages.

We designed a message struct as following:

```
int type: including NORMAL_TYPE and TERM_TYPE;
int process_index;
int message_index;
int random_number;
char payload[1200];
```

At the beginning of program, we call SP_connect_timeout and SP_join functions to ensure that all the members join in the group properly. And call E_attach_fd function to handle the events.

In the Read_message function, after receiving a message by calling SP_receive(), we analyze the message by calling Is_regular_mess and Is_membership_mess.

Membership messages

This part is similar to the sample, based on the service_type, we can determinate what kinds of membership message it is. After receiving a regular membership message, we print all the members' information. Also after receiving a joining message, we print the name of the new member.

When memb_info.gid.id[2] equals the num_of_processes that we entered, we know that all the processes have already joined the group. And then we can start transmission.

Regular messages

Regular messages include normal type and termination type messages. At the very beginning, every process can send out a burst of message if it has some messages to send. Then, in order to control the sending speed, we set a rule that a process can send a new burst of message only after it receives all the messages it sent out last round. After sending out all the applicable messages, a process multicast a term_type message to let other processes know its state. When a process receives a term_type message, it increments a local counter. If a process receives term_type messages from every process in the group, it's time to terminate.

Performance discussion

Transmission Time						
	1	2	3	4	5	Avg.
Time(s)	12.926362	13.082455	12.418333	15.978432	12.609381	13.402992

Parameters:
MESS_PER_ROUND: 100
WINDOW_LEN: 1000

We tested out program by setting number of process to 8. Let six of them send 100,000 messages and two of them send 0 message. The average transmission time is 13.40s. Due to some unstable factors on ugrad machines, the differences in time of the tests are 3~4 seconds. We find that the parameters can also influence performance a lot. When we decrease MESS_PER_ROUND to 50, the transmission time slows down about 4s. If MESS_PER_ROUND is too large, a process may send too much messages in one round and may lead to instability. As for WINDOW_LEN, we know that write messages to file can be expensive, thus if we make window longer, process needs less time cleaning window during transmission. If window is really short, the transfer speed is low since process always need to wait for writing. If we change our window length from 1000 to 500, the average transfer time slows down approximate 2s. We don’t want to set window length extremely long either. As a result, we choose 100 messages per round and set window length to 1000.