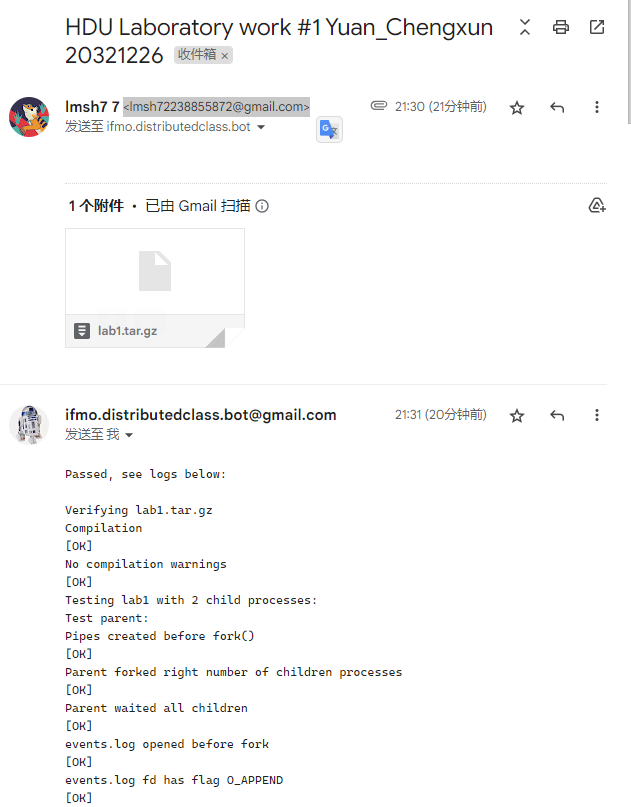
Laboratory work #1

Introduction to communication framework

Student: Yuan Chengxun  
Student HDU ID: 20321226

Mail:<lmsh72238855872@gmail.com

Screenshot or copy with mail with “Passed”:



Control questions:

1. What is distributed system? Give your understanding of such systems.

Answer: **A distributed system is a network of interconnected computers that collaborate to perform tasks, share resources, and provide reliability, scalability, and fault tolerance through decentralized processing and communication.**

**I am familiar with the CAP theorem of distributed systems, which underscores the trade-offs between Consistency, Availability, and Partition Tolerance. This fundamental concept guides the design and operation of distributed systems, helping to balance these three key attributes to meet specific application requirements and resilience needs.**

2. Why starting synchronization between processes is important in this laboratory work?

Answer: **Starting synchronization between processes is important in this laboratory work because it ensures that all child processes have completed their work before the parent process terminates. This is achieved through the execution of three subsequent phases in each child process, which includes a procedure of synchronization with all other processes in the distributed system, a "useful" job of the child process, and a procedure of synchronization processes before their completion.**

3. Execute your program several times with 9 child processes. Why order of logs can be different?

Answer: **Because the order in which the processes are scheduled to run is not deterministic. This means that the operating system can schedule the processes in any order it chooses, which can result in different orders of logs being printed to the console .**

Source code:

1. #include <stdio.h>
2. #include <unistd.h>
3. #include <string.h>
4. #include <sys/types.h>
6. #include "message.h"
7. #include "log.h"
8. #include "process.h"
10. **void** parent\_work(**int** count\_nodes)
11. {
12. // printf("Parent process started , %d children\n", count\_nodes - 1);
13. /\* STUDENT IMPLEMENTATION STARTED \*/
14. **for** (**int** i = 1; i < count\_nodes; i++)
15. {
16. Message msg;
17. receive(i, &msg);
18. **if** (msg.s\_header.s\_magic == MESSAGE\_MAGIC && msg.s\_header.s\_type == STARTED)
19. {
20. **continue**;
21. }
22. }
23. **for** (**int** i = 1; i < count\_nodes; i++)
24. {
25. Message msg;
26. receive(i, &msg);
27. **if** (msg.s\_header.s\_magic == MESSAGE\_MAGIC && msg.s\_header.s\_type == DONE)
28. {
29. **continue**;
30. }
31. }
32. }
34. **void** child\_work(**struct** child\_arguments args)
35. {
36. /\* Child arguments \*/
37. local\_id self\_id = args.self\_id;
38. **int** count\_nodes = args.count\_nodes;
40. /\* System process identifiers used for logs \*/
41. pid\_t self\_pid = getpid();
42. pid\_t parent\_pid = getppid();
44. /\* STUDENT IMPLEMENTATION STARTED \*/
45. **char** buf[BUF\_SIZE];
46. Message msg;
47. snprintf(buf, BUF\_SIZE, log\_started\_fmt, 0, self\_id, self\_pid, parent\_pid, 0);
48. fill\_message(&msg, STARTED, 0, buf, strlen(buf));
49. send\_multicast(&msg);
50. shared\_logger(buf);
52. /\* 接收消息 \*/
53. Message recv\_msg;
54. **int** cnt = 0;
55. **for** (**int** i = 1; i < count\_nodes; i++)
56. {
57. **if** (i == self\_id)
58. **continue**;
59. receive(i, &recv\_msg);
60. **if** (recv\_msg.s\_header.s\_magic == MESSAGE\_MAGIC && recv\_msg.s\_header.s\_type == STARTED)
61. {
62. cnt++;
63. }
64. }
65. **if** (cnt == count\_nodes - 2)
66. {
67. snprintf(buf, BUF\_SIZE, log\_received\_all\_started\_fmt, 0, self\_id);
68. shared\_logger(buf);
69. }
71. /\* 发送DONE消息 \*/
72. snprintf(buf, BUF\_SIZE, log\_done\_fmt, 0, self\_id, 0);
73. fill\_message(&msg, DONE, 0, buf, strlen(buf));
74. send\_multicast(&msg);
75. shared\_logger(buf);
77. /\* 接收DONE消息 \*/
78. cnt = 0;
79. **for** (**int** i = 1; i < count\_nodes; i++)
80. {
81. **if** (i == self\_id)
82. **continue**;
83. receive(i, &recv\_msg);
84. **if** (recv\_msg.s\_header.s\_magic == MESSAGE\_MAGIC && recv\_msg.s\_header.s\_type == DONE)
85. {
86. cnt++;
87. }
88. }
89. **if** (cnt == count\_nodes - 2)
90. {
91. snprintf(buf, BUF\_SIZE, log\_received\_all\_done\_fmt, 0, self\_id);
92. shared\_logger(buf);
93. }
94. }