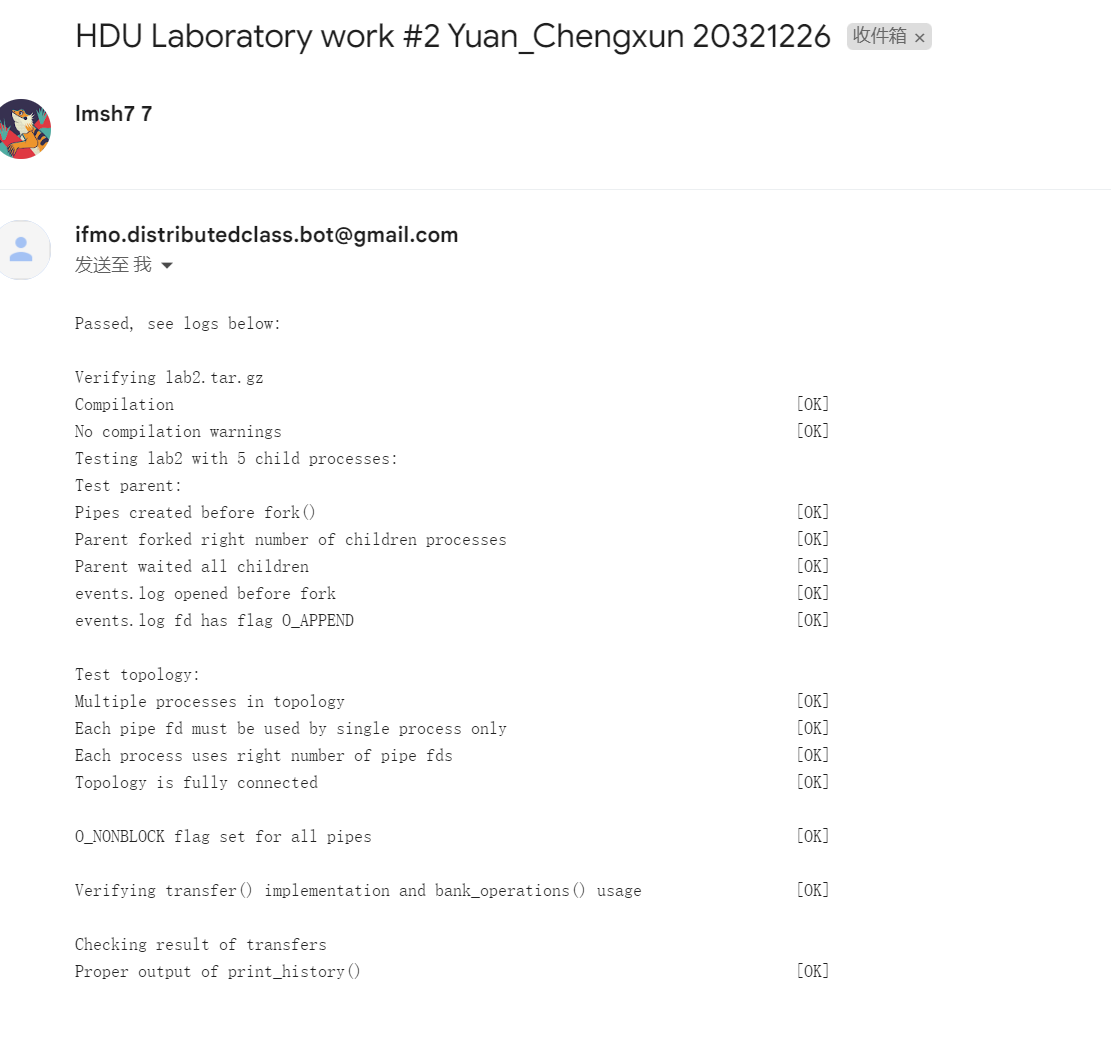
Laboratory work #2

Distributed banking system

Student: Yuan Chengxun  
Student HDU ID: 20321226

Mail: lmsh72238855872@gmail.com

Screenshot or copy with mail with “Passed”:



Control questions:

1. Transfer request. Is it synchronous or asynchronous in our model from point of client? Why?  
What is difference between synchronous and asynchronous operations?

Answer: I think it is synchronous, because the parent always needs to wait for each ack before starting the next transfer. The difference between synchronous and asynchronous depends on whether the client's request will be explicitly blocked by the server.

2. What will happened, if transfer operations isn’t instantaneous?  
Can we calculate total amount of money at each moment of time? Why?

Answer: If transfer operations are not instantaneous, it means there's a delay between initiating the transfer and its completion. During this delay, the total amount of money might not reflect the pending transfers. The system doesn't handle pending transfers correctly, it could lead to issues like overdrawing accounts or failing to transfer funds when there's a lack of sufficient balance.

3. Make an experiment. You can find in header banking.h function get\_physical\_time\_skew(). Use it instead of get\_physical\_time(). This function provides non-synchronized clocks. Why result of execution is different?  
Answer should contain output table.

Answer: Asynchronous clocks will cause inconsistent sequence of events and deviations in amounts due to delays.

Normal:

文本, 表格

描述已自动生成

Skew:

文本

描述已自动生成

Source code:

1. #include <stdio.h>
2. #include <unistd.h>
3. #include <string.h>
4. #include <sys/types.h>
5. #include "labs\_headers/message.h"
6. #include "labs\_headers/log.h"
7. #include "labs\_headers/process.h"
8. #include "labs\_headers/banking.h"
9. *//export LD\_LIBRARY\_PATH="$LD\_LIBRARY\_PATH:/workspaces/HDU-CS-2023-Distributed-computing/task\_lab2"*
10. *//LD\_PRELOAD=./libdistributedmodel.so ./lab -l 2 -p 3 B1 B2 B3*
11. void update\_history(BalanceHistory \*history, timestamp\_t time, balance\_t amount)
12. {
13. int last\_time = history->s\_history[history->s\_history\_len - 1].s\_time;
14. int last\_balance = history->s\_history[history->s\_history\_len - 1].s\_balance;
15. for (int i = last\_time + 1; i < time; i++)
16. {
17. history->s\_history[history->s\_history\_len].s\_time = i;
18. history->s\_history[history->s\_history\_len].s\_balance = last\_balance;
19. history->s\_history[history->s\_history\_len].s\_balance\_pending\_in = 0;
20. history->s\_history\_len++;
21. }
22. history->s\_history[history->s\_history\_len].s\_balance = amount;
23. history->s\_history[history->s\_history\_len].s\_time = time;
24. history->s\_history[history->s\_history\_len].s\_balance\_pending\_in = 0;
25. history->s\_history\_len++;
26. }
27. balance\_t now\_balance(BalanceHistory \*history)
28. {
29. return history->s\_history[history->s\_history\_len - 1].s\_balance;
30. }
31. void parent\_work(int count\_nodes)
32. {
33. AllHistory all\_history;
34. all\_history.s\_history\_len = count\_nodes - 1;
35. */\* STUDENT IMPLEMENTATION STARTED \*/*
36. */\* Implement starting synchronization \*/*
37. for (int i = 1; i < count\_nodes; i++)
38. {
39. Message msg;
40. receive(i, &msg);
41. if (msg.s\_header.s\_magic == MESSAGE\_MAGIC && msg.s\_header.s\_type == STARTED)
42. {
43. continue;
44. }
45. }
46. */\* Useful work \*/*
47. bank\_operations(count\_nodes - 1);
48. */\* Implement finishing synchronization and collecting AllHistory \*/*
49. */\* 广播 STOP 消息\*/*
50. Message msg;
51. fill\_message(&msg, STOP, get\_physical\_time(), NULL, 0);
52. send\_multicast(&msg);
53. */\* 收到所有 DONE 消息 \*/*
54. for (int i = 1; i < count\_nodes; i++)
55. {
56. Message msg;
57. receive(i, &msg);
58. if (msg.s\_header.s\_magic == MESSAGE\_MAGIC && msg.s\_header.s\_type == DONE)
59. {
60. continue;
61. }
62. }
63. */\* 收集所有 BalanceHistory \*/*
64. for (int i = 1; i < count\_nodes; i++)
65. {
66. Message msg;
67. receive(i, &msg);
68. if (msg.s\_header.s\_magic == MESSAGE\_MAGIC && msg.s\_header.s\_type == BALANCE\_HISTORY)
69. {
70. BalanceHistory \*history = (BalanceHistory \*)msg.s\_payload;
71. memcpy(&all\_history.s\_history[i - 1], history,msg.s\_header.s\_payload\_len);
72. }
73. }
74. print\_history(&all\_history);
75. return;
76. }
77. void child\_work(struct child\_arguments args)
78. {
79. */\* Child arguments \*/*
80. local\_id self\_id = args.self\_id;
81. int count\_nodes = args.count\_nodes;
82. uint8\_t balance = args.balance;
83. */\* BalanceHistory initialization \*/*
84. BalanceHistory history;
85. history.s\_history\_len = 1;
86. history.s\_id = self\_id;
87. memset(history.s\_history, 0, sizeof(history.s\_history));
88. history.s\_history[0].s\_balance = balance;
89. for (int i = 0; i < MAX\_T; ++i) {
90. history.s\_history[i].s\_time = i;
91. }
92. */\* System process identifiers used for logs \*/*
93. pid\_t self\_pid = getpid();
94. pid\_t parent\_pid = getppid();
95. */\* STUDENT IMPLEMENTATION STARTED \*/*
96. char buf[BUF\_SIZE];
97. Message msg;
98. timestamp\_t current = get\_physical\_time(); *// 是 lamport 逻辑时间*
99. snprintf(buf, BUF\_SIZE, log\_started\_fmt, current, self\_id, self\_pid, parent\_pid, balance);
100. fill\_message(&msg, STARTED, current, buf, strlen(buf));
101. send\_multicast(&msg);
102. shared\_logger(buf);
103. Message recv\_msg;
104. int cnt = 0;
105. for (int i = 1; i < count\_nodes; i++) {
106. if (i == self\_id)
107. continue;
108. receive(i, &recv\_msg);
109. if (recv\_msg.s\_header.s\_magic == MESSAGE\_MAGIC && recv\_msg.s\_header.s\_type == STARTED) {
110. cnt++;
111. }
112. }
113. if (cnt == count\_nodes - 2) {
114. current = get\_physical\_time();
115. snprintf(buf, BUF\_SIZE, log\_received\_all\_started\_fmt, current, self\_id);
116. shared\_logger(buf);
117. }
118. */\* 阶段 2 等待并处理任何 TRANSFER 和 STOP 消息 \*/*
119. cnt = 0;
120. while (true) {
121. Message req\_msg;
122. receive\_any(&req\_msg);
123. if (req\_msg.s\_header.s\_magic == MESSAGE\_MAGIC && req\_msg.s\_header.s\_type == TRANSFER)
124. {
125. TransferOrder \*order = (TransferOrder \*)req\_msg.s\_payload;
126. if (order->s\_src == self\_id) {
127. current = get\_physical\_time();
128. update\_history(&history, current, now\_balance(&history) - order->s\_amount);
129. snprintf(buf, BUF\_SIZE, log\_transfer\_out\_fmt, current, self\_id, order->s\_amount, order->s\_dst);
130. shared\_logger(buf);
131. Message response\_msg;
132. fill\_message(&response\_msg, TRANSFER, current, order, strlen((const char\*)order));
133. send(order->s\_dst, &response\_msg);
134. } else if (order->s\_dst == self\_id) {
135. current = get\_physical\_time();
136. update\_history(&history, current, now\_balance(&history) + order->s\_amount);
137. snprintf(buf, BUF\_SIZE, log\_transfer\_in\_fmt, current, self\_id, order->s\_amount, order->s\_src);
138. shared\_logger(buf);
139. Message response\_msg;
140. fill\_message(&response\_msg, ACK, current, NULL, 0);
141. send(0, &response\_msg);
142. }
143. } else if (req\_msg.s\_header.s\_magic == MESSAGE\_MAGIC && req\_msg.s\_header.s\_type == STOP) {
144. current = get\_physical\_time();
145. */\* 阶段 3 发送 DONE 消息 \*/*
146. snprintf(buf, BUF\_SIZE, log\_done\_fmt, current, self\_id, now\_balance(&history));
147. shared\_logger(buf);
148. Message response\_msg;
149. fill\_message(&response\_msg, DONE, current, buf, strlen(buf));
150. send\_multicast(&response\_msg);
151. break;
152. } else if (req\_msg.s\_header.s\_magic == MESSAGE\_MAGIC && req\_msg.s\_header.s\_type == DONE) {
153. cnt++;
154. }
155. }
156. */\* 接收 DONE 消息 \*/*
157. while (cnt != count\_nodes - 2)
158. {
159. Message msg;
160. receive\_any(&msg);
161. if (msg.s\_header.s\_magic == MESSAGE\_MAGIC && msg.s\_header.s\_type == DONE)
162. {
163. cnt++;
164. }
165. }
166. if (cnt == count\_nodes - 2)
167. {
168. current = get\_physical\_time();
169. snprintf(buf, BUF\_SIZE, log\_received\_all\_done\_fmt, current, self\_id);
170. shared\_logger(buf);
171. }
172. */\* 发送 BALANCE\_HISTORY 消息 \*/*
173. Message history\_msg;
174. current = get\_physical\_time();
175. fill\_message(&history\_msg, BALANCE\_HISTORY, current, &history, sizeof(history));
176. send(0, &history\_msg);
177. }
178. void transfer(local\_id src, local\_id dst,
179. balance\_t amount)
180. {
181. TransferOrder order = {src, dst, amount};
182. */\* STUDENT IMPLEMENTATION STARTED \*/*
183. */\* 发送 TRANSFER 消息 \*/*
184. Message msg;
185. fill\_message(&msg, TRANSFER, get\_physical\_time(), &order, sizeof(order));
186. send(src, &msg);
187. */\* 接受 dst 的 ACK 消息\*/*
188. receive(dst, &msg);
189. if (msg.s\_header.s\_magic == MESSAGE\_MAGIC && msg.s\_header.s\_type == ACK)
190. {
191. return;
192. }
193. }
194. */\* STUDENTS SHOULD NOT CHANGE THIS FUNCTION \*/*
195. \_\_attribute\_\_((weak)) void bank\_operations(local\_id max\_id)
196. {
197. for (int i = 1; i < max\_id; ++i) {
198. transfer(i, i + 1, i);
199. }
200. if (max\_id > 1) {
201. transfer(max\_id, 1, 1);
202. }
203. }