Laboratory work #1

Introduction to communication framework

111

Student: /\* use only Latin symbols \*/  
Student HDU ID: 242320031

Mail: 1909899341@qq.com

/\* mail, which was used to send source code to the bot \*/

Screenshot or copy with mail with “Passed”:



/\* screenshot with Passed result \*/

Control questions:

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

Answer:

A distributed system is a network of independent computers that work together to achieve a common goal. These systems allow components located on different machines to communicate and coordinate their actions, making them appear to users as a single, cohesive system.

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

Answer:

When multiple processes access shared resources, synchronization helps avoid conflicts and ensures that data remains consistent. Without synchronization, simultaneous updates might lead to data corruption or loss.

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

Answer:

Child processes run concurrently, which means they can be executed simultaneously or alternately. Depending on the schedule of the operating system, different processes may complete execution at different times, resulting in changes in the order of logs.

Source code:

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

// #include "/home/xx/Desktop/xx/task\_lab1/task\_lab1/labs\_headers/message.h"

// #include "/home/xx/Desktop/xx/task\_lab1/task\_lab1/labs\_headers/log.h"

// #include "/home/xx/Desktop/xx/task\_lab1/task\_lab1/labs\_headers/process.h"

#include "message.h"

#include "log.h"

#include "process.h"

void parent\_work(int count\_nodes)

{

Message msg;

for(int i=1;i<count\_nodes;i++){

while(true){

receive(i,&msg);

if(msg.s\_header.s\_magic == MESSAGE\_MAGIC && msg.s\_header.s\_type == STARTED){

break;

}else{

continue;

}

}

}

for(int i=1;i<count\_nodes;i++){

while(true){

receive(i,&msg);

if(msg.s\_header.s\_magic == MESSAGE\_MAGIC && msg.s\_header.s\_type == DONE){

break;

}else{

continue;

}

}

}

}

void child\_work(struct child\_arguments args)

{

local\_id self\_id = args.self\_id;

int count\_nodes = args.count\_nodes;

pid\_t self\_pid = getpid();

pid\_t parent\_pid = getppid();

char buf[BUF\_SIZE];

Message msg;

// Phase 1

snprintf(buf,BUF\_SIZE,log\_started\_fmt,0,self\_id,self\_pid,parent\_pid,0);

fill\_message(&msg,STARTED,0,buf,strlen(buf));

send\_multicast(&msg);

shared\_logger(buf);

for(int i=1;i<count\_nodes;i++){

if(i==self\_id) continue;

while(true){

receive(i,&msg);

if(msg.s\_header.s\_magic == MESSAGE\_MAGIC && msg.s\_header.s\_type == STARTED){

break;

}else{

continue;

}

}

}

printf(log\_received\_all\_started\_fmt,0,self\_id);

// Phase 2

// Phase 3

snprintf(buf,BUF\_SIZE,log\_done\_fmt,0,self\_id,0);

fill\_message(&msg,DONE,0,buf,strlen(buf));

send\_multicast(&msg);

shared\_logger(buf);

for(int i=1;i<count\_nodes;i++){

if(i==self\_id) continue;

while(true){

receive(i,&msg);

if(msg.s\_header.s\_magic == MESSAGE\_MAGIC && msg.s\_header.s\_type == DONE){

break;

}else{

continue;

}

}

}

printf(log\_received\_all\_done\_fmt,0,self\_id);

}

/\*

export LD\_LIBRARY\_PATH="$LD\_LIBRARY\_PATH:./libdistributedmodel"

LD\_PRELOAD=./libdistributedmodel.so ./lab -l 1 -p 3

\*/