1. 개요

Socket을 이용하여 Sever 프로세스와 Client 프로세스 사이의 IPC를 구현, 메시지 큐를 이용하여 Server 프로세스 내 관리 Thread와 Eval Thread 사이의 ITC를 구현하여 총 세 개의 Client 사이에서 동작하는 가위바위보 게임을 구현하였습니다.

업로드해주신 코드에서 메시지 큐 부분에서 오류 발생한다고 알려주셔서 하나의 메시지 큐를 이용하는 프로그램을 구현한다면 문제를 해결할 수 있지 않을까 생각하여 Client들의 입력을 하나의 큐에 모아두었다가 중복되는 입력이 존재한다면 탈락하는 방식으로 눈치 게임을 만들어보자 하였습니다.

하지만 관리 Thread의 끝 부분, Eval Thread의 동작 결과를 메시지 큐로 전달받아 다시 소켓을 통해 Client에게 전달하는 부분에서 하나의 메시지 큐를 사용하자 메시지 큐를 받아오는 부분에서 쓰레드가 응답하지 않는 문제가 생겼습니다. 검색을 통해 알아본 결과, 메시지 큐에 메시지를 입력할 때 long 타입 변수인 메시지 타입을 설정하고 메시지 타입을 이용하여 큐에서 원하는 메시지를 선택하여 꺼내 사용할 수 있지만 아직 메시지 큐 구현하는 데 익숙하지 않아 큐가 가득 차거나 순서가 꼬이는 등의 실수를 하여 문제가 발생하였구나 짐작하였습니다.

최대한 수정해보려 노력하였지만 시간이 부족하여 업로드해주신 코드에서 Client를 하나 더 연결할 수 있도록 수정하는 방향으로 코드를 작성하였고, 이전 코드를 작성하면서 알게 된 부분은 고찰에 함께 작성하겠습니다.

1. 프로그램 구조 설명
   1. 함수에 대한 설명
2. game\_server.c

main(int argc, char\* argv[]);

* Client 프로세스에서 들어온 요청을 확인하여 소켓을 연결하고, Server 프로세스가 여러 Client와 연결될 수 있도록 관리 Thread를 생성하여 소켓을 할당함

void\* serverthread(void\* parm);

* 소켓을 이용해 Client와 데이터를 교환함
* Client에게 입력 받은 데이터를 메시지 큐를 이용해 Eval Thread로 전달함
* Client의 수에 따른 코드의 동작을 위해 변수에 기록함. 이때 Client의 수를 저장하는 visits 변수는 관리 Thread들 간의 공유 자원이고 쓰기 작업이 이루어지기 때문에 동기화 문제가 발생할 수 있음. 따라서 mutex를 이용함.

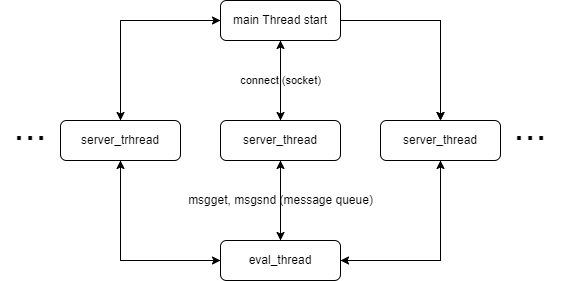
void\* evalthread();

* 메시지 큐를 통해 입력받은 값을 처리하여 메시지 큐를 통해 결과를 반환함

1. game\_client.c

main(int argc, char \*argv[]);

* 소켓을 통해 서버에서 진행 중인 게임 상황을 전달받고 입력 값을 전달함
  1. 다이어그램



1. 실행 결과
2. 최초 실행, 정상 동작

텍스트, 스크린샷, 모니터, 검은색이(가) 표시된 사진

자동 생성된 설명

1. Play again? (y), same value

텍스트이(가) 표시된 사진

자동 생성된 설명

1. Play again? (n)

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 이전 코드에서 오류가 발생하는 부분

텍스트이(가) 표시된 사진

자동 생성된 설명

관리 Thread들이 모두 하나의 메시지 큐에 값을 입력하도록 작성하였음. 여러 차례 수정을 통해 지금은 msgrcv(msqid, buffer, MAXTEXT, 0, MSG\_NOERROR) 작성되어 있으나, 이전에는 메시지 큐에 구조체 형태로 데이터를 저장하도록 하였고, long 타입 변수인 mtype으로 tid를 전달하도록 하여 각각의 Thread가 자신의 tid와 맞는 데이터를 메시지 큐에서 꺼내어 사용하면 되기 때문에 Eval Thread에서 처리한 결과값을 메시지 큐에 차례로 입력해 주었습니다. 하지만 예상과는 다르게 메시지 큐에서 데이터를 찾지 못해 입력 대기 상태에 머물러 있어 쓰레드가 더 이상 동작하지 않고 멈춘 것 처럼 보였습니다.

1. 고찰

눈치 게임을 구현하기 위해 하나의 메시지 큐에 각 관리 Thread에서 입력한 값들을 저장하고, Eval Thread에서 값을 차례로 꺼내어 현재 입력받아야 하는 값, 이전에 이미 사용된 적이 있었는지 여부 등을 판단하여 프로그램을 구현하고자 하였습니다.

메시지 큐에 전달하는 데이터 형식인 msgbuf 구조체를 정의하였고, mtype으로 관리 Thread의 TID를 입력하여 Eval Thread에서 어떤 쓰레드에서 입력한 값인가 확인하여 PASS 여부를 확인하여 다시 메시지 큐에 PASS 메시지와 함께 입력하여 관리 Thread에서 메시지를 수신하고 전달받은 결과값을 출력하도록 작성하였습니다.

하지만 관리 쓰레드에서 입력했던 TID 값을 mtype으로 가지는 메시지를 찾지 못해 그 부분에서 쓰레드가 진행되지 않는 문제가 발생하였습니다. 메시지에 따로 번호를 지정해 주기 때문에 우선순위 큐처럼 원하는 메시지부터 뽑아 사용할 수 있을 것이라 생각했지만, 검색을 통해 찾아본 결과 제가 짐작한 것과 다른 방식으로 동작하는 것을 알게 되었습니다.

예시 코드를 수정하여 작성한 현재 코드에서 연결된 Client의 개수를 저장하는 visits 변수는 critical section으로 감싼 후에 값을 수정합니다. 여러 쓰레드에 의해 수정되는 값이므로 공유 메모리이고, 읽기 작업만 수행되는 것이 아니라 쓰기 작업이 수행되므로 동시에 Client가 접속하는 경우에 동기화 문제가 발생할 수 있습니다. 따라서 mutex를 이용하여 변수를 잠근 후에 값을 수정합니다.

값을 세기 때문에 이런 동작도 counting semaphore에 해당하는지 궁금한 점이 생겨 좀 더 확인해본 결과 어떤 공유 자원의 접근을 제한하기 위해서 사용하는 것이 아니기 때문에 semaphore에 해당하지는 않는다고 판단하였습니다.

visits 변수의 접근을 제한하기 위해 사용한 것은 mutex입니다.

데이터를 처리하는 Server 프로세스와 데이터를 입력, 사용자에게 값을 출력하는 Client 프로세스 사이 데이터를 주고받기 위해 IPC를 구현해야 하는데, 이때 소켓이 사용되었습니다.

서로 간에 검증할 수 있도록 포트를 열어두고 그것을 통해 데이터를 송수신하는 방식입니다.

관리 Thread와 Eval Thread 사이에 값을 주고 받기 위해 메시지 큐가 사용되었습니다. 메시지 큐는 커널 영역에 생성되어 따로 동작하기 때문에 쓰레드 간의 데이터를 전달하는데 사용할 수 있습니다. 큐이므로 입력한 값을 순서대로 저장해두었다가 앞에서부터 순서대로 꺼내어 사용할 수 있습니다. 메시지 타입 등을 설정하여 특정 쓰레드, 프로세스 간에 값을 꺼내어 사용하는 것도 가능합니다.

같은 scope 안에 관리 Thread와 Eval Thread가 존재하기 때문에 전역 변수와 뮤텍스를 잘 사용하여 값을 전달하는 것도 가능하지 않을까 짐작해 보았습니다.

1. 프로그램 소스 코드

game\_server.c

#include <sys/types.h> //socket, 정수형 확장 자료형, pthread 관련 자료형을 포함하는 헤더

#include <sys/socket.h> //socket관련 함수, 자료형들을 사용하기 위한 헤더

#include <netinet/in.h> //network관련 함수, 자료형을 사용하기 위한 헤더

#include <netdb.h> //network관련 함수, 자료형을 사용하기 위한 헤더

#include <pthread.h> //POSIX 쓰레드 관련 함수

#include <stdio.h> //표준입출력

#include <string.h> //문자열 처리

#include <sys/ipc.h> //msgq, semaphore등 ipc를 사용하기 위한 헤더

#include <sys/msg.h> //msg queue 사용을 위한 확장 자료형을 포함하는 헤더

void\* serverthread(void\* parm); /\* thread function prototype \*/

void\* evalthread(); /\* thread function prototype \*/

pthread\_mutex\_t mut;

#define PROTOPORT 5193 /\* default protocol port number \*/

#define QLEN 6 /\* size of request queue \*/

#define MAXTEXT 100

int visits = 0; /\* counts client connections \*/

int evalgame(char, char, char);

main(int argc, char\* argv[])

{

struct hostent\* ptrh; /\* pointer to a host table entry \*/

struct protoent\* ptrp; /\* pointer to a protocol table entry \*/

struct sockaddr\_in sad; /\* structure to hold server's address \*/

struct sockaddr\_in cad; /\* structure to hold client's address \*/

int sd, sd2; /\* socket descriptors \*/

int port; /\* protocol port number \*/

int alen; /\* length of address \*/

pthread\_t tid; /\* variable to hold thread ID \*/

pthread\_t eid; /\* variable to hold thread ID \*/

pthread\_mutex\_init(&mut, NULL);

memset((char\*)&sad, 0, sizeof(sad)); /\* clear sockaddr structure \*/

sad.sin\_family = AF\_INET; /\* set family to Internet \*/

sad.sin\_addr.s\_addr = INADDR\_ANY; /\* set the local IP address \*/

/\* Check command-line argument for protocol port and extract \*/

/\* port number if one is specfied. Otherwise, use the default \*/

/\* port value given by constant PROTOPORT \*/

if (argc > 1)

{ /\* if argument specified \*/

port = atoi(argv[1]); /\* convert argument to binary\*/

}

else

{

port = PROTOPORT; /\* use default port number \*/

}

if (port > 0) /\* test for illegal value \*/

sad.sin\_port = htons((u\_short)port);

else

{ /\* print error message and exit \*/

fprintf(stderr, "bad port number %s/n", argv[1]);

exit(1);

}

/\* Map TCP transport protocol name to protocol number \*/

if (((int)(ptrp = getprotobyname("tcp"))) == 0)

{

fprintf(stderr, "cannot map \"tcp\" to protocol number");

exit(1);

}

/\* Create a socket \*/

sd = socket(PF\_INET, SOCK\_STREAM, ptrp->p\_proto);

if (sd < 0)

{

fprintf(stderr, "socket creation failed\n");

exit(1);

}

/\* Bind a local address to the socket \*/

if (bind(sd, (struct sockaddr\*)&sad, sizeof(sad)) < 0)

{

fprintf(stderr, "bind failed\n");

exit(1);

}

/\* Specify a size of request queue \*/

if (listen(sd, QLEN) < 0)

{

fprintf(stderr, "listen failed\n");

exit(1);

}

alen = sizeof(cad);

pthread\_create(&eid, NULL, evalthread, NULL); // eval thread run first

/\* Main server loop - accept and handle requests \*/

fprintf(stderr, "Server up and running.\n");

while (1)

{

printf("SERVER: Waiting for contact ...\n");

if ((sd2 = accept(sd, (struct sockaddr\*)&cad, &alen)) < 0)

{

fprintf(stderr, "accept failed\n");

exit(1);

}

pthread\_create(&tid, NULL, serverthread, (void\*)sd2);

} /\* end of while \*/

close(sd);

}

void\* serverthread(void\* parm)

{

int tsd;

char buf[100]; /\* buffer for string the server sends \*/

int n;

pthread\_t id;

int mqid;

// struct gameid my\_data;

tsd = (int)parm;

pthread\_mutex\_lock(&mut);

++visits;

printf(" %d clients connected...\n", visits);

pthread\_mutex\_unlock(&mut);

if (visits == 1)

{

if ((mqid = msgget((key\_t)1234, 0666 | IPC\_CREAT)) == -1)

{

printf("1234 msgget failed \n");

pthread\_exit(0);

}

}

else if (visits == 2)

{

if ((mqid = msgget((key\_t)1235, 0666 | IPC\_CREAT)) == -1)

{

printf("1235 msgget failed \n");

pthread\_exit(0);

}

}

else if (visits == 3)

{

if ((mqid = msgget((key\_t)1236, 0666 | IPC\_CREAT)) == -1)

{

printf("1236 msgget failed \n");

pthread\_exit(0);

}

}

id = pthread\_self();

sprintf(buf, "Your PID in Server is %u .. \n", (unsigned int)id);

printf("SERVER thread: %s .. waiting..\n", buf);

send(tsd, buf, strlen(buf), 0);

while (1)

{

n = recv(tsd, buf, sizeof(buf), 0);

buf[n] = '\0';

printf("Server thread : you type -> %s\n", buf);

switch (buf[0])

{

case 'W':

case 'M':

case 'V':

msgsnd(mqid, buf, MAXTEXT, 0);

printf("Server thread : send it to evalthread \n");

break;

case 'Q':

sprintf(buf, " Finish game .. \n");

send(tsd, buf, strlen(buf), 0);

printf("SERVER thread: %u .. exit()..", id);

if (msgctl(mqid, IPC\_RMID, NULL) == -1)

{

printf("msgctl failed \n");

}

close(tsd);

pthread\_exit(0);

break;

default:

sprintf(buf, " invalid input.. try again.. \n");

send(tsd, buf, strlen(buf), 0);

break;

}

if (msgrcv(mqid, buf, MAXTEXT, 0, MSG\_NOERROR) == -1)

{

printf("mq\_b msgrcv failed with error \n");

pthread\_exit(0);

}

printf("SERVER : thread(%u) is %s \n", id, buf);

send(tsd, buf, strlen(buf), 0);

buf[0] = '\0'; // clear buff string

} /\*\* end of while \*/

}

void\* evalthread()

{

int t;

int mqid\_a;

int mqid\_b;

int mqid\_c;

char bufA[10], bufB[10], bufC[10];

key\_t key\_qa = 1234, key\_qb = 1235, key\_qc = 1236;

printf("SERVER: eval thread running .\n");

if ((mqid\_a = msgget(key\_qa, IPC\_CREAT | 0666)) == -1)

{

printf("key\_qa msgget failed \n");

pthread\_exit(0);

}

if ((mqid\_b = msgget(key\_qb, IPC\_CREAT | 0666)) == -1)

{

printf("key\_qb msgget failed \n");

pthread\_exit(0);

}

if ((mqid\_c = msgget(key\_qc, IPC\_CREAT | 0666)) == -1)

{

printf("key\_qc msgget failed \n");

pthread\_exit(0);

}

printf(" three message queues connected\n");

while (1)

{

bufA[0] = '\0';

bufB[0] = '\0';

bufC[0] = '\0';

if (msgrcv(mqid\_a, bufA, MAXTEXT, 0, MSG\_NOERROR) == -1)

{

printf("mg\_a msgrcv failed with error \n");

pthread\_exit(0);

}

printf(" I get one and waiting : A-> %s \n", bufA);

if (msgrcv(mqid\_b, bufB, MAXTEXT, 0, MSG\_NOERROR) == -1)

{

printf("mq\_b msgrcv failed with error \n");

pthread\_exit(0);

}

printf(" I get two and waiting : B-> %s \n", bufB);

if (msgrcv(mqid\_c, bufC, MAXTEXT, 0, MSG\_NOERROR) == -1)

{

printf("mq\_b msgrcv failed with error \n");

pthread\_exit(0);

}

printf(" I get three and eval it : C-> %s \n", bufC);

if (bufA[0] != '\0' && bufB[0] != '\0' && bufC[0] != '\0')

{

t = evalgame(bufA[0], bufB[0], bufC[0]);

printf("SERVER evalgame: eval -> %d\n", t);

switch (t)

{

case 0:

strcpy(bufA, "retry");

strcpy(bufB, "retry");

strcpy(bufC, "retry");

break;

case 1:

strcpy(bufA, "loose");

strcpy(bufB, "loose");

strcpy(bufC, "win");

break;

case 2:

strcpy(bufA, "win");

strcpy(bufB, "win");

strcpy(bufC, "loose");

break;

case 3:

strcpy(bufA, "loose");

strcpy(bufB, "win");

strcpy(bufC, "loose");

break;

case 4:

strcpy(bufA, "loose");

strcpy(bufB, "win");

strcpy(bufC, "win");

break;

case 5:

strcpy(bufA, "win");

strcpy(bufB, "loose");

strcpy(bufC, "win");

break;

case 6:

strcpy(bufA, "win");

strcpy(bufB, "loose");

strcpy(bufC, "loose");

break;

} /\* end of switch \*/

msgsnd(mqid\_a, bufA, MAXTEXT, 0);

msgsnd(mqid\_b, bufB, MAXTEXT, 0);

msgsnd(mqid\_c, bufC, MAXTEXT, 0);

} // end of if

} /\* end of whie \*/

}

int evalgame(char a, char b, char c)

{

switch (a)

{

case 'W':

switch (b)

{

case 'W':

switch (c)

{

case 'W':

return 0; /\* retry... \*/

break;

case 'V':

return 1; /\* C win...\*/

break;

case 'M':

return 2; /\* A,B win...\*/

break;

}

break;

case 'V':

switch (c)

{

case 'W':

return 3; /\* B win...\*/

break;

case 'V':

return 4; /\* B,C win...\*/

break;

case 'M':

return 0; /\* retry... \*/

break;

}

break;

case 'M':

switch (c)

{

case 'W':

return 5; /\* A,C win...\*/

break;

case 'V':

return 0; /\* retry... \*/

break;

case 'M':

return 6; /\* A win...\*/

break;

}

break;

}

break;

case 'V':

switch (b)

{

case 'W':

switch (c)

{

case 'W':

return 6; /\* A win...\*/

break;

case 'V':

return 5; /\* A,C win...\*/

break;

case 'M':

return 0; /\* retry... \*/

break;

}

break;

case 'V':

switch (c)

{

case 'W':

return 2; /\* A,B win...\*/

break;

case 'V':

return 0; /\* retry... \*/

break;

case 'M':

return 1; /\* C win...\*/

break;

}

break;

case 'M':

switch (c)

{

case 'W':

return 0; /\* retry... \*/

break;

case 'V':

return 3; /\* B win...\*/

break;

case 'M':

return 4; /\* B,C win...\*/

break;

}

break;

}

break;

case 'M':

switch (b)

{

case 'W':

switch (c)

{

case 'W':

return 4; /\* B,C win...\*/

break;

case 'V':

return 0; /\* retry... \*/

break;

case 'M':

return 3; /\* B win...\*/

break;

}

break;

case 'V':

switch (c)

{

case 'W':

return 0; /\* retry... \*/

break;

case 'V':

return 6; /\* A win...\*/

break;

case 'M':

return 5; /\* A,C win...\*/

break;

}

break;

case 'M':

switch (c)

{

case 'W':

return 1; /\* C win...\*/

break;

case 'V':

return 2; /\* A,B win...\*/

break;

case 'M':

return 0; /\* retry... \*/

break;

}

break;

}

break;

}

} /\*\* end of evalgame \*/

game\_client.c

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <pthread.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#define closesocket close

#define PROTOPORT 5193 /\* default protocol port number \*/

extern int errno;

char localhost[] = "localhost"; /\* default host name \*/

main(int argc, char \*argv[])

{

struct hostent \*ptrh; /\* pointer to a host table entry \*/

struct protoent \*ptrp; /\* point to a protocol table entry \*/

struct sockaddr\_in sad; /\* structure to hold server's address \*/

int sd; /\* socket descriptor \*/

int port; /\* protocol port number \*/

char \*host; /\* pointer to host name \*/

int n; /\* number of characters read \*/

char buf[100]; /\* buffer for data from the server \*/

int flag = 1;

memset((char \*)&sad, 0, sizeof(sad)); /\* clear sockaddr structure \*/

sad.sin\_family = AF\_INET; /\* set family to Internet \*/

/\* Check command-line argument for protocol port and extract \*/

/\* port number if on is specified. Otherwise, use the default \*/

/\* port value biven by constant PROTOPORT \*/

if (argc > 2)

port = atoi(argv[2]);

else

port = PROTOPORT;

if (port > 0)

sad.sin\_port = htons((u\_short)port);

else

{

fprintf(stderr, "bad port number %s\n", argv[2]);

exit(1);

}

if (argc > 1)

host = argv[1];

else

host = localhost;

ptrh = gethostbyname(host);

if (((char \*)ptrh) == NULL)

{

fprintf(stderr, "invalid host: %s\n", host);

exit(1);

}

memcpy(&sad.sin\_addr, ptrh->h\_addr, ptrh->h\_length);

if (((int)(ptrp = getprotobyname("tcp"))) == 0)

{

fprintf(stderr, "cannot map \"tcp\" to protocol number\n");

exit(1);

}

sd = socket(PF\_INET, SOCK\_STREAM, ptrp->p\_proto);

if (sd < 0)

{

fprintf(stderr, "socket creation failed\n");

exit(1);

}

if (connect(sd, (struct sockaddr \*)&sad, sizeof(sad)) < 0)

{

fprintf(stderr, "connect failed\n");

exit(1);

}

n = recv(sd, buf, sizeof(buf), 0);

fprintf(stderr, "CLIENT: %s \n", buf);

fprintf(stderr, "CLIENT: connected \n");

buf[n] = '\0';

while (flag)

{

switch (flag)

{

case 1:

gets(buf);

fprintf(stderr, "CLIENT: you type %s \n", buf);

if (buf[0] == 'W' || buf[0] == 'M' || buf[0] == 'V')

{

fprintf(stderr, "CLIENT: valid and send it to server %s \n", buf);

send(sd, buf, strlen(buf), 0);

flag = 2;

}

else

flag = 1; /\* continue \*/

break;

case 2:

n = recv(sd, buf, sizeof(buf), 0);

buf[n] = '\0';

fprintf(stderr, "CLIENT: %s \n\n", buf);

if (strncmp(buf, "retry", 5) == 0)

{

printf("CLIENT : same value.. Retry again..\n");

flag = 1;

}

else

{

fprintf(stderr, "CLIENT: play the game again ? (y/n)..\n");

gets(buf);

if (buf[0] == 'y')

{

flag = 1;

buf[0] = '\0';

}

else

{

buf[0] = 'Q';

buf[1] = '\0';

send(sd, buf, strlen(buf), 0);

fprintf(stderr, "CLIENT: finish game..\n");

closesocket(sd);

exit(0);

}

} // else

break;

} // end of switch

} // end of while

closesocket(sd);

exit(0);

}

1. 자료 출처

강의 자료실 예시 코드