acm.mipt.ru

олимпиады по программированию на Физтехе

Очереди сообщений.

Задача Игра.

Требуется написать программы, для организации игры в крестики-нолики или морской бой. Предполагается, наличие **программы-сервера**, одной или двух **программ-клиентов**, которые общаются с сервером.

Программа-сервер обеспечивает:

1. прием и пересылку сообщений от клиентов
2. проверку правильности хода и решение о завершении игры (выигрыш, проигрыш, прервана)

Программа-клиент обеспечивает:

1. возможность сделать ход (либо через консоль, либо автоматически через реализованный алгоритм стратегии игры).
2. пересылку и прием сообщений от сервера.

Для взаимодействия между процессами испрользуется системная структура **очередь сообщений.**

Пример взаимодействия клиент-сервер на языке С

Заголовочный файл

```
#define MAXBUF 100
#define PERM 0666

// Структура для передачи сообщений
typedef struct msgbuf{
// тип сообщения для чтения или записи
   long type;
// для содержания сообщения
   char buf[MAXBUF];
// могут быть и еще другие поля
}Message;
```

Сервер

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <stdlib.h>
#include <sys/msg.h>
#include "mes.h"

int main(){
// сообщение
  Message privet;
// ключ для создания системного ресурса
  key_t key;
// дескриптор очереди сообщений
  int mesid;

  int lng,n;
// получение ключа
// нужно имя заведомо существующего файла
// и символ
  if ((key = ftok( "server", 'A' ) ) < 0 ){
     printf("Can't get key\n");
     exit(1);
  }
// для записи сообщения, его тип
//указывается в структуре
// будем читать и писать тип 1
  privet.type=1L;

// создание очереди сообщений
  if( ( mesid = msgget(key, PERM|IPC_CREAT ) ) < 0 ){
    printf("Can't create message's queue\n");
    exit(1);
  }
// чтение из очереди сообщений
// если сообщений в очереди нет, программа в ожидании
// сообщение должен послать клиент
  n = msgrcv( mesid, &privet, sizeof(Message), privet.type, 0);
// для печати - 0 в конце
  privet.buf[n]='\0';

// n - количество прочитанных байт
// если что-то прочитали, то печать
  if ( n > 0 ){
   printf("Privet: %s poluchen ot %ld\n", privet.buf, privet.type);

// Подготовка нового сообщения для отправки
  lng = sprintf(privet.buf, "Thank's\n");

// отправка сообщения
```

Клиент

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <stdlib.h>

#include "mes.h"

int main(){
  Message privet;
  int key;
  int mesid;
  int lng,n;

  if ( ( key = ftok("server", 'A' ) ) < 0 ){
    printf("Can't get key\n");
    exit(1);
  }
// Очередь уже создана, получаем
// дескриптор
  if( ( mesid = msgget( key, 0 ) ) < 0 ){
    printf("Can't get message's queue\n");
    exit(1);
  }

// Создаем сообщение для посылки
  privet.type = 1L;
 lng = sprintf(privet.buf, "Q-ku");

// отсылаем сообщение (сервер его ждет)
  if ( msgsnd( mesid, (void*)&privet, lng, 0) < 0 ){
    printf("Can't write message\n");
    exit(1);
 }
 char c;
 c = getchar();
// Удалаем очередь сообщений когда не нужна
// сервер может и не успеть прочитать сообщение
 if( msgctl( mesid, IPC_RMID, 0 ) < 0 ){
   printf("Can't delete queue\n");
   exit(1);
 }
return 0;
}
```

```
     if ( msgsnd( mesid, (void*)&privet, lng, 0) < 0 ){
       printf("Can't write message\n");
       exit(1);
     }
  }else{
    printf("The message is not valid\n");
    exit(1);
 }

 return 0;
}
```

Задачи

Задача 1

Добавить в класс SystemFile? две функции int lockM() и int unlockM() для блокировки доступа к файлу. Блокировка должна позволять операции с файлом независимо от очередности доступа к файлу (кто первый захватил, тот первый и работает). Если удалось заблокировать или разблокировать файл, возвращается 0, если нет 1.

Проверить их работу на работес файлами users, и work_log для задачи из предыдущего задачния.

Задача 2. Класс Sender

Класс **Sender** служит для создания и открытия очередей сообщений, для передачи и приема сообщений, а также для удаления очереди сообщений

Заголовочный файл

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <stdlib.h>
#include <sys/msg.h>
#include <time.h>
#include <errno.h>
#include <string.h>

#include <iostream>
#include <cstdlib>
#include <fstream>

#define PERM 0666

using namespace std;

class Sender{
// внутренняя структура для сообщений
   struct Mess{
      long long type;// тип сообщения
// само сообщение
        char buf[100];
// Дополнительные поля:
      int len; // размер сообщения в байтах
// pid процесса, посылавшего сообщение
        int pid;
   };
// сообщение
   Mess msg;
   int key; // ключ
// дескриптор очереди сообщений
   int mesid;
// pid текущего процесса
   int pid;
   public:
// Конструктор для сервера
// Требуется имя файла и символ
      Sender(const char*, char);
// Конструктор для клиента.
// Добавляется еще параметр для различия
      Sender(const char*, char, int);
// деструктор
      ~Sender();
// чтение с возвратом кода ошибки
// ms - тело сообщения, type - тип сообщения
      unsigned char read( void* ms, long long type);

// посылка с возвратом кода ошибки
// ms - тело сообщения, type - тип сообщения
// len - размер тела сообщения
      unsigned char send( void* ms, int len, long long type);
// удаление очереди сообщений
      void delQueue();
};
```

Пример тестирующей программы

Клиент

```
#include "sender.h"

int main(){
   int err;
   string buf;
   int len, queue = 0;
   cout << getpid() << endl;
   Sender snd("serv", 'A', 77);
   cout << "читаем..." << endl;
   err = snd.read( (void*) buf.c_str(), 1L);
```

Сервер

```
#include "sender.h"

int main(){
   string buf = "1";
   int err;
   int len, queue = 0;
   char m1;
   cout << getpid() << endl;
   Sender snd("serv",'A');
   err = snd.send((void*)buf.c_str(), sizeof(char), 1L);
```

```
    cout << "прочитали" << buf << endl;
// очередь удалять будет сервер
}
```

```
   buf = "2";
   err = snd.send((void*)buf.c_str(), sizeof(char), 1L);
   while( buf != "*" ){
      cin >> buf;
  }
  snd.delQueue();
}
```

Написать код для всех функций класса **Sender** и отладить

Задача 3.

Дан примерный заголовочный файл для описания игры (класс **Game**).

```
#include "sender.h"
using namespace std;

class Game{
// Внутренняя структура для описания игрового поля
   struct Pole{
// поле зависит от размера.
// память будет выделена динамически
      int *pl;
// размер по горизонтали и вертикали
      int n, m;
// конструктор
      Pole(int n, int m);
// деструктор
      ~Pole();
// оператор, возвращающий указатель на строку с номером n
// обращение через объект a[i][j]
      int* operator[](int n);
   };
// Указатель на игровое поле
// конструктор только с параметрами, поэтому по умолчанию
// объект не получится
   Pole *gameP;
// размеры
   int m, n;
// указатель на Sender
// для обмена сообщениями
   Sender *snd;
   public:
// конструктор
      Game(int, int);
// в деструукторе удаляется динамическая память и
// очередь сообщений через Sender
      ~Game();
// один игровой ход
      void step(int, int);
// показать состояние поля
      void show();
};
```

Пример реализации отдельных функций класса **Game**

```
#include "game.h"
// описание коструктора Pole
Game::Pole::Pole(int a, int b){
   n = a;
   m = b;
   pl = new int[ n * m ];
   bzero(pl, n * m * sizeof(int));
};
//деструктор Pole
Game::Pole::~Pole(){
   if (pl)
      delete[] pl;
};
//оператор [].
// просто возвращает указатель на строку k
int* Game::Pole::operator[](int k){
      return pl + k * n;
};

Game::Game(int a, int b){
// создаем игровое поле
   gameP = new Pole(a,b);
   n = a;
   m = b;
};

Game::~Game(){
   delete gameP;
};

void Game::show(){
// пример использования []
// если есть указатель нна объект
  int *z = (*gameP)[0];
  cout<<*z<<endl;

// печать игрового поля
  for(int y = 0; y < m; y++){
    for(int x = 0; x < n; x++)

// получив указатель на int, можем поступать
// с ним как с обычным массивом
      cout << (*gameP)[y][x] << ' ';

    cout<<endl;
```

```
  }
};
```

1. Написать все функции для класса **Game** (крестики-нолики или морской бой).

2. Написать **программу-сервер** для получения сообщений от клиентов (ходов), передачи им (информации о сотоянии поля), организации очередности игры и определения победителя.

3. Написать **программу-клиент** для предоставления возможности сделать ход. Обеспечить четкую очередность хода.

4**. Написать **программу-клиент**, которая играет за компьютер. Проверить работу с другой такой же программой.

-- TatyanaOvsyannikova2011 - 26 Oct 2017