acm.mipt.ru

олимпиады по программированию на Физтехе

Раздел «Язык Си» . OOP-Her_begin3 :

Наследники для класса Actor. Очереди сообщений.

Доступ к ***разделяемому ресурсу*** можно регулировать с помощью системного устройства **=очередь сообщений=**

Опишем заголовочный файл (**actor.h**) с описанием структуры сообщений и класса Actor – для действий через очередь сообщений

```
#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;

// Сама очередь сообщений ограничена по памяти (4064 бай)
// поэтому сообщения не должны занимать много места

// структура сообщений
struct Mess{
// тип сообщений. При указании типа не равным 0,
// читаются только сообщения этого типа
// Если тип = 0, то читаются все сообщения
  long type;
// буфер для содержательной части сообщения
  char buf[100];
// могут быть и другие поля
  int pid;// pid процесса
// для передачи целых чисел
  int number;
};

class Actor{
// Две переменные для сообщений:
// для чтения и записи
   Mess mSend, mRead;
// ключ для создания очереди сообщений,
// одной для разных процессов
   key_t key;
//  тип сообщения для чтения
   long typeToRead;
// тип сообщения для посылки
   long typeToSend;
// дескриптор очереди сообщений
   int mesid;
// другие переменные
```

```
   int lng,n;
public:
// конструктор создает очередь сообщений
    Actor();
// деструктор удалаяет очередь сообщений
    ~Actor();
// открывает доступ к ресурсу посылкой сообщения в очередь
    void welcome(int);
// что-то выполняет за int времени
    void act(int);
// подготовить информацию для посылки текста
    void toSend(string);
// получить прочитанный текст
    void write();
// подготовка к посылке числа
   void toSendNumber(int);
//  получить число
   int toGetNumber();
    int stop(int);
};
```

Рассмотрим реализацию некоторых функций

```
#include "actor.h"
// Конструктор создает очередь сообщений
Actor::Actor(){
// ftok() генерит ключ, который нужен для создания очереди сообщений
// для генерации ключа нужен символ и СУЩЕСТВУЮЩИЙ в системе файл
  if ((key = ftok("actor",'A'))<0){
   printf("Can't get key\n");
   exit(1);
  }
// Создание очереди сообщений с проверкой существования
  if((mesid = msgget(key,PERM|IPC_CREAT|IPC_EXCL))<0){
// если очередь уже есть (процесс не первый пришел), просто открываем
    if(errno == EEXIST){
      if ((mesid = msgget(key,0))< 0){
// если не получилось, прерываем программу
        printf("Can't create message's queue\n");
        exit(1);
      }
    }else{
// если другая ошибка - прерываем программу
      printf("Can't create message's queue\n");
      exit(1);
   }
  }
// пока все типы - 0
  typeToSend=typeToRead = 0;
// полю в сообщении для посылки присваиваем свой pid
  mSend.pid = getpid();
};

// Деструкт---++ор удаляет очередь.
// После завершения программы очередь остается в системе.
// Если ее не удалить, последующие запуски будут приводить к ошикам
// в работе программ
Actor::~Actor(){
// msgctl - манипулятор для очереди сообщений
  if( msgctl(mesid,IPC_RMID,0)<---++0){
      printf("Can't delete queue\n");
//      exit(1);
   }
};

// Открываем доступ к ресурсу.
// посылаем сообщений в очередь
// Если процесс может прочитать сообщение,
// он его читает и продолжает дальше работать
// если не может, переходит в ожидание
void Actor::welcome(int p){
```

```
// указываем тип сообщения
    mSend.type = p;
// отсылаем сообщение.
// Здесь следует напсать провеку (удалось ли отослать сообщение)
    msgsnd(mesid,(void*)&mSend,sizeof(Mess),0);
};

// Подготовка  текста к передаче
void Actor::toSend(string s){
   strcpy(mSend.buf,s.c_str());
};

// напечатаь полученный текст.
void Actor::write(){
   string s=mRead.buf;
   cout<<s<<endl;
};

// Закрыть ресурс.
// Читаем сообщение из очереди. Если нет сообщений, ждем.
// Если сообщение было, читаем и количество сообщений в очереди
// уменьшается на одно
int Actor::stop(int p){
    mRead.type = p;
    n=msgrcv(mesid,&mRead, sizeof(Mess), mRead.type,0);
    if(n<0){
      exit(1);
    }
// возвращаем pid отправителя
    return mRead.pid;
};

// что-то желаем tm секунд
void Actor::act(int tm){
    sleep(tm);
};
```

Рассмотрим две программы: сервер и клиент, которые по-очереди могут вводить строки

---++

Сервер

```
#include "actor.h"
// сервер начинает певым
// он читает сообщения типа 2, а
// отсылает тип 1
int main(){
  Actor a;
  string s;
  int time;

  while(1){

      cin>>s;
      a.toSend(s);
      if(s == "*") break;
      a.welcome(1);
      a.stop(2);
      a.write();
  }
  return 0;
}
```

Клиент

```
#include "actor.h"
// Клиент начинает вторым.
// Он читает сообщения типа 1, а отсылает тиа 2
int main(){
  Actor a;
  int rang;
  int set = 0;
  string s;
  int time;

  while(1){
      a.stop(1);
      a.write();
      cin>>s;
      if(s == "*") break;
      a.toSend(s);
      a.welcome(2);
  }
  return 0;
}
```

Если нам хочется расширить возможности класса **Actor** , создадим класс-наследник **CWriter** для реализации чата. При этом будем учитывать, что любой процесс может начинать работу как первым, так и вторым.

Заголовочный файл **cwriter.h**

```
// подключение заголовочного файла
// для класса actor
#include "actor.h"

// класс CWriter - НАСЛЕДНИК класса Actor
class CWriter:public Actor{
// дополнительные атрибуты
  int myPid;    //pid этого процесса
  int otherPid; // pid собеседника
  string s;     // строка для сообщений
// типы сообщений (для ЭТОГО класса)
  long typeToSend,typeToRead;
public:
// Конструктор и деструктор будут---++ использоваться
// из класса Actor
// нам нужна только дополнительная функция,
// которая поможет определить очередность посылок
  int IamHere();
};
```

Ралиация:

```
#include "cwriter.h"

int CWriter::IamHere(){
// посылаем сообщение типа 1
   welcome(1);
// устанавливаем свой pid
   int pd = getpid();
// pid собеседника
   int pdd;
// будем читать из очереди и писать в нее,
// пока не получим другой pid
   while(1){
     if((pdd=stop(1))!= pd) break;
     welcome(1);
   }

   typeToSend =pdd ;
   typeToRead = pd;
  cout<<"Все пришли: я-"<<typeToRead << " он - "<<typeToSend <<"\n";

   return pdd;
};
```

Предполагается что для двух собеседников запускается одна и та же программа:

```
#include "cwriter.h"

int main(){
  CWriter a;
// получаем pid собеседника
  int opid = a.IamHere();
  int pid = getpid();
  int rang;
  int set = 0;
  cout<<"opid="<<opid<<endl;
  string s;
// вычисляем очередность
//  у кого больше pid, тот ходит первым.
  rang = opid - pid;

  while(1){
    if(rang > 0 && set == 0){
     cout<<"я хожу вторвым. жду\n";
      a.stop(pid);
     a.write();

    }else{
      cout<<"посылаю\n";
```

```
    }
      set++;
      cin>>s;
      if(s == "*") break;
      a.toSend(s);---++
      a.welcome(opid);
      a.stop(pid);
      a.write();

  }
  return 0;
}
```

Задачи

Задача 1

Для класса **Actor** дописать функции для передачи чисел типа *int* .

Задача 2.

На остановке стоит очередь из пассажиров. Иногда к остановке подъезжает автобус вместимостью N пассажтиров. Каждый пассажир садится в автобус определенное (для себя время). Автобус ждет пока садятся пассажиры **tm** секунд. Если никакой пассажир в это время не садится в автобус, автобус уезжает. Автобус не должен закрывать двери пока садится пассажир. Пассажир не должен пытаться сесть в автобус, который уехал или еще не приехал. Два и более пассажира не могут садится в автобус одновременно.

Написать два класса-наследника классу **Actor : Bus** и **Passenger** и промоделировать работу остановки, запуском нескольких процессов-пасажиров и автобусов.

```
#include "actor.h"
#include <ctime>

class Bus:public Actor{
  int passnd; // вместимость
  float timeBoard; // всремя загрузки
  clock_t begin; // начало загрузки
public:
// коструктор int - пассажиры
// float для времени
    Bus(int, float);
// подъезд автобуса
    void arrive();
// отправление
    void go();
};

class Passenger:public Actor{
   int timeBoard; // время на посадку
   int mpid;      // pid пассажира
public:
// клнструктор
    Passenger(int);
// посадка
    void toSit();
};
```

Задача 3

Описать класс **Catalog** как наследник класса **SystemFile.**

Для демонстрации работы написать программу, которая печатает только файлы, принадлежащие пользователю, запустившему программу.

-- TatyanaOvsyannikova2011 - 02 Nov 2016