Операционные системы

Межпроцессное взаимодействие

Межпроцессное взаимодействие

Передача информации между процессами через Win32 API

Механизмы межпроцессного обмена (1)

- DDE (Dynamic Data Exchange)
- OLE
- atom (атомы)
- pipes (анонимные каналы)
- named pipes (именованные каналы)
- почтовые ящики (mailslots)
- RPC
- сокеты
- файлы, проецируемые в память (memory-mapped files)
- разделяемая память (Shared Memory)

  *Отличается от предыдущего способа только тем, что в качестве разделяемого файла используется часть файла подкачки.

Механизмы межпроцессного обмена (2)

Сообщения WM_COPYDATA	Лучший способ пересылки блока данных из одной программы в другую.
Анонимные каналы (Anonymous pipes)	Полезны для организации прямой связи между двумя процессами на одном ПК.
Именованные каналы (Named pipes)	Полезны для организации прямой связи между двумя процессами на одном ПК или в сети.
Почтовые ячейки (mailslots)	Полезны для организации связи одного процесса со многими на одном ПК или в сети.
Гнезда (sockets)	Полезны для организации пересылки данных в гетерогенных средах.
Вызов удаленных процедур RPC	Слишком сложен, чтобы использовать его для простых пересылок данных.
Разделяемая память	Непросто выделить вне DLL.
Файлы отображаемой памяти	Обеспечивают одновременный доступ к объектам файла отображения из нескольких процессов.

Атомы

- Атомы – самый простой и доступный способ IPC.
- Идея состоит в том, что процесс может поместить строку в таблицу атомов и эта строка будет видна другим процессам. Когда процесс помещает строку в таблицу атомов, он получает 32-х битное значение (атом), и это значение используется для доступа к строке. Система не различает регистр строки.
- Набор атомов собирается в таблицу (**atom table**).
- Система обеспечивает два типа таблиц атомов для разных задач:
  - локальные (доступны только из приложения);
  - глобальные (доступны из всех приложений).

Функции Win32 API для работы атомами

- GlobalAddAtom
- GlobalGetAtomName
- GlobalFindAtom
- GlobalDeleteAtom

Сообщение WM_COPYDATA

- Сообщение WM_COPYDATA позволяет приложениям копировать данные между их адресными пространствами.
- Перед отправкой сообщения WM_COPYDATA необходимо инициализировать структуру COPYDATASTRUCT с информацией о предстоящей пересылке данных, в том числе с указателем на блок данных. Затем с помощью функции *SendMessage ()* сообщение WM_COPYDATA пересылается в принимающую программу; при этом параметр wParam содержит дескриптор окна вашей программы, а lParam – адрес структуры COPYDATASTRUCT.
- Когда сообщение поступает обработчику WM_COPYDATA принимающей программы, то переданные данные находятся по указателю lpData структуры COPYDATASTRUCT. Размер блока данных извлекается из элемента cbData.
- В структуре COPYDATASTRUCT имеется третье, необязательное поле, dwData, в котором можно передать 32-разрядное число.

Пример использования сообщения WM_COPYDATA

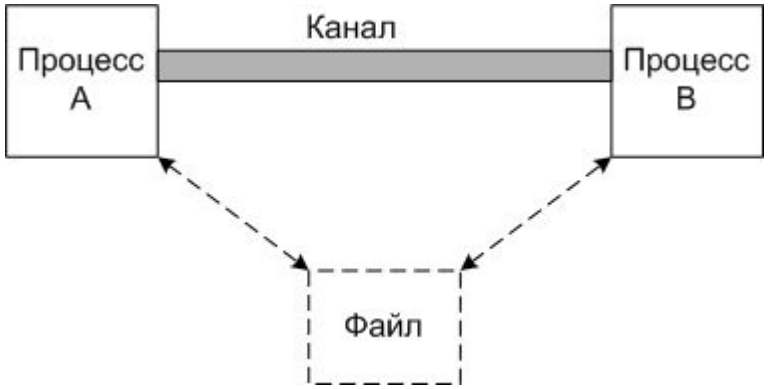
Отправитель:

```
COPYDATASTRUCT cds;
cds.cbData = (DWORD) nSize;
cds.lpData = (PVOID) pBuffer;
SendMessage (hWndTarget, WM_COPYDATA, (WPARAM) hWnd,
    (LPARAM) &cds);
```

Получатель:

```
PCOPYDATASTRUCT pcds = (PCOPYDATASTRUCT) lParam;
PBYTE pBuffer = (PBYTE) pcds -> lpData;
```

Каналы



- Канал представляет собой виртуальное соединение, по которому передается информация от одного процесса к другому.
- С точки зрения программиста канал является специальным файлом с организацией типа буфера FIFO.

Передача информации между процессами через Win32 API

Анонимные каналы

Анонимные каналы

- Анонимные каналы не имеют имен.
- Не пригодны для обмена через сеть.
- Главная цель – служить каналом между родительским и дочерним процессом или между дочерними процессами.
- Односторонний обмен.
- Не возможен асинхронный обмен.

Использование анонимных каналов

- Главная цель – служить каналом между родительским и дочерним процессом или между дочерними процессами.
- Родительский процесс может быть консольным или GUI-приложение, но дочернее приложение должно быть консольным. Как вы знаете, консольное приложение использует стандартные дескрипторы для ввода и вывода.
- Если мы хотите перенаправить ввод/вывод консольного приложения, мы можем заменить один дескриптор другим дескриптором одного конца канала. Консольное приложение не будет знать, что оно использует один конец канала, оно будет считать, что это стандартный дескриптор.
- В некотором роде это вид полиморфизма, который позволяет обойтись без модификации родительского процесса.

Создание анонимных каналов

```
BOOL CreatePipe(
    PHANDLE hReadPipe, //дескриптор конца чтения канала
    PHANDLE hWritePipe, //дескриптор конца записи канала
    LPSECURITY_ATTRIBUTES lpPipeAttributes, //атрибуты защиты
    DWORD nSize //размер буфера канала (0 – по умолчанию)
);
```

ReadFile () – чтение из канала

WriteFile () – запись в канал

Пример использования анонимного канала

- Создаем анонимный канал с помощью *CreatePipe ()*.
- Подготавливаем строку параметров для дочернего процесса.
- Вызываем *CreateProcess ()*, чтобы загрузить дочернее приложение.
- Закрываем дескриптор записи канала. Это необходимо, так для родительского процесса этот дескриптор не нужен, а канал не будет работать, если открыть более чем один дескриптор записи.
- Теперь вы можете читать данные с помощью *ReadFile ()*. Вы должны последовательно вызывать *ReadFile ()*, пока она не возвратит ноль, что будет означать, что больше данных нет.
- После окончания работы закроем дескриптор чтения канала.

Проверка наличия данных без считывания

- С помощью функции *PeekNamedPipe ()* есть возможность проверить наличие данных в анонимном канале без их считывания.
- Подробнее при рассмотрении именованных каналов.

Передача информации между процессами через Win32 API

Именованные каналы

Именованные каналы

- **Named Pipe File System** является виртуальной файловой системой, которая управляет именованными каналами.
- Каналы named pipes относятся к классу файловых объектов API Win32.
- Именованный канал может быть установлен между процессами разных компьютеров, объединенных сетью.
- Именованный канал может быть однонаправленным или двунаправленным (дуплексным).
- Именованный канал поддерживает асинхронный обмен.

Формат имени канала

\\.\pipe*[path]pipename*

\\computer_name\pipe*[path]pipename*

Создание именованного канала

```
HANDLE CreateNamedPipe (
    LPCTSTR lpName,
    DWORD dwOpenMode,
    DWORD dwPipeMode,
    DWORD nMaxInstances,
    DWORD nOutBufferSize,
    DWORD nInBufferSize,
    DWORD nDefaultTimeOut,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes
);
```

Параметры создания канала

- lpName – имя именованного канала;
- dwOpenMode – направление передачи данных (PIPE_ACCESS_DUPLEX, PIPE_ACCESS_INBOUND, PIPE_ACCESS_OUTBOUND);
- dwPipeMode – режим передачи данных (см. далее);
- nMaxInstances – максимальное количество каналов с данным именем, которые могут открыть клиенты (обычно от 1 до 255);
- nOutBufferSize и nInBufferSize – размер буферов на отправку и прием (со стороны сервера), 0 – размер по умолчанию;
- nDefaultTimeout – время ожидания по умолчанию для функции *WaitNamedPipe ()*;
- lpSecurityAttributes – указатель на структуру с атрибутами защиты создаваемого объекта.

Режимы передачи

- PIPE_TYPE_BYTE, PIPE_TYPE_MESSAGE – данные записываются в канал как поток байт или как сообщения
- PIPE_READMODE_BYTE, PIPE_READMODE_MESSAGE – данные считываются из канала как поток байт или как сообщения
- PIPE_WAIT, PIPE_NOWAIT – обмен происходит в блокирующем или неблокирующем режиме

Подключение к именованному каналу

```
BOOL ConnectNamedPipe (
   HANDLE hNamedPipe,
   LPOVERLAPPED lpOverlapped
);


BOOL DisconnectNamedPipe (
   HANDLE hNamedPipe
);
```

Работа с каналом на стороне сервера

- После того как канал создан, сервер подключается к нему с помощью функции *ConnectNamedPipe ()* и начинает ожидать подключения клиента. Необходимо отметить, что подключение сервера к каналу может осуществляться как синхронным, так и асинхронным способом (с использованием структуры OVERLAPPED).
- После установления виртуального соединение серверный процесс и клиентский процесс могут обмениваться информацией при помощи функций *ReadFile ()* и *WriteFile ()*.
- После завершения обмена необходимо отключиться от канала с помощью функции *DisconnectNamedPipe ()*.
- Затем можно снова открыть канал и ожидать подключения следующего клиента, а по завершению работы с каналом необходимо закрыть его дескриптор функцией *CloseHandle ()*.

Пример клиент-серверного приложения (сервер)

```
HANDLE hPipe =
    CreateNamedPipe("\\.\\pipe\\PipeSrv",PIPE_ACCESS_DUPLEX |
    WRITE_DAC, PIPE_TYPE_BYTE,1,100,100,100,NULL);
if (hPipe==INVALID_HANDLE_VALUE) {
    …//Обработка ошибки создания канала
}
ConnectNamedPipe(hPipe,NULL);
DWORD lpBuf; char cName[100];
ZeroMemory(&cName[0],sizeof(cName));
strcpy(&cName[0],"Hello world!");
WriteFile(hPipe,&cName,sizeof(cName),&lpBuf,NULL);
DisconnectNamedPipe(hPipe);
CloseHandle(hPipe);
```

Работа с каналом на стороне клиента

- Клиенты производят подключение к каналу посредством вызова функции *Create File ()*.
- Далее сервер и клиенты могут обмениваться данными с помощью функций *ReadFile ()* и *WriteFile ()*.
- Клиентский процесс может отключиться от канала в любой момент с помощью функции *CloseHandle ()*.

Пример клиент-серверного приложения (клиент)

```
…
char szName [] = “\\ServerName\pipe\PipeSrv”;
HANDLE hFile = CreateFile(szName,GENERIC_READ | GENERIC_WRITE,
    0,NULL,OPEN_EXISTING,0,NULL);

if (hFile==INVALID_HANDLE_VALUE) {
    …//Обработка ошибки открытия канала
}

char str[100]; DWORD lpBuff;
ZeroMemory(&str[0],sizeof(str));
ReadFile(hFile,str,sizeof(str),&lpBuff,NULL);
printf("Server sent:\n%s",str);
CloseHandle(hFile);
```

Реализация нескольких экземпляров канала

- При помощи одного и того же канала сервер может одновременно обслуживать нескольких клиентов. Для этого серверный процесс может создать N-ное количество экземпляров канала, вызвав N-ное количество раз функцию *CreateNamedPipe ().*
- При вызове *CreateNamedPipe ()* необходимо указывать в параметре lpName одно и тоже имя канала, а в параметре nMaxInstances количество экземпляров канала (N).

Установление гарантированного соединения со стороны клиента

- Для установления гарантированного подключения клиента к именованному каналу перед вызовом функции *CreateFile ()* на стороне клиента возможен вызов функции *WaitNamedPipe ()*, которая переведет процесс в режим ожидания соединения с сервером.
- Функция успешно завершается, если на сервере имеется незавершенный вызов функции *ConnectNamedPipe ()*, который указывает на наличие доступного экземпляра именованного канала.

Функция *WaitNamedPipe*

```
BOOL WaitNamedPipe (
   LPCTSTR lpNamedPipeName, //имя канала
   DWORD nTimeOut // интервал ожидания
 );
```

- NMPWAIT_USE_DEFAULT_WAIT – интервал времени ожидания определяется значением параметра nDefaultTimeOut, который задается в функции *CreateNamedPipe ()*;
- NMPWAIT_WAIT_FOREVER – бесконечное время ожидания связи с именованным каналом.

Определение наличия данных в канале

- С помощью функции *PeekNamedPipe ()* процесс-читатель может также определить, имеются ли в канале данные без их удаления.
- Функции *PeekNamedPipe ()* позволяет определить общее количество байт в канале и размер первого в очереди сообщения, либо просто определить факта наличия каких-либо данных в канале.
- Функция *PeekNamedPipe ()* работает как с именованными, так и с анонимными!

Функция *PeekNamedPipe*

```
BOOL PeekNamedPipe(
    HANDLE hNamedPipe, //дескриптор канала
    LPVOID lpBuffer, //адрес буфера для чтения
    //(если проверяется только факт наличия данных, то NULL)
    DWORD cbBuffer, //размер буфера для чтения
    LPDWORD lpBytesRead, //кол-во считанных байт сообщения
    LPDWORD lpTotalBytesAvail, //суммарно кол-во байт в канале
    LPDWORD lpcbMessage //кол-во несчитанных байт сообщения
)
```

Передача информации между процессами через Win32 API

Почтовые ящики

Почтовые ящики (MailSlots)

- *Mailslot* является одним из механизмов, предназначенных для осуществления обмена данными между процессами. При этом процессы могут быть запущены как на одном компьютере (локально), так и разных компьютерах, объединенных сетью (удалённо).
- Обмен данными посредством Mailslot осуществляется в между двумя процессами – клиентом и сервером.
- Приложение-сервер открывает почтовый ящик, а клиенты могут писать в него. Ящик сохраняет сообщения до тех пор, пока сервер их не прочтет.
- Одно приложение может одновременно быть сервером и клиентом, обеспечивая двунаправленную связь. При этом приложения могут находиться даже на разных компьютерах в сети.

Формат имени сервера

\\.*mailslot**[path]name*

*ComputerName**mailslot**[path]name*

***mailslot**[path]name*

*DomainName**mailslot**[path]name*

Создание почтового ящика на сервере

```
HANDLE CreateMailslot (
    LPCTSTR lpName, //имя ящика
    DWORD nMaxMessageSize,
    //максимальный размер сообщения
    DWORD lReadTimeout,
    //интервал-тайм аута чтения
    LPSECURITY_ATTRIBUTES  lpSecurityAttributes
    //атрибуты защиты
);
```

Пример создания сервера

```
HANDLE hSlot = NULL;
hSlot = CreateMailslot ("\\\\computername\\mailslot\\messngr",
   0, MAILSLOT_WAIT_FOREVER, NULL);

if (hSlot != INVALID_HANDLE_VALUE)
{
   char buffer[255]; DWORD nBytesRead;
   ReadFile(hSlot, &buffer, 255, &nBytesRead, NULL);
   …
}
```

Открытие клиентом почтового ящика

```
HANDLE hSlot =
   CreateFile(("\\\\computername\\mailslot\\messngr",
   GENERIC_WRITE, FILE_SHARE_READ, NULL,
   OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

if (hSlot != INVALID_HANDLE_VALUE)
{
   char buf = "From\0\To\0Message\0";
   uint cb = sizeof(buf);
   WriteFile(hSlot, buf, cb, &cb, NULL);
   …
}
```

Ошибка открытия ящика

- В **MSDN** указано, что если клиент открывает слот прежде чем слот был создан сервером, то он получит **INVALID_HANDLE_VALUE**.

Использование mailslot

- Использование почтовых ящиков особенно удобно в системах «клиент-сервер», работающих в пределах локальной сети.
- Кроме систем «клиент-сервер» почтовые ящики можно использовать, например, для определения, запущена ли еще одна копия программы где-либо в локальной сети. Это делается посылкой сообщения всем компьютерам в заданном домене (второй сервер прикидывается клиентом и пытается установить связь с сервером). Если связь установлена, то работу не продолжаем, а если нет, то можно самому работать сервером.

Получение информации о почтовом ящике

```
BOOL GetMailslotInfo (
    HANDLE hMailslot, //указатель на слот
    LPDWORD lpMaxMessageSize,
    //максимальный размер сообщения
    LPDWORD lpNextSize,
    //размер следующего сообщения
    LPDWORD lpMessageCount,
    //количество сообщений
    LPDWORD lpReadTimeout //тайм аут
);
```

Изменение настроек почтового ящика

```
BOOL SetMailslotInfo(
   HANDLE hMailslot,
   DWORD lReadTimeout
);
```

Передача информации между процессами через Win32 API

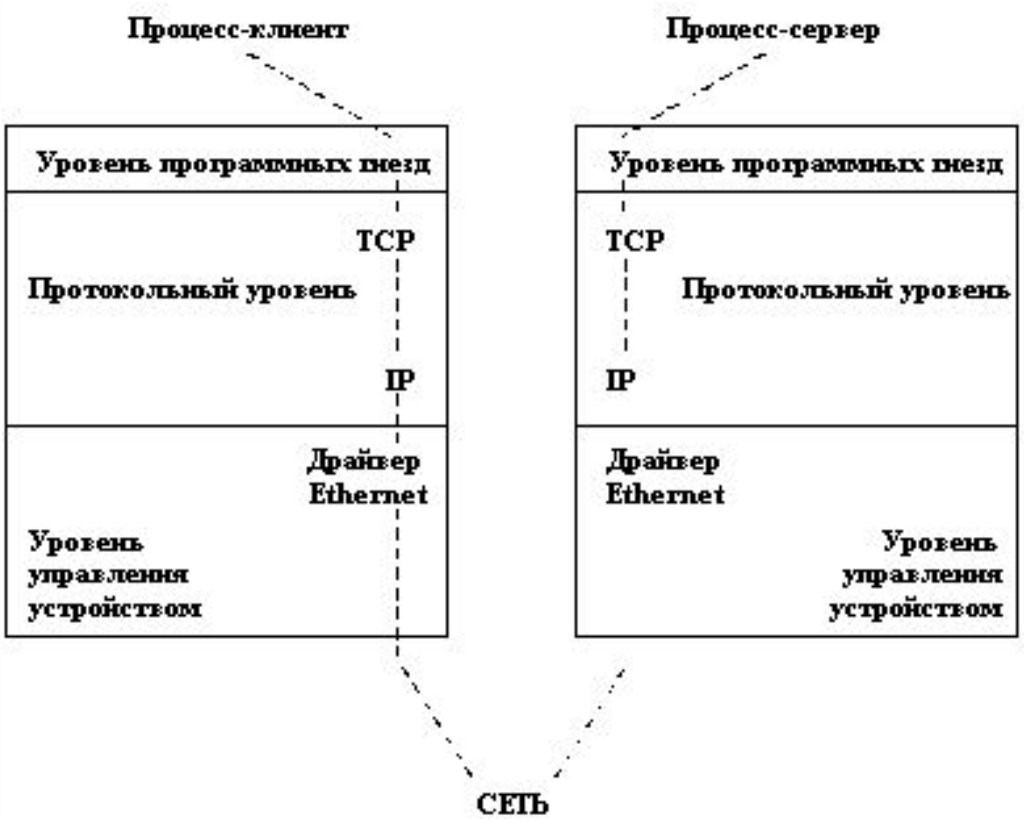
Другие механизмы

Удаленный вызов процедур (RPC)

- RPC (Remote Procedure Call) – это API, позволяющий приложению удаленно вызывать функции в других процессах как на своем, так и на удаленном компьютере.
- RPC реализован как надстройка над NPFS.
- Предоставляемая Win32 API модель RPC совместима со спецификациями Distributed Computing Environment (DCE), разработанными Open Software Foundation. Это позволяет приложениям Win32 удаленно вызывать процедуры приложений, выполняющихся на других компьютерах под другими операционными системами.
- RPC обеспечивают автоматическое преобразование данных между различными аппаратными и программными архитектурами.

Сокеты (программные гнезда)

- Взаимодействие процессов на основе сокетов (программных гнезд) основано на модели «клиент-сервер».



Типы сокетов

- Выделяются два типа сокетов – с виртуальным соединением (stream sockets) и датаграммные сокеты (datagram sockets).
- При использовании сокетов с виртуальным соединением обеспечивается передача данных от клиента к серверу в виде непрерывного потока байтов с гарантией доставки. При этом до начала передачи данных должно быть установлено соединение.
- Датаграммные сокеты не гарантируют абсолютной надежной, последовательной доставки сообщений и отсутствия дубликатов пакетов данных – датаграмм. Но для использования датаграммного режима не требуется предварительное установление соединений, и поэтому этот режим во многих случаях является предпочтительным.
- Система по умолчанию сама обеспечивает подходящий протокол. Например, протокол TCP используется по умолчанию для виртуальных соединений, а протокол UDP – для датаграммного способа коммуникаций.

Сокеты или именованные каналы

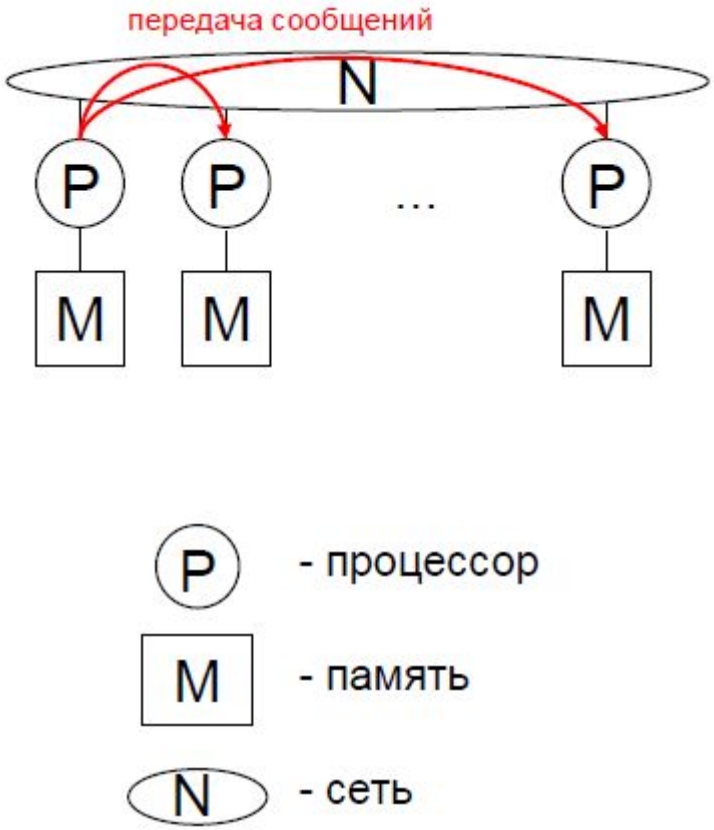
- Если процессы работают на одном компьютере или в рамках одной быстродействующей локальной сети, то имеет смысл использовать именованные каналы. Именованные каналы работают на уровне ядра и поэтому обмен будет более производительный.
- В маршрутизируемых (глобальных) сетях необходимо использовать сокеты, т.к. TCP\IP генерирует меньше трафика, чем именованные каналы при выполнении одной и тоже операции, одновременно именованные каналы не могут работать в маршрутизируемых сетях.

Стандарт MPI

- В вычислительных системах с распределенной памятью процессоры работают независимо друг от друга. Для организации параллельных вычислений в этом случае необходимо иметь возможность распределять вычислительную нагрузку и организовать информационное взаимодействие между процессорами.
- Такие системы сложнее программировать, т.к. каждый процессор системы может использовать только свою локальную память, а для доступа к данным другого процессора необходимо явно выполнить операции передачи сообщений.
- Стандарт MPI (message passing interface) позволяет решить поставленную проблему.

Модель SPMP

- В рамках MPI принят простой подход – для решения задачи разрабатывается одна программа, которая запускается одновременно на выполнение на всех имеющихся процессорах.
- Подобный способ организации параллельных вычислений получил наименование модели "одна программа множество процессов" (single program multiple processes or SPMP).



Параллельная программа с использованием MPI

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char* argv[])
{
    int ProcNum, ProcRank, RecvRank;
    MPI_Status Status; MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
    if ( ProcRank == 0 )
    {
        // Действия, выполняемые только процессом с рангом 0
        printf("\n Hello from process %3d", ProcRank);
        for (int i = 1; i < ProcNum; i++ )
        {
            MPI_Recv(&RecvRank, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
                MPI_COMM_WORLD, &Status);
            printf("\n Hello from process %3d", RecvRank);
        }
    } else // Сообщение, отправляемое всеми процессами
                //кроме процесса с рангом 0
        MPI_Send(&ProcRank,1,MPI_INT,0,0,MPI_COMM_WORLD); MPI_Finalize();
    return 0;
}
```