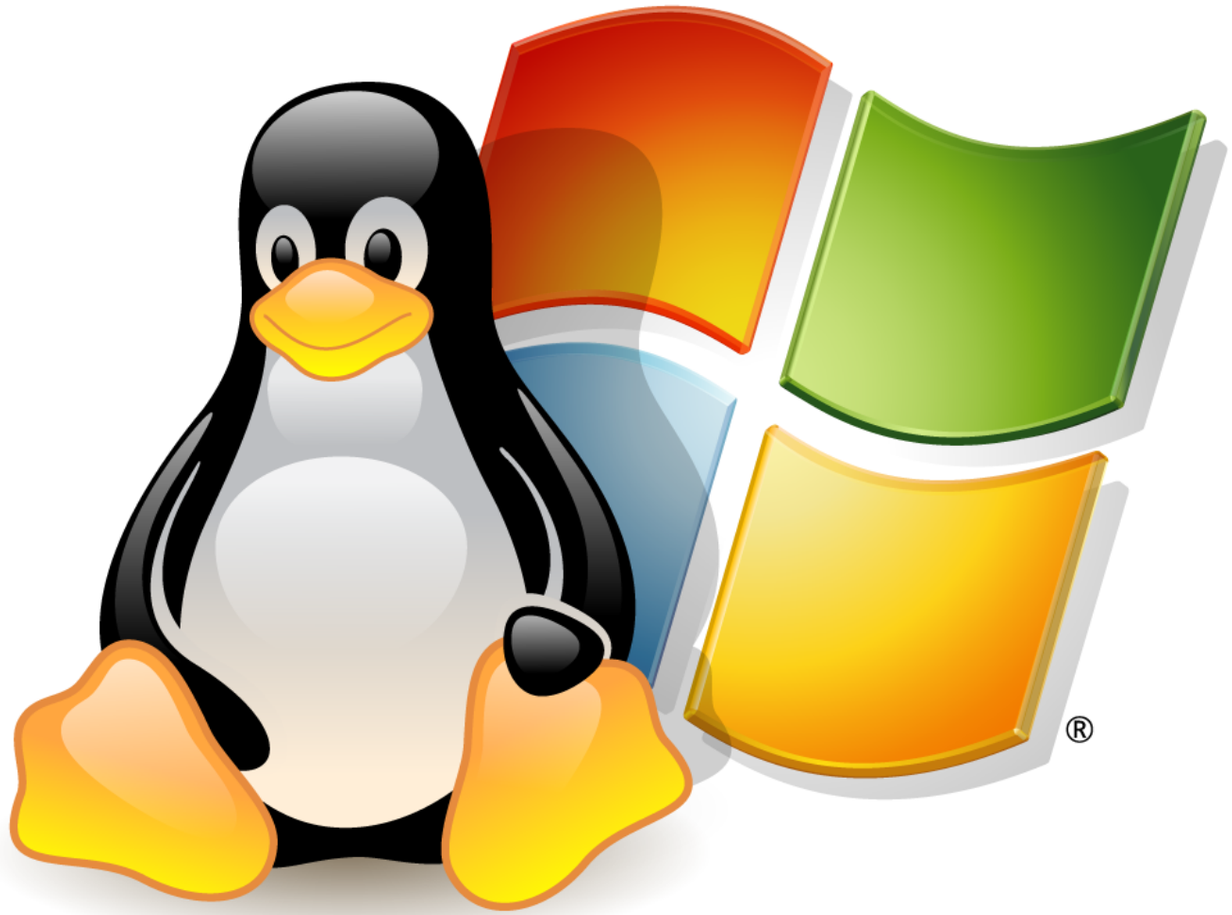


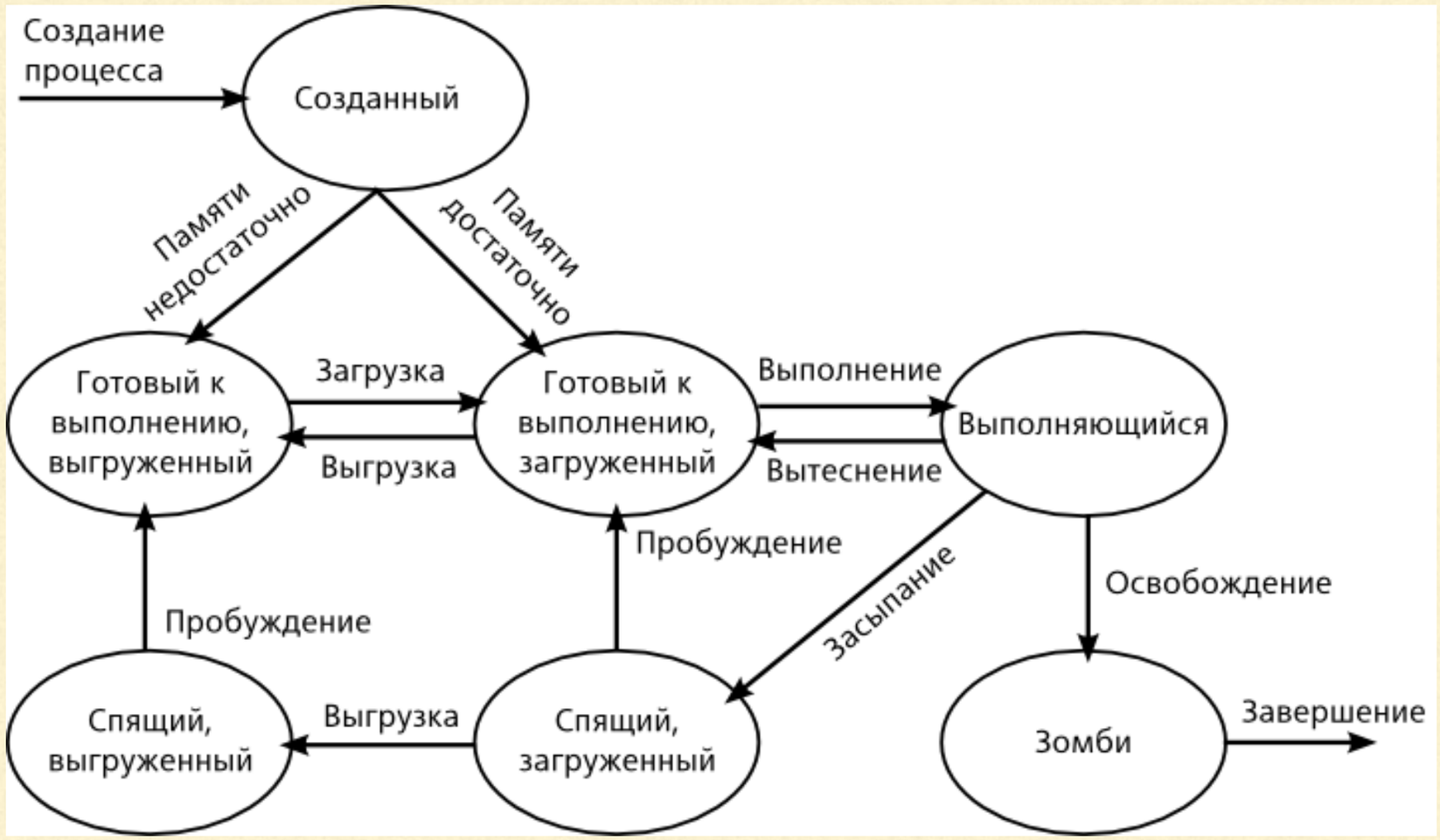
Операционные системы
Лекция 3. Процессы и потоки

Кафедра ВС, Бочкарев Борис Вячеславович

Осень-2019

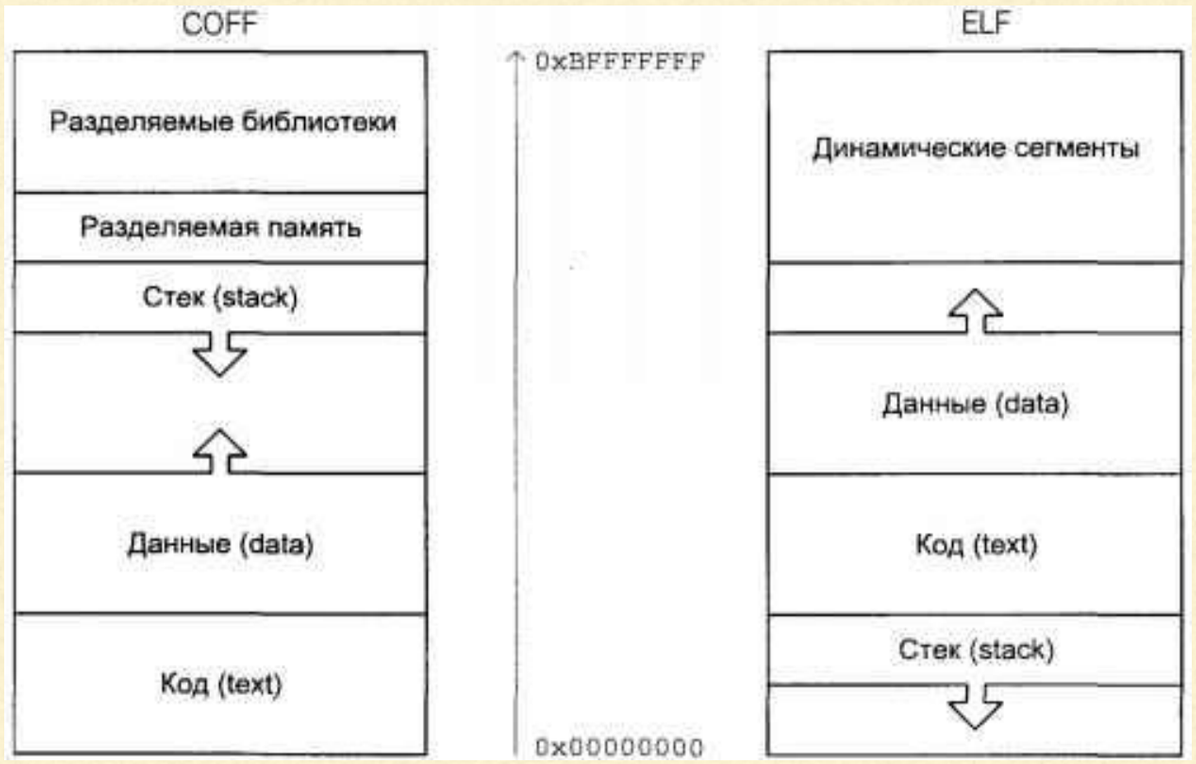
ПРОЦЕСС

программа, выполняющаяся в текущий момент. Имеет идентификатор, идентификатор родительского процесса и состояние.



АДРЕСНОЕ ПРОСТРАНСТВО

совокупность всех допустимых адресов каких-либо объектов вычислительной системы — ячеек памяти, секторов диска, узлов сети и т. п., которые могут быть использованы для доступа к этим объектам при определенном режиме работы (состоянии системы).



СОЗДАНИЕ ПРОЦЕССОВ (LINUX)

```
#include <unistd.h>
pid_t fork(void);
```

pid_t ~ int:
-1 - ошибка
0 - возвращается в дочернем процессе
<pid> - возвращается в родительский процесс

```
pid_t getpid(void);
pid_t getppid(void);
```

```
int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
```

СОЗДАНИЕ ПРОЦЕССОВ (LINUX)

```
main(){
  pid_t pid;
  int rv;
  switch(pid=fork()) {
  case -1:
        perror("fork"); /* произошла ошибка */
        exit(-1); /*выход из родительского процесса*/
  case 0:
        printf(" CHILD: Это процесс-потомок!\n");
        printf(" CHILD: Мой PID -- %d\n", getpid());
        printf(" CHILD: PID моего родителя -- %d\n",
            getppid());
  default:
        printf("PARENT: Это процесс-родитель!\n");
        printf("PARENT: Мой PID -- %d\n", getpid());
        printf("PARENT: PID моего потомка %d\n",pid);
        printf("PARENT: Я жду, пока потомок
                    не вызовет exit()...\n");
        wait();
  }
}
```

СОЗДАНИЕ ПРОЦЕССОВ (LINUX)

```
#include <unistd.h>
#include <stdio.h>

int main (void) {

        /* Определить массив с завершающим нулем команды для запуска
         следующим за любым параметром,  в этом случае никаким */
        char *arg[] = { "/usr/bin/ls",  0 };

        /* fork и exec в порожденном процессе */
        if (fork() == 0) {
                printf("In child process:\n");
                execv(arg[0],  arg);
                printf("I will never be called\n");
        }
        printf("Execution continues in parent process\n");
}
```

СОЗДАНИЕ ПРОЦЕССОВ (WINDOWS)

```
BOOL CreateProcess
(
    LPCTSTR lpApplicationName,                  // имя исполняемого модуля
    LPTSTR lpCommandLine,                      // Командная строка
    LPSECURITY_ATTRIBUTES lpProcessAttributes, // Указатель на структуру SECURITY_ATTRIBUTES
    LPSECURITY_ATTRIBUTES lpThreadAttributes,  // Указатель на структуру SECURITY_ATTRIBUTES
    BOOL bInheritHandles,                      // Флаг наследования текущего процесса
    DWORD dwCreationFlags,                     // Флаги способов создания процесса
    LPVOID lpEnvironment,                      // Указатель на блок среды
    LPCTSTR lpCurrentDirectory,                // Текущий диск или каталог
    LPSTARTUPINFO lpStartupInfo,               // Указатель нас структуру STARTUPINFO
    LPPROCESS_INFORMATION lpProcessInformation // Указатель нас структуру PROCESS_INFORMATION
);
```

http://www.firststeps.ru/mfc/winapi/r.php?131

СОЗДАНИЕ ПРОЦЕССОВ (WINDOWS)

dwCreationFlags. Флаг способа создание процесса и его приоритет.

CREATE_DEFAULT_ERROR_MODE Новый процесс не наследует режим ошибок (error mode) вызывающего процесса.

CREATE_NEW_CONSOLE Новый процесс получает новую консоль вместо того, чтобы унаследовать родительскую.

CREATE_NEW_PROCESS_GROUP Создаваемый процесс - корневой процесс новой группы.

CREATE_SEPARATE_WOW_VDM только Windows NT: Если этот флаг установлен, новый процесс запускается в собственной Virtual DOS Machine (VDM).

CREATE_SHARED_WOW_VDM только Windows NT: Этот флаг указывает функции CreateProcess запустит новый процесс в разделяемой Virtual DOS Machine.

CREATE_SUSPENDED Первичная нить процесса создается в спящем (suspended) состоянии и не выполняется до вызова функции ResumeThread.

CREATE_UNICODE_ENVIRONMENT Если этот флаг установлен, блок переменных окружения, указанный в параметре lpEnvironment, использует кодировку Unicode. Иначе - кодировку ANSI.

DEBUG_PROCESS Если этот флаг установлен, вызывающий процесс считается отладчиком, а новый процесс - отлаживаемым.

DEBUG_ONLY_THIS_PROCESS Если этот флаг не установлен и вызывающий процесс находится под отладкой, новый процесс так же становится отлаживаемым тем же отладчиком.

DETACHED_PROCESS Создаваемый процесс не имеет доступа к родительской консоли. Этот флаг нельзя использовать с флагом CREATE_NEW_CONSOLE.

HIGH_PRIORITY_CLASS Указывает на то, что процесс выполняет критичные по времени задачи

IDLE_PRIORITY_CLASS Указывает процесс, выполняются только когда система находится в состоянии ожидания

NORMAL_PRIORITY_CLASS Указывает на процесс, без каких либо специальных требований к выполнению.

REALTIME_PRIORITY_CLASS Указывает процесс имеющий наивысший возможный приоритет.

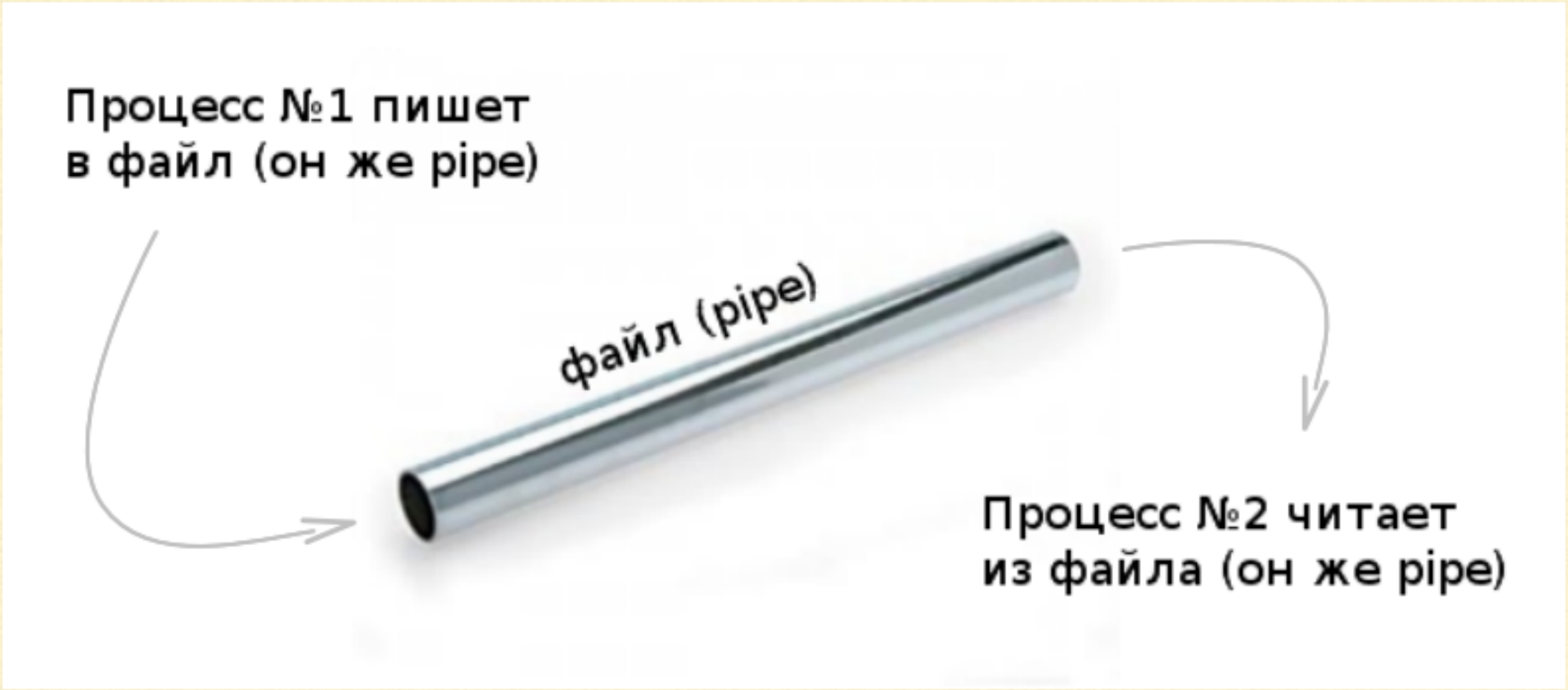
СОЗДАНИЕ ПРОЦЕССОВ (WINDOWS)

```
#include "stdafx.h"
#include "windows.h"
#include "iostream.h"

void main()
{
	STARTUPINFO cif;
	ZeroMemory(&cif,sizeof(STARTUPINFO));
	PROCESS_INFORMATION pi;
	if (CreateProcess("c:\\windows\\notepad.exe",NULL,
		NULL,NULL,FALSE,NULL,NULL,NULL,&cif,&pi)==TRUE)
	{
		cout << "process" << endl;
		cout << "handle " << pi.hProcess << endl;
		Sleep(1000);                          // подождать
		TerminateProcess(pi.hProcess,NO_ERROR);       // убрать процесс
	}
}
```

ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ (LINUX)

1. Именованный канал (pipe) для родственных процессов



ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ (LINUX)

IPC - inter-process communication

```
main(){
    int pipedes[2];
    pid_t pid;
    pipe(pipedes);
    pid = fork();
    if ( pid > 0 ) {
        char *str = "String passed via pipe\n";
        close(pipedes[0]);
        write(pipedes[1], (void *) str, strlen(str) + 1);
        close(pipedes[1]);
    } else {
        char buf[1024];
        close(pipedes[1]);
        int len = read(pipedes[0], buf, 1024);
        write(2, buf, len);
        close(pipedes[0]);
    }
}
```

ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ (LINUX)

2. Очереди сообщений. Запись

```
int main (void) {
    key_t ipckey;
    int mq_id;
    struct { long type; char text[100]; } mymsg;
    /* Генерация IPC ключа */
    ipckey = ftok("/tmp/foo", 42);
    printf("My key is %d\n", ipckey);
    /* Создание очереди */
    mq_id = msgget(ipckey, IPC_CREAT | 0666);
    printf("Message identifier is %d\n", mq_id);
    /* Отправка сообщения */
    memset(mymsg.text, 0, 100); /* Clear out the space */
    strcpy(mymsg.text, "Hello, world!");
    mymsg.type = 1;
    msgsnd(mq_id, &mymsg, sizeof(mymsg), 0);
}
```

ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ (LINUX)

2. Очереди сообщений. Чтение

```
int main (void) {
    key_t ipckey;
    int mq_id;
    struct { long type; char text[100]; } mymsg;
    int received;
    /* Генерация IPC ключа */
    ipckey = ftok("/tmp/foo", 42);
    printf("My key is %d\n", ipckey);
    /* Подключение к очереди */
    mq_id = msgget(ipckey, 0);
    printf("Message identifier is %d\n", mq_id);
    /* Считывание сообщения */
    received = msgrcv(mq_id, &mymsg, sizeof(mymsg), 0, 0);
    printf("%s (%d)\n", mymsg.text, received);
}
```

ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ (WINDOWS)

1. Создание анонимного канала

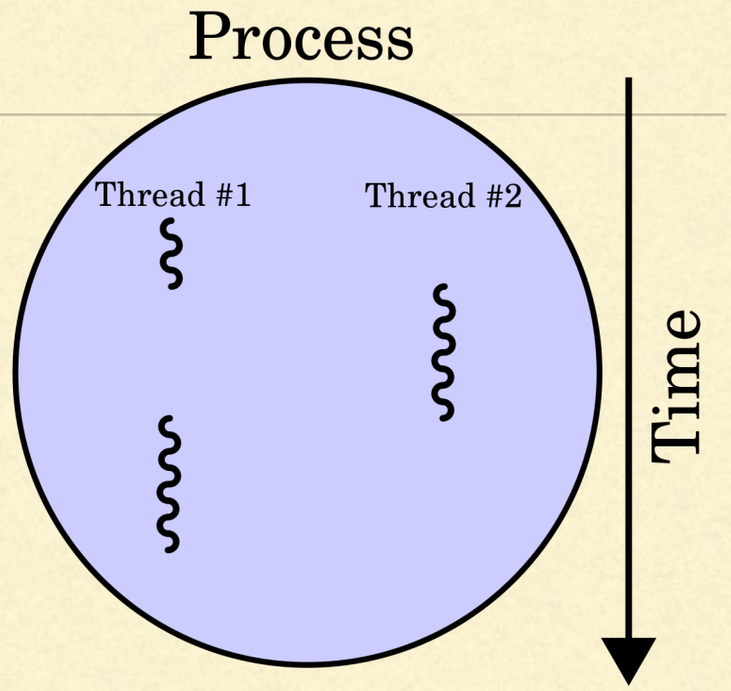
```
BOOL CreatePipe(
  PHANDLE hReadPipe,  // адрес переменной, в которую будет
                      // записан идентификатор канала для
                      // чтения данных
  PHANDLE hWritePipe, // адрес переменной, в которую будет
                      // записан идентификатор канала для
                      // записи данных
LPSECURITY_ATTRIBUTES lpPipeAttributes, // адрес переменной
                    // для атрибутов защиты
DWORD nSize);       // количество байт памяти,
                    // зарезервированной для канала
```

2. Создание именованного канала

```
HANDLE CreateNamedPipe(
  LPCTSTR lpName,           // адрес строки имени канала
  DWORD   dwOpenMode,       // режим открытия канала
  DWORD   dwPipeMode,       // режим работы канала
  DWORD   nMaxInstances,    // максимальное количество
                            // реализаций канала
  DWORD   nOutBufferSize,   // размер выходного буфера в байтах
  DWORD   nInBufferSize,    // размер входного буфера в байтах
  DWORD   nDefaultTimeOut, // время ожидания в миллисекундах
  LPSECURITY_ATTRIBUTES lpSecurityAttributes); // адрес
                            // переменной для атрибутов защиты
```

ПОТОКИ (LINUX)

```
void* threadFunc(void* thread_data){
	//завершаем поток
	pthread_exit(0);
}
int main(){
	void* thread_data = NULL;
	//создаем идентификатор потока
	pthread_t thread;
	//создаем поток по идентификатору thread и функции потока threadFunc
	//и передаем потоку указатель на данные thread_data
	pthread_create(&thread, NULL, threadFunc, thread_data);
	//ждем завершения потока
	pthread_join(thread, NULL);
	return 0;
}
```



ПОТОКИ (WINDOWS)

```
DWORD WINAPI thread2(LPVOID t){
	cout << "Second thread\n";
	return 0;
}
int main(){
	cout << "First thread\n";
	HANDLE thread = CreateThread(NULL,0,thread2,NULL, 0, NULL);
	cout << "More data from first thread\n";
	for (int i = 0; i < 1000000; i++){}
	cout << "Even more data from first thread\n";
	_getch();
	return 0;
}
```

```
HANDLE CreateThread(
    LPSECURITY_ATTRIBUTES lpThreadAttributes, // дескриптор защиты
    SIZE_T dwStackSize,                       // начальный размер стека
    LPTHREAD_START_ROUTINE lpStartAddress,    // функция потока
    LPVOID lpParameter,                       // параметр потока
    DWORD dwCreationFlags,                    // опции создания
    LPDWORD lpThreadId                        // идентификатор потока
);
```