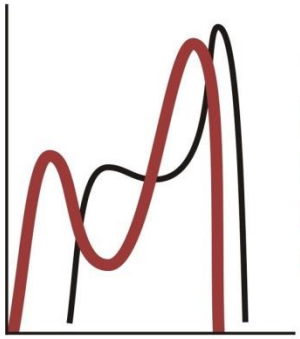


ИНСТИТУТ
МАТЕМАТИКИ
МЕХАНИКИ
КОМПЬЮТЕРНЫХ
НАУК

имени И.И. Воровича —

Операционные системы и компьютерные сети

Лекция 3. Управление процессами и потоками

Андреева Евгения Михайловна
доцент кафедры информатики и вычислительного эксперимента

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

План лекции

- Практическое задание
- Потоки исполнения
- Создание процессов Windows API/Posix
- Создание потоков Windows API/Posix

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Практическое задание №2

- Пример:

Написать сценарий, который принимает единственное имя файла как аргумент, и добавляет разрешения на выполнение для владельца, но только если это обычный файл. Сценарий должен проверять, что у него точно один аргумент, в противном случае должен вывести сообщение об использовании сценария.

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Написание сценария

- Основы написания сценариев
- Bash-скрипты
- Сценарий — это последовательность команд системы
- Можно сохранять в файлах с расширением .sh (script.sh)
- Для командной оболочки bash файл начинается со строки:

  #!/bin/bash
- Дальше идет последовательность команд
- Для запуска сценарию нужно дать права на выполнение

  chmod +x script1.sh
- Запустить на выполнение

  ./script1.sh

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Использование переменных

- Переменные окружения

  BASH=/bin/bash

  HOME=/home/student

  LANG=en

  ```
  #!/bin/bash
  echo "Home for the current user is: $HOME"
  ```

- Пользовательские переменные

  ```
  #!/bin/bash
  grade=5
  person="Adam"
  echo "$person is a good boy, he is in grade $grade
  ```

  (не должно быть пробелов вокруг «=»)

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Присвоение значений переменным

- Присваивание значения одной переменной другой

```
var2=$var1
```

- Присвоение результата вывода любой команды
  - С помощью значка обратного апострофа «`»

```
#!/bin/bash
mydir=`pwd`
echo $mydir
```

  - С помощью конструкции $()

```
#!/bin/bash
mydir=$(pwd)
echo $mydir
```

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Математические операции

- Для выполнения математических операций в файле скрипта можно использовать конструкцию вида $((a+b)):

```
#!/bin/bash
var1=$(( 5 + 5 ))
echo $var1
var2=$(( $var1 * 2 ))
echo $var2
```

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Передача параметров командной строки

- Сценарий с параметрами

  ./script.sh 10 20

- Чтение параметров

  $# — количество параметров

  $0 — имя скрипта

  $1 — первый параметр, $2 — второй параметр и так далее

  $* — все параметры, введённые в командной строке, в виде единого «слова»

  $@ — все параметры разбиты на отдельные «слова»

- Пример:

```
#!/bin/sh
echo "Сценарий: $0"
echo "Всего параметров: $#"
echo "Первый параметр: $1"
echo "Второй параметр: $2"
```

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Условный оператор

- Если код команды выполняется успешно, она возвращает нулевое значение, тогда содержимое условного оператора выполняется, а если ненулевое, то не выполняется.

```
if command
then
  commands
fi
```

- Пример

```
if grep $user /etc/passwd
then
  echo "The user $user Exists"
elif ls /home
then
  echo "The user doesn’t exist but anyway there is a directory under /home"
fi
```

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Проверка условий

```
if [ condition ]
then
        commands
Fi
```

- Нужны пробелы до и после условия condition.
- Выделяют три вида условий, которые можно проверять в квадратных скобках:
  - числовые (только для целых чисел);
  - строковые (сравнение строк);
  - файловые (существование, права доступа и пр.)
- Операторы сравнений описаны во вспомогательных материалах

Пример

```
#!/bin/bash
filename="$1"
if [ $# -ne 1 ]
then
  echo "Сценарий $0 должен использоваться с одним параметром"
elif [ -f $filename ]
then
  chmod u+x $filename
fi
```

Зачем нужны потоки

Ввести массив A

Ожидание ввода A

Ввести массив B

Ожидание ввода B

Ввести массив C

Ожидание ввода C

A=A+B

C=A+C

Вывести массив C

Ожидание вывода C

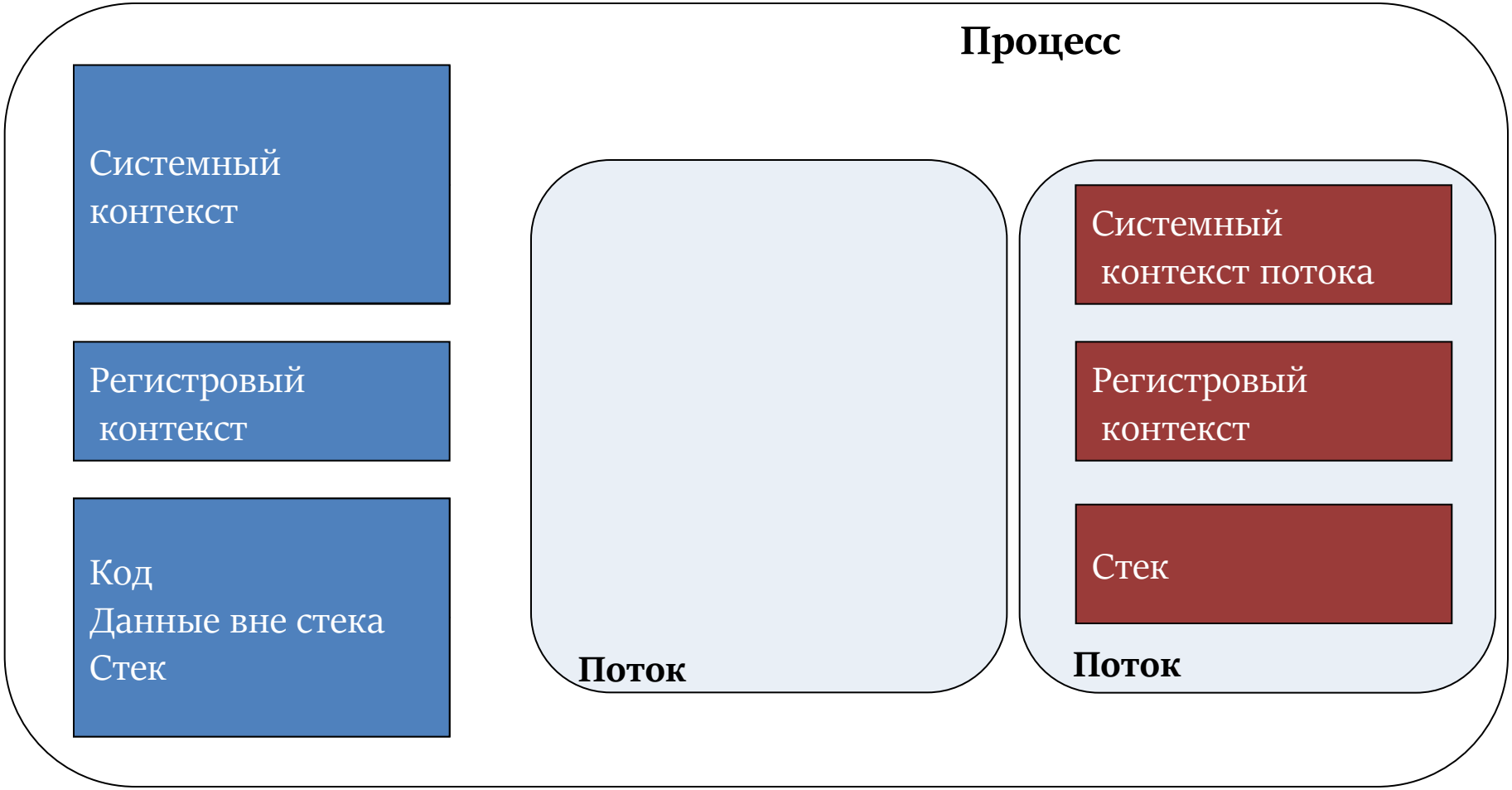
ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Решение 1

Процесс 1		Процесс 2
Создание процесса 2	Переключение контекста	
		Создание общей памяти
	Переключение контекста	
		Ожидание ввода A и B
Создание общей памяти		
Ввести массив A		
Ожидание ввода A		
Ввести массив B		
Ожидание ввода B		
	Переключение контекста	
Ввести массив C		
		A=A+B
Ожидание ввода C		
	Переключение контекста	
C=A+C		
Вывести массив C		
Ожидание вывода C		

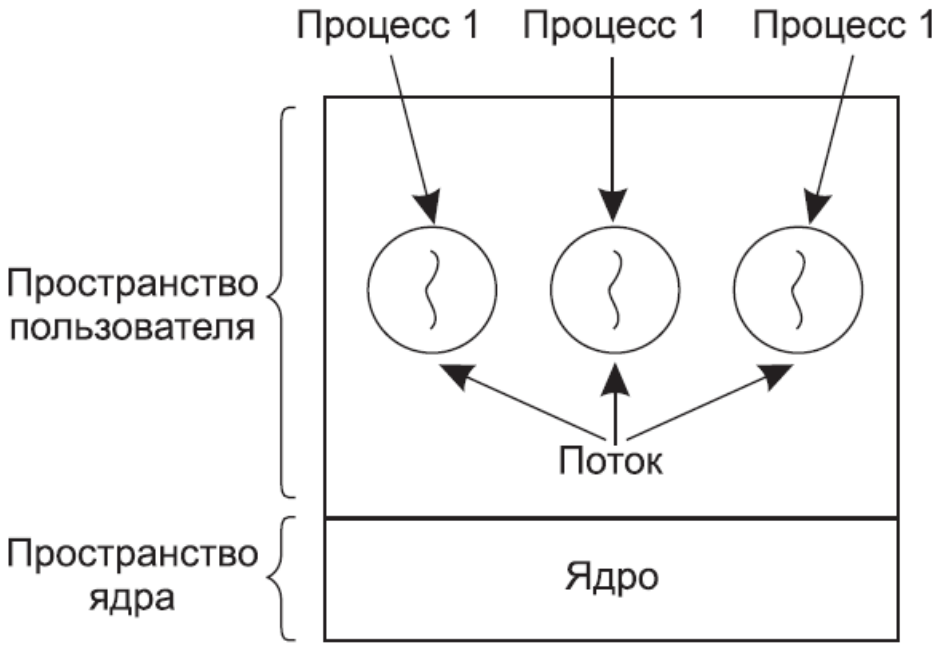
ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

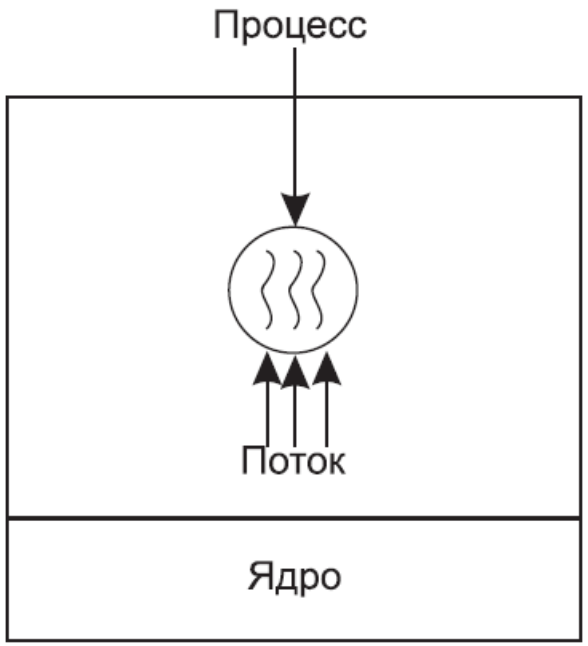
Потоки



ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Многопроцессорный и многопоточный режим





ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Решение 2

Поток 1		Поток 2
Создание потока 2		
	Переключение контекста	
		Ожидание ввода A и B
	Переключение контекста	
Ввести массив A		
Ожидание ввода A		
Ввести массив B		
Ожидание ввода B		
	Переключение контекста	
Ввести массив C		
		A=A+B
Ожидание ввода C		
	Переключение контекста	
C=A+C		
Вывести массив C		
Ожидание вывода C		

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

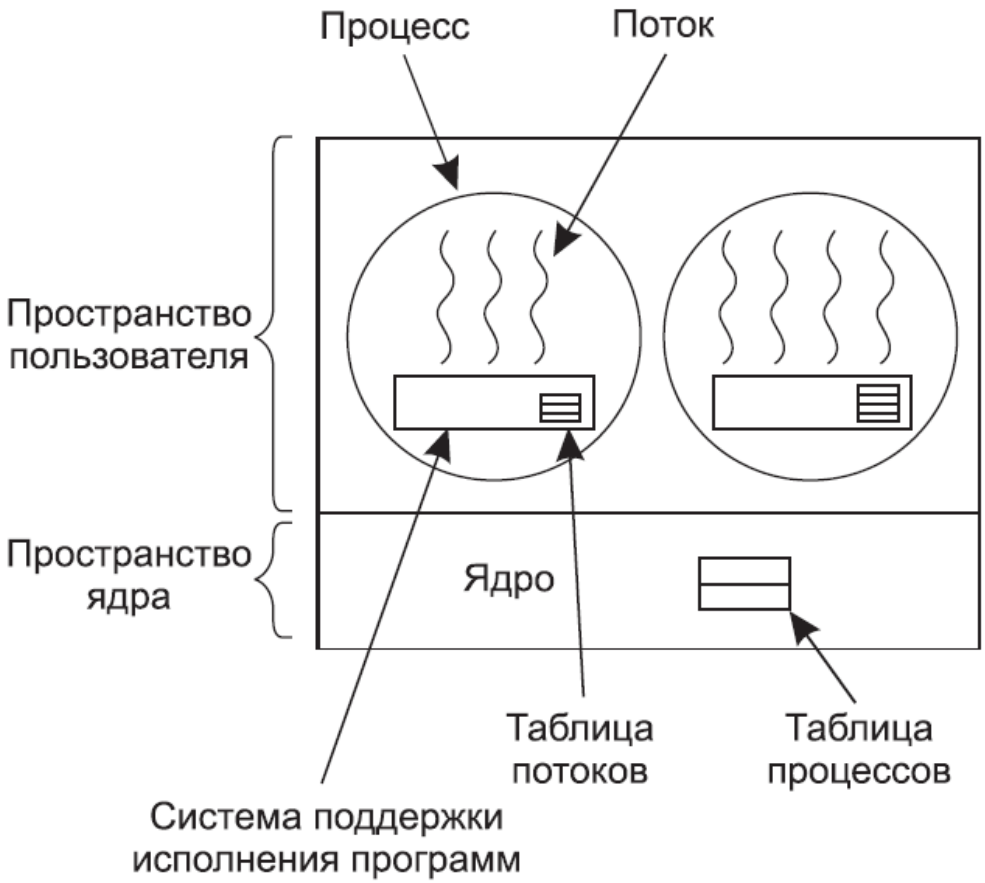
Состояния процесса

- Исполнение:
  - хотя бы один поток находится в состоянии «Исполнение»
- Готовность:
  - хотя бы один поток находится в состоянии «Готовность»
  - никто не находится в состоянии «Исполнение»
- Ожидание:
  - хотя бы один поток находится в состоянии «Ожидание»
  - никто не находится в состоянии «Исполнение» и «Готовность»
- Завершение:
  - Все потоки находятся в состоянии «Завершение»

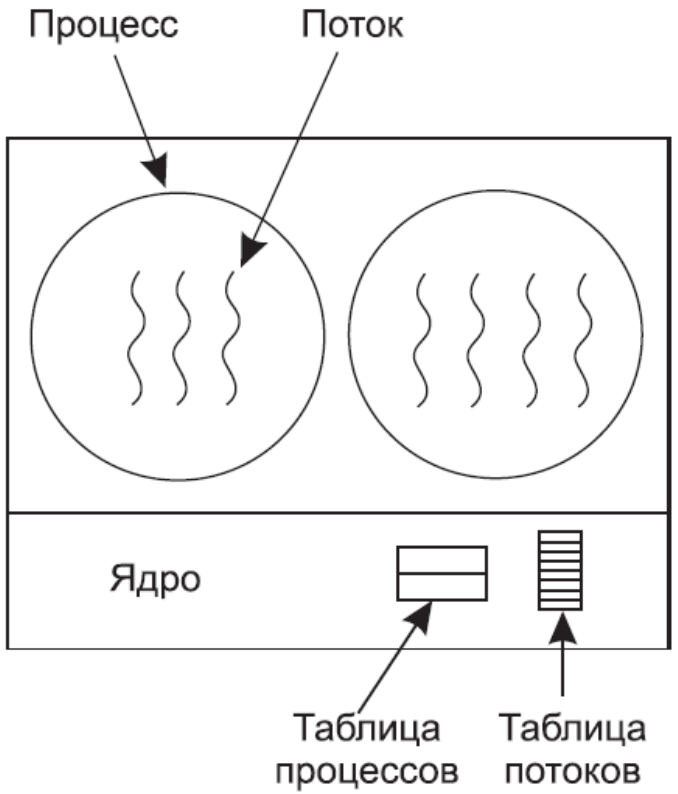
ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Реализация потоков

- **В пользовательском пространстве**



- **В ядре**



ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Реализация потоков в пользовательском пространстве

- Ядро о потоках ничего не знает
- Процедура, сохраняющая информацию о потоке, и планировщик являются локальными процедурами, и их вызов существенно более эффективен, чем вызов ядра
- При запуске одного потока ни один другой поток не будет запущен, пока первый поток добровольно не отдаст процессор
- Проблема реализации блокирующих системных запросов.

Реализация потоков в ядре

- Все вызовы, способные заблокировать поток, реализованы как системные
- Когда поток блокируется, ядро запускает другой поток
- Некоторые системы используют повторное использование потоков (потоки помечаются как нефункционирующие)
- Управление потоками в ядре не требует новых не блокирующих системных вызовов
- Основным недостатком управления потоками в ядре является существенная цена системных вызовов

Создание процессов в Windows API

- #include <windows.h>
- BOOL CreateProcess(

```
    lpApplicationName,                      // NULL
    LPTSTR lpCommandLine,                   // CommandLine
    LPSECURITY_ATTRIBUTES lpsaProcess,      // NULL
    LPSECURITY_ATTRIBUTES lpsaThread,       // NULL
    BOOL bInheritHandles,                   // FALSE
    DWORD dwCreationFlags,                  // CREATE_NEW_CONSOLE
    LPVOID lpEnvironment,                   // NULL
    LPCTSTR lpCurDir,                       // NULL
    LPSTARTUPINFO lpStartupInfo,            // &startup_info
    LPPROCESS_INFORMATION lpProcInfo)       // &process_information
```

Флаги

Флаги

CREATE_NEW_CONSOLE

CREATE_SUSPENDED

CREATE_UNICODE_ENVIRONMENT

Классы приоритетов

IDLE_PRIORITY_CLASS

NORMAL_PRIORITY_CLASS

HIGH_PRIORITY_CLASS

REALTIME_PRIORITY_CLASS

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Структура, хранящая информацию о процессе

- typedef struct PROCESS_INFORMATION {

  HANDLE hProcess;

  HANDLE hThread;

  DWORD dwProcessId;

  DWORD dwThreadId;

  } PROCESS_INFORMATION;

- Дескрипторы в структуре PROCESS_INFORMATION, когда они больше не нужны, должны быть закрыты функцией **CloseHandle**.

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Завершение процесса

- Функция **ExitProcess** - предпочтительный метод завершения процесса.

  VOID ExitProcess(UINT uExitCode);

- Функция **TerminateProcess** принудительно завершает работу заданного процесса и всех его потоков.

  BOOL TerminateProcess(HANDLE hProcess, UINT uExitCode);

Пример

```
#include <windows.h>
#include <iostream>
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))
	{
		std::cout << "process" << std::endl;
		std::cout << "handle " << pi.hProcess << std::endl;
		Sleep(10000);
		TerminateProcess(pi.hProcess, 1);
	 }
}
```

Создание процессов в Posix

- Функции создания процесса

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

- Функции замещения тела процесса

```
int execve(const char *file, char *const argv[], char *const envp[]);
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const char *path, const char *arg, ..., char *const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
```

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

Ожидание завершения процесса

```
#include <sys/wait.h>
pid_t wait(int *pnStatus);
pid_t waitpid(pid_t pid, int *pnStatus, int nOptions);
```

Проверка	Дополнительная информация
WIFEXITED(nStatus)	WEXITSTATUS(nStatus)
WIFSIGNALED(nStatus)	WTERMSIG(nStatus),
WIFSTOPPED(nStatus)	WSTOPSIG(nStatus)

Завершение работы процесса

- #include <stdlib.h>
- Вызовом библиотечной функции

  void exit(int status);
- Вызовом системного вызова

  void _exit(int status);
- Получением необрабатываемого, неигнорируемого и неблокируемого сигнала, который вызывает по умолчанию нормальное или аварийное завершение процесса.

  void abort(void);
- Принудительное завершение процесса (<signal.h>)

  int kill(pid_t pid, int sig);

Пример

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
int main(int argc, char * argv[]) {
int pid, status;
if (argc < 2) {
        printf("Usage: %s command, [arg1 [arg2]...]\n", argv[0]);
        return EXIT_FAILURE; }
printf("Starting %s...\n", argv[1]);
pid = fork();
if (!pid) {
        execvp(argv[1], &argv[1]); perror("execvp");
        return EXIT_FAILURE; // Never get there normally }
```

Пример (продолжение)

```
else { if (wait(&status) == -1) {
            perror("wait");
            return EXIT_FAILURE; }
        if (WIFEXITED(status))
            printf("Child terminated normally with exit code %i\n",
            WEXITSTATUS(status));
        if (WIFSIGNALED(status))
            printf("Child was terminated by a signal #%i\n",
            WTERMSIG(status));
        if (WIFSTOPPED(status))
            printf("Child was stopped by a signal #%i\n",
            WSTOPSIG(status));
      }
return EXIT_SUCCESS; }
```

Создание потоков в Windows API

- #include <windows.h>
- HANDLE WINAPI CreateThread(

  LPSECURITY_ATTRIBUTES lpThreadAttributes,

  SIZE_T dwStackSize,

  LPTHREAD_START_ROUTINE lpStartAddress,

  LPVOID lpParameter,

  DWORD dwCreationFlags,

  LPDWORD lpThreadId);
- DWORD WINAPI ThreadProc(LPVOID lpParameter)

Пример

```
#include <windows.h>
#include <iostream>
#include <conio.h>
DWORD WINAPI thread2(LPVOID t)
{
	std::cout << "Second thread\n"; return 0;
}
int main()
{
	std::cout << "First thread\n";
	HANDLE thread = CreateThread(NULL, 0, thread2, NULL, 0, NULL);
	for (int i = 0; i < 1000000; i++) {}
	std::cout << "More data from first thread\n";
	_getch();
	return 0;
}
```

Создание потоков в Posix

- #include <pthread.h>
- int pthread_create(

  pthread_t *pThread,

  const pthread_attr_t *pAttr,

  void *(* pStart)(void *),

  void *pArg);

Другие функции

- Ожидания потока

  int pthread_join(pthread_t thread, void ** pValue);

- Завершение потока

  void pthread_exit(void *retval);

- Досрочное завершение потока

  int pthread_cancel (pthread_t THREAD_ID);

Пример

```
#include <pthread.h>
 void* threadFunc(void* thread_data)
{
        pthread_exit(0);
}
 int main()
{
    void* thread_data = NULL;
    pthread_t thread;
    pthread_create(&thread, NULL, threadFunc, thread_data);
    pthread_join(thread, NULL);
    return 0;
}
```