Лекция 2. Интерфейс прикладных программ и процессы

Операционные системы

26 февраля 2016 г.

Стандартные потоки

Определение

Код возврата процесса: (exit status, exit code и т. д.) — целое число, возвращаемое процессом операционной системе при его завершении.

Стандартные потоки: (standard streams) — средства обмена текстовой информацией между процессом и стандартными устройствами ввода/вывода.

Поток	№	POSIX	C	C++
		unistd.h	stdio.h	iostream
Ввода	0	STDIN_FILENO	stdin	std::cin, std::wcin
Вывода	1	STDOUT_FILENO	stdout	std::cout, std::wcout
Ошибок	2	STDERR_FILENO	stderr	std::cerr, std::wcerr, std::clog, std::wclog

Таблица 1: стандартные потоки

Виды библиотек

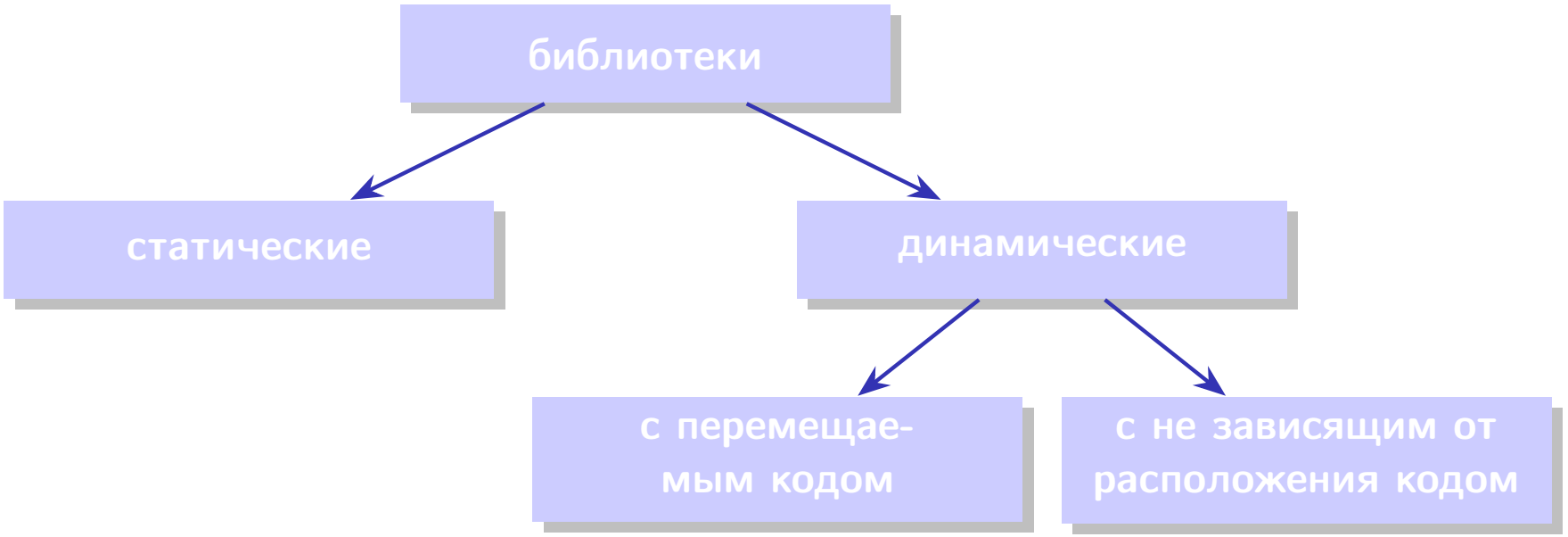

Рис. 1: виды библиотек

Создание библиотеки в ОС Linux

Пример

```
$ gcc -shared -fPIC -o libsample.so.0.0.1 sample.c
$ file libsample.so.0.0.1
libsample.so.0.0.1: ELF 64-bit LSB shared object, x86-64, ...
dynamically linked, ...
```

Подключение разделяемых библиотек

- статическое подключение при сборке;
- динамическая загрузка во время выполнения.

Определение зависимостей в ОС Linux

Пример

```
$ ldd llc
  linux-vdso.so.1 => ...
  libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 ...
  libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6 ...
  libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 ...
  libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 ...
  libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 ...
  /lib64/ld-linux-x86-64.so.2 ...
```

Определение зависимостей в ОС Windows

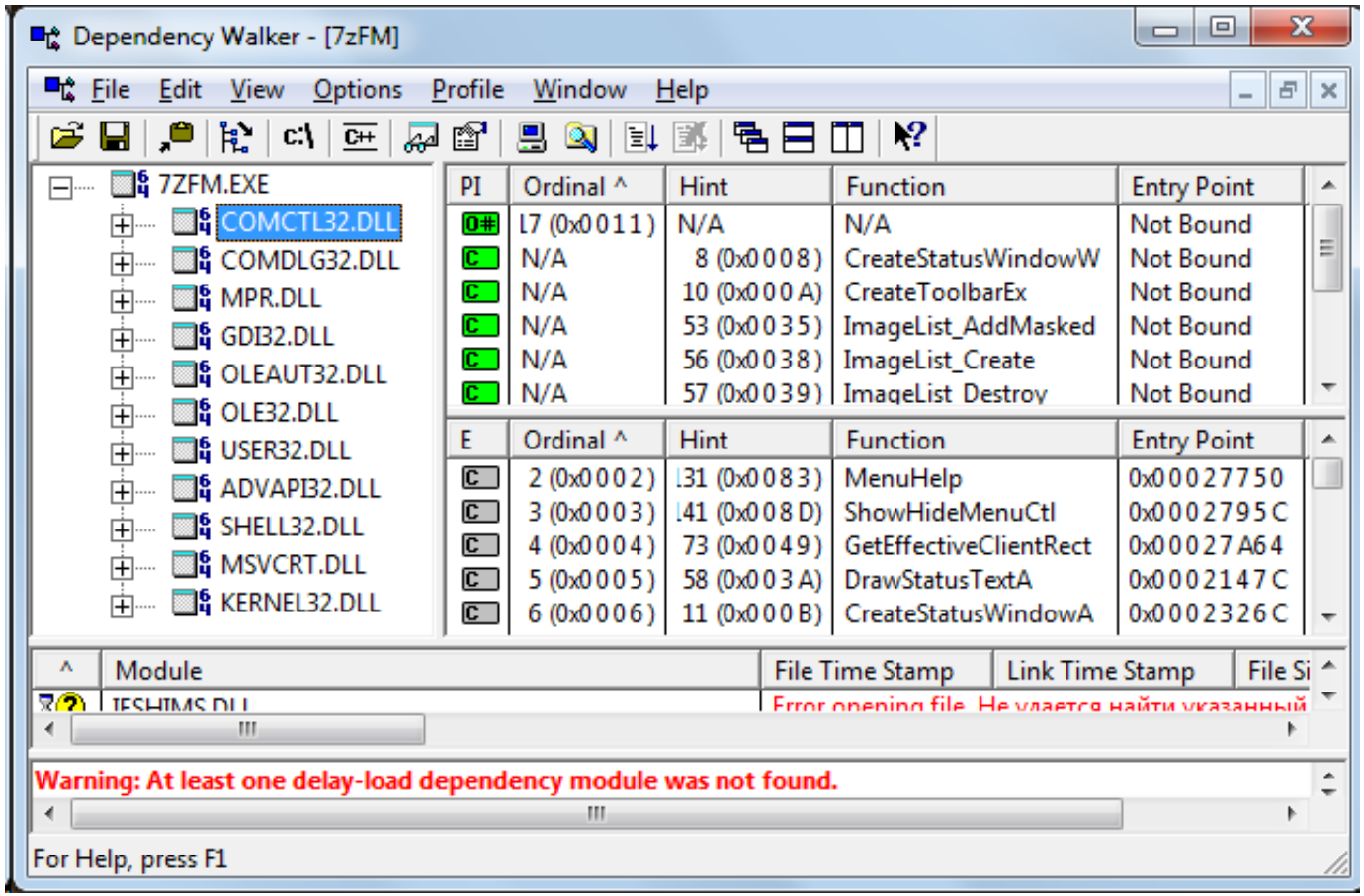

Рис. 2: Окно программы Dependency Walker

Проблемы разделяемых библиотек

Проблемы при использовании разделяемых библиотек (DLL Hell)

- Перезапись общих/системных библиотек в ранних версиях Windows при установке ПО;
- Зависимость работы ПО от системных настроек (`PATH`, `LD_LIBRARY_PATH`, ...);
- Удаление библиотек из системы;
- ...

Атрибуты процессов, определяемые при запуске

Атрибуты процессов, определяемые при запуске

- Аргументы командной строки;
- Переменные окружения (`VAR=value`);
- Текущий каталог.
- Атрибуты безопасности.

Пример

```
int main(int nArgC, char *apszArgV[], char *apszEnvP[])
{
  // ...
}
```

Возможные сценарии завершения процесса

Возможные сценарии завершения

- Добровольное завершение;
- Принудительное завершение (ошибка);
- Принудительное завершение другим процессом.

Пример

```
#include <stdlib.h>

int main()
{
  // ...
  if (bError)
    return EXIT_FAILURE;
  // ...
  return EXIT_SUCCESS;
}
```

Состояния процесса

Основные состояния процесса

- Создан;
- Ожидает выполнения;
- Выполняется;
- Заблокирован;
- Завершён.

Дополнительные состояния процесса (Unix)

- Выгружен и ожидает;
- Выгружен и заблокирован.

Прикладные программные интерфейсы

Стандарты API

- Windows API (Win32 API);
- POSIX.

Пример (вход/выход из режима ядра, X86)

	Ранние версии	Intel Pentium II/Linux 2.6
Вход	`int $0x80`	`sysenter`
Выход	`iret`	`sysexit`

Вызов функции API

Пример (C)

```c
if (MessageBeep(MB_ICONHAND))
  /* ... */;
```

Пример (Assembler X86)

```
    push    31h
    push    10h   ; MB_ICONHAND
    mov     eax, 1143h
    push    eax
    push    @f
    mov     edx, esp
    sysenter
@@: add     esp, 12
```

Создание процесса, Windows API

Функция создания процесса

```
BOOL WINAPI CreateProcess(
  __in_opt    LPCTSTR                lpApplicationName,
  __inout_opt LPTSTR                 lpCommandLine,
  __in_opt    LPSECURITY_ATTRIBUTES  lpProcessAttributes,
  __in_opt    LPSECURITY_ATTRIBUTES  lpThreadAttributes,
  __in        BOOL                   bInheritHandles,
  __in        DWORD                  dwCreationFlags,
  __in_opt    LPVOID                 lpEnvironment,
  __in_opt    LPCTSTR                lpCurrentDirectory,
  __in        LPSTARTUPINFO          lpStartupInfo,
  __out       LPPROCESS_INFORMATION  lpProcessInformation
);
```

Макросы APIENTRY, MB_OK, MB_ICONINFORMATION

Определение макроса APIENTRY

```
#define _stdcall __attribute__((stdcall))
// ...
#define WINAPI __stdcall
#define APIENTRY __stdcall
```

Макрос TEXT()

Определение макроса TEXT()

```
#ifdef UNICODE
#  define __TEXT(q) L##q
#else
#  define __TEXT(q) q
#endif
#define TEXT(q) __TEXT(q)
```

Пример

- TEXT("Hello World") → L"Hello World"
- TEXT("Hello World") →

Макрос TEXT()

Определение макроса TEXT()

```
#ifdef UNICODE
#  define __TEXT(q) L##q
#else
#  define __TEXT(q) q
#endif
#define TEXT(q) __TEXT(q)
```

Пример

- TEXT("Hello World") → L"Hello World"
- TEXT("Hello World") → "Hello World"

Пример

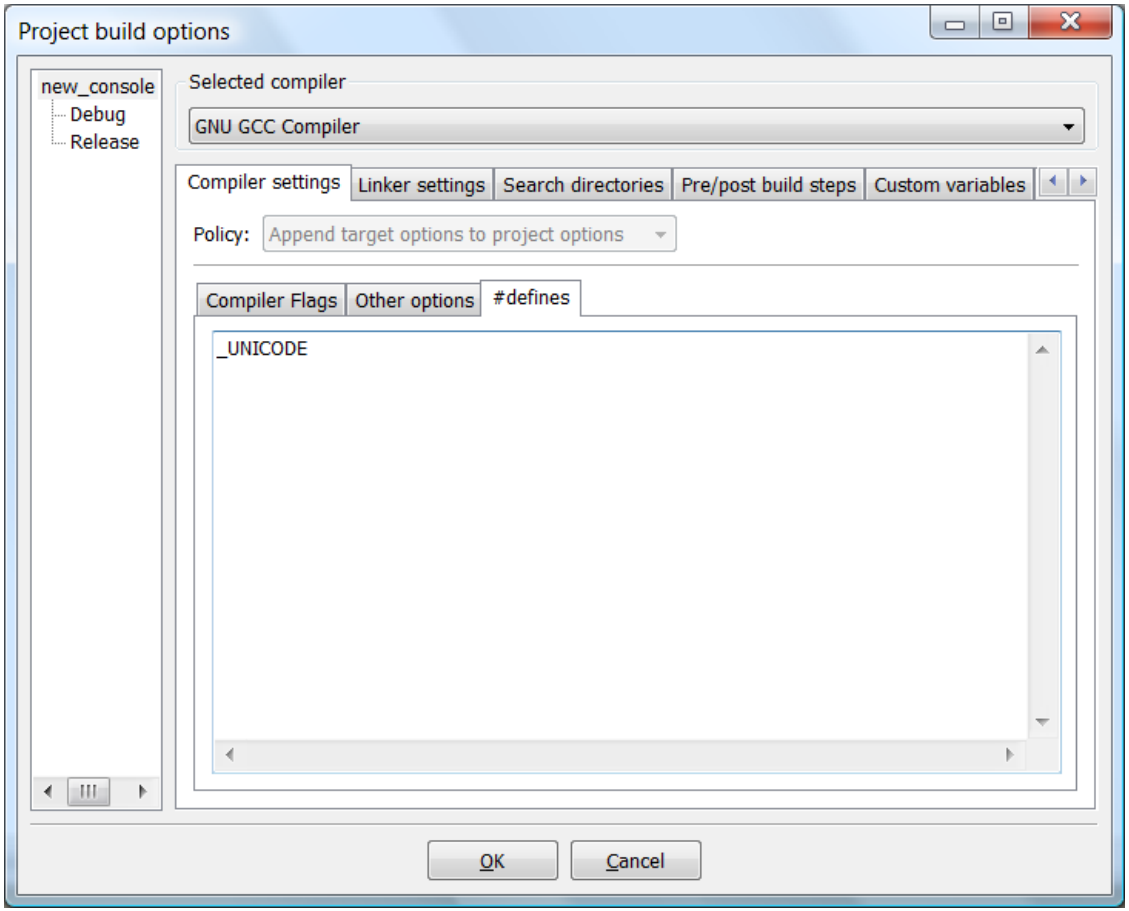

Рис. 3: определение символа _UNICODE

Создание процесса, Windows API (продолжение)

Различные флаги	Классы приоритета
CREATE_NEW_CONSOLE	IDLE_PRIORITY_CLASS
CREATE_SUSPENDED	NORMAL_PRIORITY_CLASS
CREATE_UNICODE_ENVIRONMENT	HIGH_PRIORITY_CLASS
	REALTIME_PRIORITY_CLASS

Таблица 2: Флаги создания процесса (dwCreationFlags)

Создание процесса, Windows API (окончание)

Структура, принимающая информацию о процессе

```
typedef struct _PROCESS_INFORMATION
{
  HANDLE hProcess;
  HANDLE hThread;
  DWORD  dwProcessId;
  DWORD  dwThreadId;
}
  PROCESS_INFORMATION, *LPPROCESS_INFORMATION;
```

Создание процесса

Пример

```
#include <windows.h>
#include <tchar.h>

#include <stdio.h>

int main()
{
  STARTUPINFO startup_info =
  {
    sizeof (STARTUPINFO), 0
  };
```

Создание процесса (продолжение)

Пример (продолжение)

```
//
PROCESS_INFORMATION process_information =
{
  INVALID_HANDLE_VALUE,          // hProcess
  INVALID_HANDLE_VALUE,          // hThread
  0,                             // dwProcessId
  0                              // dwThreadId
};
//
```

Создание процесса (продолжение)

Пример (продолжение)

```
BOOL bSuccess = CreateProcess(
  NULL,                                  // lpApplicationName
  _T("notepad WinProcess.cpp"),          // lpCommandLine
  NULL,                                  // lpProcessAttributes
  NULL,                                  // lpThreadAttributes
  FALSE,                                 // bInheritHandles
  CREATE_UNICODE_ENVIRONMENT |           // dwCreationFlags
    NORMAL_PRIORITY_CLASS,
  NULL,                                  // lpEnvironment
  NULL,                                  // lpCurrentDirectory
  &startup_info,                         // lpStartupInfo
  &process_information);                 // lpProcessInformation
```

Создание процесса (продолжение)

Пример (продолжение)

```
//
if (!bSuccess)
{
  DWORD dwError = GetLastError();
  LPVOID lpMsgBuf;
  //
  FormatMessage(
    FORMAT_MESSAGE_ALLOCATE_BUFFER |
      FORMAT_MESSAGE_FROM_SYSTEM,
    NULL, dwError,
    MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
    (LPTSTR) &lpMsgBuf, 0, NULL);
  //
```

Создание процесса (окончание)

Пример (продолжение)

```
    static TCHAR s_tszMsg[1024], s_tszOem[1024];
    _stprintf(
      s_tszMsg,
      _T("Error creating a process - %08Xh: %s\n"),
      dwError, (LPCTSTR) lpMsgBuf);
    //
    CharToOem(s_tszMsg, s_tszOem);
    _tprintf(s_tszOem);
  }    // if (!bSuccess)
  //
  CloseHandle(process_information.hProcess);
  CloseHandle(process_information.hThread);
}    // main()
```

Создание процесса, POSIX

Функции создания процесса (<unistd.h>)

```
#include <unistd.h>

pid_t fork(void);

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[]);
```

Завершение процесса

Функция завершения процесса, Windows API

```
VOID WINAPI ExitProcess(
  __in  UINT uExitCode
);
```

Функции завершения процесса, POSIX (<stdlib.h>, <unistd.h>)

```
void  exit(int status);    /* EXIT_SUCCESS, EXIT_FAILURE */
void _Exit(int status);    /* C99 */
void _exit(int status);
```

Принудительное завершение процесса

Функция завершения процесса, Windows API

```
BOOL WINAPI TerminateProcess(
  __in  HANDLE hProcess,
  __in  UINT   uExitCode
);
```

Функция завершения процесса, POSIX (<signal.h>)

```
int kill(pid_t pid, int sig);
```

Сигнал

Определение

Сигнал: средство межпроцессного взаимодействия в POSIX-совместимой ОС. Представляет собой асинхронное сообщение процессу или потоку, прерывая его на неатомарной операции. При появлении сигнала в контексте процесса-получателя вызывается обработчик сигнала.

Виды сигналов

Имя	№	Клавиши	Перехват	Значение
SIGKILL	9	—	✗	немедленное завершение
SIGINT	2	Ctrl + C	✓	прерывание
SIGTSTOP		—	✗	временный останов
SIGTSTP		Ctrl + Z	✓	временный останов с терминала
SIGCONT		—	✓	продолжение после SIGTSTOP или SIGTSTP
SIGSEGV		—	✓	неправильный доступ к памяти
SIGILL		—	✓	неправильная машинная инструкция
SIGFPE		—	✓	неправильная арифметическая операция
SIGPIPE		—	✓	запись в канал, из которого никто не читает

Таблица 3: некоторые часто используемые сигналы POSIX

Ожидание дочернего процесса

Функции ожидания процесса (<sys/wait.h>)

```
pid_t wait(int *pnStatus);
pid_t waitpid(pid_t pid, int *pnStatus, int nOptions);
```

Проверка состояния завершения процесса

- WIFEXITED(status) (WEXITSTATUS(status))
- WIFSIGNALED(status) (WTERMSIG(status), WCOREDUMP(status))
- WIFSTOPPED(status) (WSTOPSIG(status))
- WIFCONTINUED(status)

Создание процесса

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;
```

Создание процесса (продолжение)

Пример (продолжение)

```
if (argc < 2)
{
  printf("Usage: %s command, [arg1 [arg2]...]\n", argv[0]);
  return EXIT_FAILURE;
}
printf("Starting %s...\n", argv[1]);
pid = fork();
```

Создание процесса (продолжение)

Пример (продолжение)

```
if (pid == 0)
{
  execvp(argv[1], &argv[1]);
  perror("execvp");
  return EXIT_FAILURE;    // Never get there normally
}
else
{
  if (wait(&status) == -1)
  {
    perror("wait");
    return EXIT_FAILURE;
  }
```

Создание процесса (продолжение)

Пример (продолжение)

```
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 (WCOREDUMP(status))
    printf("Child dumped a core\n");
}
```

Создание процесса (окончание)

Пример (окончание)

```
    if (WIFSTOPPED(status))
      printf(
        "Child was stopped by a signal #%i\n",
        WSTOPSIG(status));
  }
  return EXIT_SUCCESS;
}
```

Работа создания процесса

Пример

```
gcc -o exec exec.c
```

Пример (вывод)

```
$ ./exec ./exec ls -l
Starting ./exec...
Starting ls...
итого 16
-rwxr--r-- 1 stu003 stu003   39 2011-09-10 03:43 build.sh
-rwxr-xr-x 1 stu003 stu003 7316 2011-09-10 03:43 exec
-rw-r--r-- 1 stu003 stu003  979 2006-10-23 01:44 exec.c
Child terminated normally with exit code 0
Child terminated normally with exit code 0
```

Дерево процессов

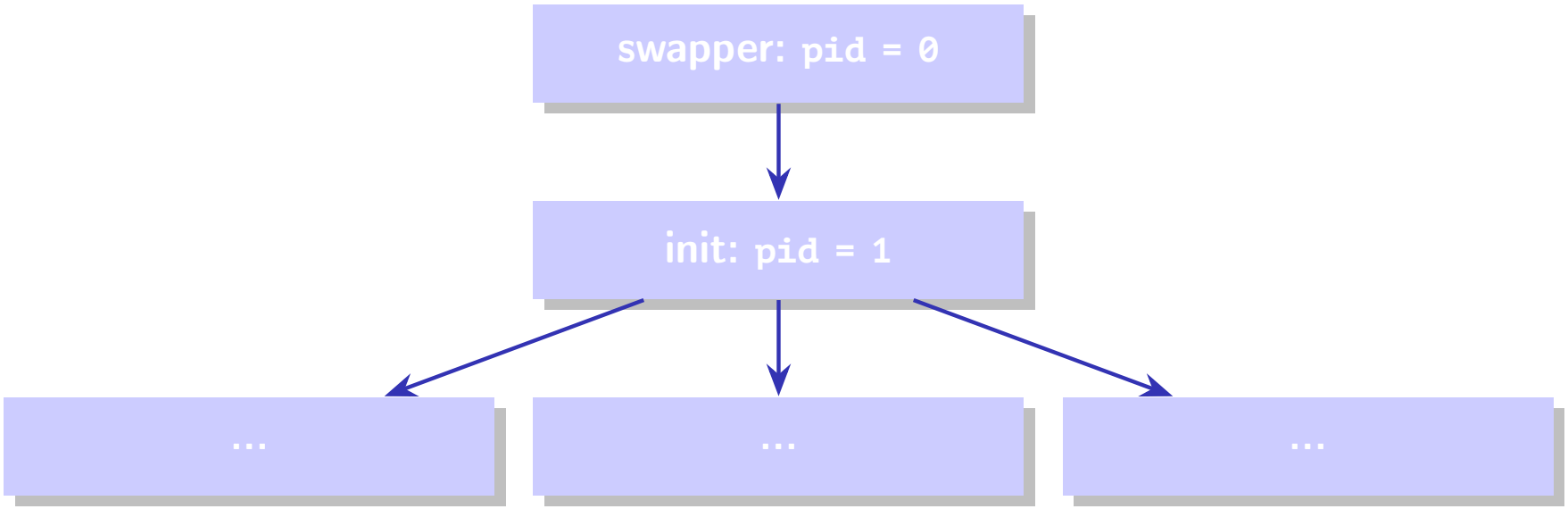

Рис. 4: дерево процессов в Linux и других ОС

Вывод дерева процессов в ОС Linux

Пример

```
$ pstree -p -A
init(1)-+-NetworkManager(506)-+-dhclient(591)
        |                     |-{NetworkManager}(518)
        |                     `-{NetworkManager}(592)
        |-accounts-daemon(1077)---{accounts-daemo}(1078)
        |-acpid(755)
        |-atd(762)
        |-avahi-daemon(500)---avahi-daemon(507)
        |-bamfdaemon(1920)---{bamfdaemon}(1928)
        |-bluetoothd(840)
        |-colord(962)-+-{colord}(974)
        |             `-{colord}(1046)
        ...
```

Вывод дерева процессов в ОС Windows

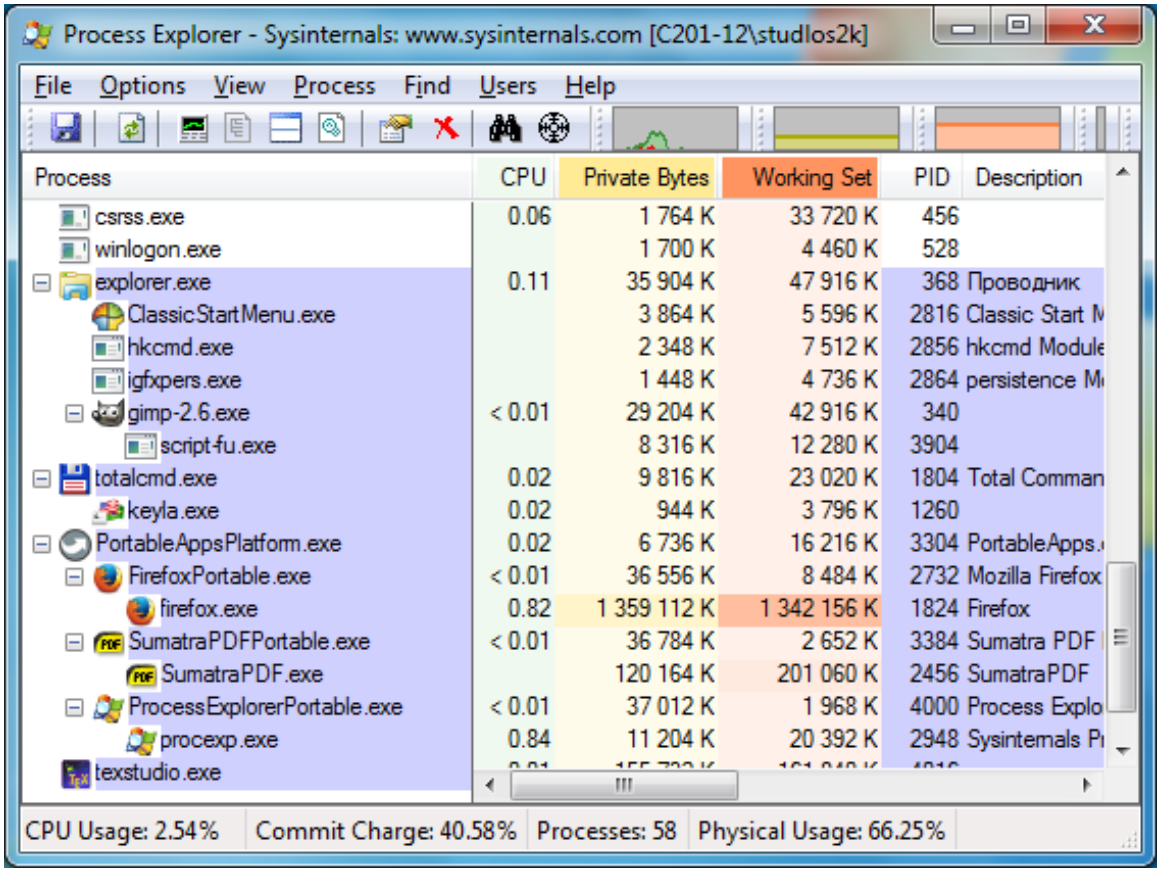

Рис. 5: программа Process Explorer (`http://sysinternals.com`)