

Работа с процессами

- Способы создания процессов
- Передача информации при запуске файла на исполнение
 - аргументы командной строки
 - переменные окружения
- Создание и завершение процессов в ОС Windows
 - функции CRT
 - функции WinAPI
- Создание и завершение процессов, запуск исполняемых файлов в ОС Unix

Способы создания новых процессов

- Новый процесс может создаваться двумя способами:
 - **запуск на исполнение файла** (ОС Windows) — этот способ прост для программиста, но лишает его некоторых возможностей;
 - **создание копии существующего процесса** (ОС Unix) — сложнее, однако шире возможности передачи информации дочернему процессу. В этом случае запуск файла на исполнение выполняется отдельной командой.

Передача параметров при запуске файла на исполнение

- Часто при запуске файла, выполняющего какие-либо действия, необходимо передать ему параметры этих действий.
- Например архиватору требуется знать команду (упаковать или распаковать?) и файлы для обработки.
- Автоматическая передача этих параметров позволяет избежать дублирования их ввода пользователем (этим пользуются например, программы типа Total Commander, вызывая архиваторы для выбранных пользователем файлов)

Аргументы командной строки

- **Аргументы командной строки** это строки, записываемые через пробел после имени файла при запуске его из командной строки.
- Аргументы разделяются между собой *пробелами*. Если один аргумент должен содержать внутри пробелы (или другие специальные символы), его следует взять в двойные кавычки.
- Первым аргументом считается имя запускаемого файла.

Задание

- Программа `unzip` при запуске получает следующие аргументы командной строки (кроме имени файла):
 - имя архива для распаковки;
 - путь, по которому следует вести распаковку.
- Вызовите программу `unzip` для распаковки файла `archive.zip`, поместив результат в каталог `C:\Program Files\Incoming`

Аргументы командной строки

```
unzip archive.zip "C:\Program  
Files\Incoming"
```

Получение аргументов командной строки

- Программа на языке C/C++ получает аргументы командной строки как аргументы функции `main`:
 - `int` – код завершения процесса
- `main` (
 - `int argc` – количество аргументов командной строки,
 - `char *argv[]` – массив аргументов командной строки)

Задание

Напишите программу, распечатывающую на экран все аргументы командной строки (каждый аргумент выводится на отдельной строке)

Работа с аргументами командной строки

```
int main(int argc, char *argv[])
{
    int i;
    for (i=0; i<argc; i++)
    {
        printf("%s\n", argv[i]);
    }
    return 0;
}
```

Переменные окружения

- **Переменные окружения** (environment variables) — это набор строк типа <имя>=<значение>, которые описывают среду, в которой работает программа.
- В командной строке с переменными окружения работает команда `set`:
 - без аргументов она показывает все переменные окружения и их значения
 - `set <имя>` - показывает значение конкретной переменной
 - `set <имя>=<значение>` - изменяет значение переменной

Переменные окружения из программы

- Программа может получить массив переменных окружения в качестве третьего аргумента функции `main`, однако работать с ним неудобно — массив содержит строки типа `<имя>=<значение>`.
- Для работы с переменными окружения обычно используют функции CRT `getenv` и `_putenv`.

Функции работы с переменными окружения

- `char *` – значение переменной
- `getenv (`
 - `const char *` – имя переменной`)`
 - `int` — 0 при успешном завершении
- `_putenv (`
 - `const char *` – строка типа
 `<ИМЯ>=<значение>``)`

Работа с переменными окружения

- Функция `_putenv` меняет копию значения переменной, доступную программе. Она не изменяет «внешнего» значения переменной окружения. Однако измененное значение может быть унаследовано дочерним процессом.
- При запуске на исполнение файла вы можете дать новому файлу унаследовать текущие значения переменных окружения, или задать для него полностью новый блок переменных окружения.

Задание

- Напишите программу, распечатающую на экран стандартные пути для поиска файлов, выводя каждый путь в отдельной строке.
- Стандартные пути для поиска файлов содержаться в переменной окружения PATH, разделяемые точкой с запятой.

Работа с переменными окружения

```
int main(int argc, char *argv[]) {  
    char path[500];  
    strcpy(path, getenv("PATH"));  
    for(i=0; i<strlen(path); i++) {  
        if(path[i] == ';')  
            path[i] = '\\n';  
    }  
    puts(path);  
    return 0;  
}
```

Создание процессов в ОС Windows CRT

- Для создание нового процесса с запуском исполняемого файла в C Run-time Library ОС Windows используется семейство функций `_spawn`.
- Все функции этого семейства начинаются со слова `_spawn` и содержат дополнительные буквы, указывающие особенности их работы.
- Функция может содержать от одной до трех букв: `_spawnl` или `_spawnvpe`.

Базовые параметры функции `_spawn`

- Каждая из функций семейства `_spawn` имеет два обязательных аргумента:
 - `int` — режим запуска:
 - `_P_WAIT` — функция возвращается только после завершения созданного процесса;
 - `_P_NOWAIT` — создаваемый процесс выполняется параллельно с текущим;
 - `_P_DETACH` — создаваемый процесс выполняется параллельно, он лишен доступа к консоли порождающего процесса.
 - `const char *` - имя файла (возможно с путем) для исполнения.
- Другие параметры зависят от вида функции.

Семейство функций `_spawn`

- Дополнительные буквы (в порядке указания):
 - `l` (`List`) — аргументы командной строки передаются списком строк, через запятую, признаком конца служит нулевой указатель (несовместима с `v`, однако одну из этих букв указывать обязательно)
 - `v` (`Var`) — аргументы командной строки передаются массивом строк, который должен завершаться нулевым указателем (обычно применяется если количество аргументов не фиксировано)

Семейство функций `_spawn`

- Дополнительные буквы (в порядке указания):
 - `p` (`Path`) — если имя файла указано без пути и он отсутствует в текущем каталоге, то функция с буквой `p` будет искать его по путям, заданным в переменной окружения `PATH`
 - `e` (`Environment`) — последним аргументом программы является массив строк (указателей на символ), задающий блок переменных окружения для дочернего процесса, массив должен завершаться нулевым указателем

Задание

Запустите файл `unzip.exe` с параметрами командной строки `archive.zip` и `C:\Incoming`, с поиском исполняемого файла по путям, заданным в переменной окружения `PATH`, сохраняя текущие значения переменных окружения.

Программа должна продолжить работу только по окончании распаковки.

Семейство функций _spawn

```
_spawnlp(_P_WAIT, "unzip.exe",  
        "unzip.exe", "C:\\Incoming", NULL);
```

//или

```
char argv[3][40]=  
    {"unzip.exe", "C:\\Incoming", NULL};  
_spawnvp(_P_WAIT, "unzip.exe", argv);
```

Функции, вызываемые при завершении процесса

- С помощью функции `atexit` можно зарегистрировать функции, которые будут вызываться при нормальном завершении процесса.
- Регистрируемые функции не получают параметров и не возвращают значений
- Каждый вызов `atexit` регистрирует одну функцию. Если зарегистрировано несколько функций, то они выполняются в порядке, обратном порядку регистрации.

Функции, вызываемые при завершении процесса

- `int` — 0 при успешном завершении
- `atexit(`
 - `void (__cdecl *func) (void)` - имя функции для регистрации`)`

Завершение процесса в CRT

- Завершение процесса в CRT происходит при следующих условиях:
 - завершилась функция `main` (если эта функция имеет тип `int`, то кодом завершения процесса является возвращаемое ее значение, если `void` — то 0, общепринятый признак нормального завершения процесса);
 - выполнялась функция `exit` (`_exit`), код завершения является единственным параметром функции;
 - произошла невосстановимая ошибка в программе, либо пользователь прервал процесс.(например нажатием Ctrl-C в его консоли).

Завершение процесса в CRT

- Функции завершения процесса:
- `void exit (`
 - `int` — код завершения процесса`)`
- Функция `_exit` аналогична `exit`, но она вызывает экстренный выход из программы, без записи информации из буфера в файл и вызова функций, зарегистрированных `atexit`

Создание процесса в Windows API

- Создание процессов в Win32 API происходит с помощью функции `CreateProcess`
 - `BOOL` — 0 при ошибке, ненулевое значение при успехе
- `CreateProcess` (
 - `LPCTSTR` — путь (с именем) к файлу для запуска, может быть `NULL`,
 - `LPTSTR` — аргументы командной строки, может быть `NULL`,
 - `LPSECURITY_ATTRIBUTES` — атрибуты безопасности процесса, обычно `NULL`

Создание процесса в Windows API

- LPSECURITY_ATTRIBUTES — атрибуты безопасности потока, обычно NULL,
- BOOL — наследовать ли дескрипторы; дескрипторы должны быть созданы как наследуемые чтобы наследование состоялось,
- DWORD — флаги:
 - CREATE_NEW_CONSOLE — новый процесс получает отдельную консоль, а не наследует родительскую,
 - DETACHED_PROCESS — новый процесс запускается без консоли
- LPVOID — указатель на массив со строками переменных окружения, массив заканчивается двойным нулем, может быть NULL

Создание процесса в Windows API

- LPCTSTR — текущий каталог для создаваемого процесса, может быть NULL
 - LPSTARTUPINFO — указатель на структуру, содержащую данные о начальном размере окна запускаемого приложения и т.д.,
 - LPPROCESS_INFORMATION — указатель на структуру, куда будут записаны данные о созданном процессе — например его дескриптор
-)

Завершение работы процесса в WinAPI

– void

- `ExitProcess (`

- `UINT` — код завершения

- `BOOL` — 0 при неудаче

- `TerminateProcess (`

- `HANDLE` — дескриптор процесса,

- `UINT` — код завершения

Дескрипторы процессов в ОС Windows

- **Дескриптор** (HANDLE) — это переменная, указывающая на открытый процессом объект (файл, другой процесс и т.д.)
- При запуске дочернего процесса родительский получает дескриптор на него через структуру `PROCESS_INFORMATION`.
- Дескриптор может быть использован для управления процессом, а также для запроса статистической информации о нем (код завершения, время работы и т.д.)

Дескрипторы процессов в ОС Windows

- Процесс продолжает существовать после своего завершения, хотя большинство ресурсов его (память, открытые файлы) освобождается.
- Это требуется для того, чтобы можно было узнать код завершения и другую информацию о работе процесса.
- Эта информация удаляется, когда закрывается последний дескриптор, указывающий на этот процесс.

Идентификаторы процессов в ОС Unix

- В ОС Unix каждый процесс характеризуется идентификатором, уникальным в пределах системы.
- Этот идентификатор имеет тип `pid_t`, который является производным от одного из целых чисел.
- Процесс может узнать у системы два идентификатора:
 - `pid_t getpid(void)` — возвращает идентификатор текущего процесса;
 - `pid_t getppid(void)` — возвращает идентификатор родительского процесса

Создание процесса в ОС Unix

- Создание процесса в ОС Unix осуществляется функцией `fork`.
- Функция `fork` создает копию текущего процесса и при успешном выполнении возвращается дважды: в родительский и дочерний процессы.
- Определить то, в каком процессе вы находитесь, можно анализируя значение, возвращаемое `fork`.

Создание процесса в ОС Unix

- `pid_t` — возвращается следующим образом:
 - -1 при ошибке (невозможно создать новый процесс)
 - 0 — возвращается в процесс-ребенок
 - положительное значение — идентификатор процесса-ребенка — возвращается в родительский процесс
- `fork(void)`

Запуск файла на исполнение

- Запуск файла на исполнение в ОС Unix осуществляется с использованием функций семейства `exec`.
- Функции семейства `exec` имеют добавочные буквы, аналогичные функциям семейства `_spawn`, их параметры аналогичны за исключением того, что первый параметр (режим) отсутствует.
- Функция семейства `exec` не возвращает в существующий процесс, она уничтожает его содержимое и заменяет на запускаемую программу.

Наследование параметров при порождении процесса и запуске файла

- При выполнении `exec` процесс сохраняется, меняется лишь выполняемая программа, поэтому следующие параметры остаются неизменными:
 - идентификаторы процесса и родительского процесса;
 - идентификаторы группы и сеанса;
 - идентификаторы пользователя и группы пользователей;
 - рабочий каталог;
 - файловые дескрипторы открытых файлов.

Задание

- Создайте новый процесс и запустите в нем на исполнение файл `child` с параметрами командной строки `Carl Philip`, и единственной переменной окружения `SURNAME` со значением `Bach`. Поиск по путям не производить.
- После запуска дочернего процесса родительский должен вывести сообщение `I'm still there!`
- При невозможно создать дочерний процесс в поток ошибок выводится сообщение `Fork failed.`

Создание процесса и запуск файла на исполнение

```
pid_t pid = fork();  
if(pid == -1) {  
    printf("Fork failed:");  
} else if (pid == 0) {  
    char *env[50]=  
        {"SURNAME=Bach", NULL};  
    execle("child", "child", "Carl",  
        "Philip", NULL, env);  
} else {  
    printf("I'm still there!\n");  
}
```

Идентификаторы процессов

- ОС Unix не содержит дескрипторов процессов, данные о завершении работы процесса запрашиваются через идентификатор.
- В отличие от дескриптора, идентификатор не открывается и не может быть закрыт.
- Данные о процессе удаляются, когда завершается породивший его процесс, либо они запрошены через функцию `wait` или `waitpid`
- Завершившийся процесс, который еще не удален, называется в ОС Unix зомби-процессом