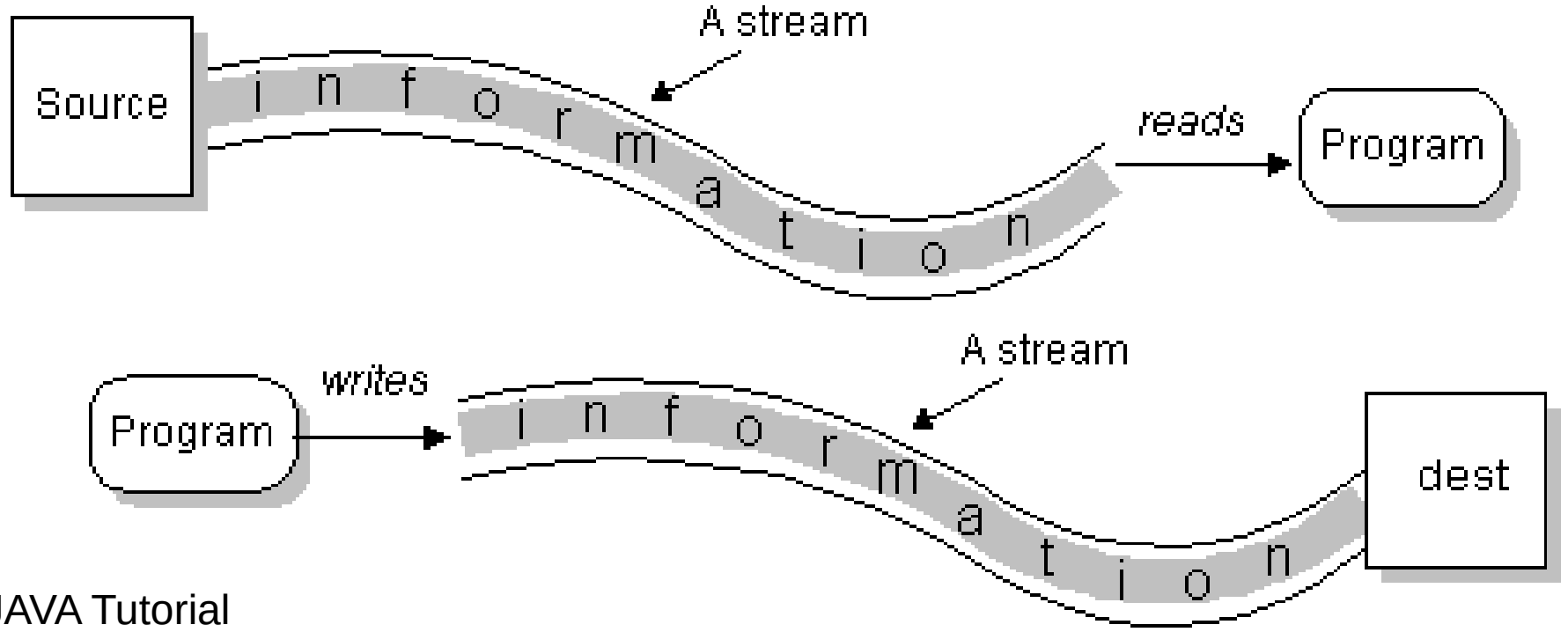
Работа с файлами

Основы языка C, лекция 7

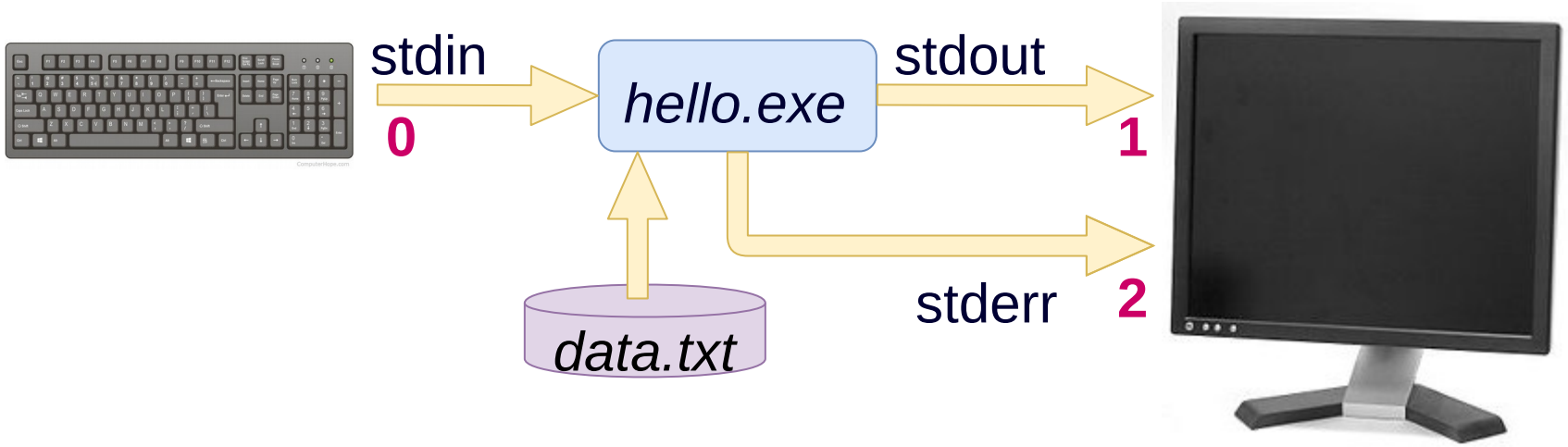
stream (поток данных)

- ~~thread —~~поток (нить) исполнения
- stream — абстракция, производит или потребляет информацию
- Односторонний последовательный поток данных. с дуплексными потоками пока не будем работать



stdin, stdout, stderr

- С каждой программой при старте связаны потоки
  - stdio.h — ОС kernel
  - stdin (standard input) — file descriptor = 0
  - stdout (standard output) — file descriptor = 1
  - stderr (standard error) — file descriptor = 2
  - scanf, printf, fread, fwrite — read, write



Перенаправление потоков

- Командная строка (shell)
- stdout перенаправить в файл out.txt (перезаписать)
  hello.exe > out.txt
- stdout перенаправить в файл out.txt
  (добавить в конец)
  hello.exe >> out.txt
- на stdin подать содержимое файла data.txt
  hello.exe < data.txt
- Конвейер (pipe) - перенаправить выход stdout программы hello.exe на stdin программы convert.exe
  hello.exe | convert.exe

С указанием file descriptor

- **FileDescriptor>**
- Вывести содержимое файла foo.txt на экран
  cat foo.txt
- Перенаправить содержимое в файл out.txt, stdout — в файл
  cat foo.txt 1> out.txt
- Перенаправить сообщения об ошибках в файл out.txt, stderr — в файл
  cat nop.txt 2> error.txt
  cat error.txt
  cat: nop.txt: No such file or directory

2 потока в один файл

- **&FileDescritor**
- Перенаправь stderr на stdout, и stdout перенаправить
  cat foo.txt > output.txt 2>&1
- cat output.txt
  foo
  bar
- **cat nofile.txt > output.txt 2>&1**
- cat output.txt
  cat: nofile.txt: No such file or directory
- **hello.exe 1>&2**

Полезные утилиты

- Увидеть сообщения об ошибках постранично
  gcc hello.c 2>&1 | **less**
- Фильтрация
  hello.exe | **grep** "ERROR"
- Сортировка
  hello.exe | **sort**
- Подсчет строк, слов и символов
  hello.exe | **wc**
- **man wc**

Стандартные функции

- Стандартные функции — man 3
  fopen, fclose, fscanf, scanf, fprintf, printf...
- Системные вызовы — man 2
  В курсе IPC
  open, close, read, write
- Не смешивать системные вызовы и стандартные функции

Открытие и закрытие потока

- FILE * **fopen** (const char * **path**, const char * **mode**);
  открывает новый поток для файла path
- FILE * **freopen** (const char * **path**, const char * **mode**,
  FILE * **stream**);
  закрывает stream (если открыт) и открывает новый
- FILE * fdopen (int fd, const char * mode);
- int **fclose** (FILE * **stream**); закрывает stream
- int **feof** (FILE * **stream**);
  поверяет, что в потоке stream достигнут **состояние** EOF
- **EOF** — End Of File (конец файла)

path — путь к файлу

- Абсолютный
  "/home/gr978/st97801/hello.c"
- Относительный
  "../work/t1.c"
  "data.txt"
- Если у вас Windows:
  - пишем в cmd.exe
    "C:\Users\natasha\hello.c"
  - пишем строковую константу в C (экранируем \)
    "C:\\Users\\natasha\\hello.c"

mode — режим открытия файла

mode	что сокращает	Значение
r	read	чтение
w	write	запись (обрезать до 0)
a	append	писать в конец
r+	rw	Дополнительные режимы
w+	rw	
a+	ra	
rb	binary *в UNIX игнорируются*	\r\n → \n \n → \r\n таких преобразований нет
wb		
ab		
rb+		комбинации и далее

fopen mode = open flags

fopen mode	open flags
r	O_RDONLY
w	O_WRONLY | O_CREATE | O_TRUNC
a	O_WRONLY | O_CREATE | O_APPEND
r+	O_RDWR
w+	O_RDWR | O_CREATE | O_TRUNC
a+	O_RDWR | O_CREATE | O_APPEND

Какую моду выберем для

- Чтения входных данных (матрица для вычисления)
- Записи результата работы программы
- Записи в один файл результата нескольких запусков программы
  (для сравнения, логирование, накопление данных)
- Правка в файле части данных
  (изменяем файл конфигурации)

Копирование файла

- // студенческая версия

```
int c;
FILE * fin  = fopen("a.txt", "r");
FILE * fout = fopen("b.txt", "w");

while ( (c = fgetc(fin)) != EOF )
    fputc(c, fout);

fclose(fin);
fclose(fout);      // точно дописали?
```

Открытие файла

- fopen возвращает NULL, если не может открыть файл
- Еще хуже:

```
fin = fopen ("a.txt", "r");
if (fin == NULL) {
    printf("Нет файла\n");
    return 1;
}
```

- Что может пойти не так?
  - нет файла
  - файл есть, но нет прав на чтение

Лучше без диагностики, чем с неправильной диагностикой

Обработка ошибок <errno.h>

- #include <stdio.h>
  void **perror** (const char * **msg**);
  Печатает сообщение об ошибке
  msg : системное сообщение об ошибке
  - msg = NULL или "" - просто системное сообщение
- #include <errno.h>
  const char * const **sys_errlist** [ ];
  int **sys_nerr**;
  int **errno**;
- #include <string.h>
  char * **strerror** (int errnum);

Обработка ошибок

- FILE * fin = fopen ("b.txt", "r");
  if (fin == NULL) {
  perror("b.txt");
  exit (1); // exit (errno);
  }
- #include <stdlib.h>
  void **exit** (int status);
  Останавливает процесс (у нас — программа) и возвращает status & 0377
  Из main можно return
  Из любого другого места придется exit

Функции ввода/вывода

- int **printf** (const char * **format**, ...);
  int **fprintf** (FILE * **stream**, const char * **format**, ...);
  int **sprintf** (char * **str**, const char * **format**, ...);
  int **snprintf** (char * **str**, size_t **size**,
  const char * **format**, ...);
- int **scanf** (const char * **format**, ...);
  int **fscanf** (FILE * **stream**, const char * **format**, ...);
  int **sscanf** (const char * **str**, const char * **format**, ...);

Функции ввода/вывода - 2

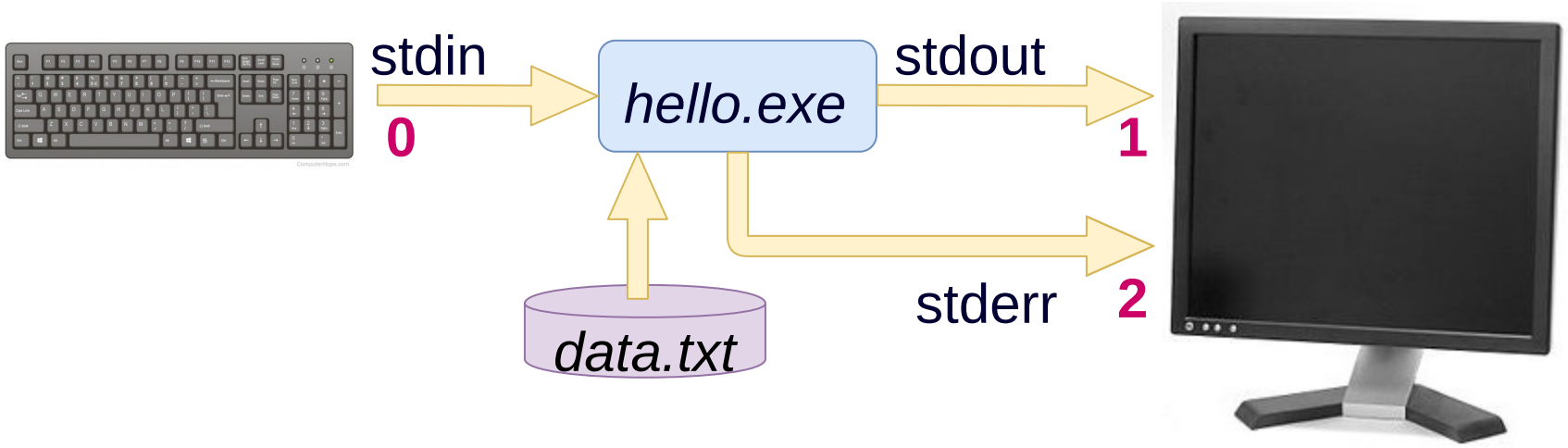
- ```
  int    fputc    (int c, FILE * fp);
  int    putc     (int c, FILE * fp);  // м.б. макросом
  int    putchar  (int c);             // putc(c, stdout)
  int    fputs    (const char * str, FILE * stream);
  int    puts     (const char * str);
  ```

- ```
  int    fgetc    (FILE * fp);
  int    getc     (FILE * fp);  // м.б. макросом
  int    getchar  ( );          // getc(stdout)
  int    ungetc (int c, FILE * stream);      // :(
  char *   fgets    (char * str, size_t size,
                     FILE * stream);
  char *   gets     (char * str);        // don't use!
  ```

Буферизация

- int **fflush** (FILE * **stream**);
  int **setvbuf**(FILE * **stream**, char * **buf**,
  int **mode**, size_t **size**);

- **_IONBD** — unbuffered
  _IOLBD — line buffered (связан с терминалом)
  _IOFBD — fully buffered (вывод в файл)



Включение буфера

- #define BUFSIZE 1024

  ```
  char buf [BUFSIZE];
  setvbuf (stdout, buf, _IOFBF, BUFSIZE);
  printf("Hello, world!\n");
  ```

"Вилка" поиска где упало

- плохо
  ```
  printf ("Здесь еще печатает\n");
  foo(x);                                   // x = 5;
  printf ("Здесь уже не печатает\n");
  ```
- разумнее:
  ```
  fprintf (stderr, "%d : %s\n", __LINE__, __FUNCTION__);
  foo (x);
  fprintf (stderr, "%d : %s\n", __LINE__, __FUNCTION__);
  ```
- `#define prn(msg) fprintf (stderr, "%d : %s %s\n", __LINE__, __FUNCTION__, msg)`
- **gdb** — самый разумный выход

Позиционирование в файле

- Если записи в файле заранее известной постоянной длины, то удобнее по файлу перемещаться позиционно
- int **fseek** (FILE * stream, long offset, int whence);
  int **fgetpos** (FILE * stream, fpos_t * pos);
  int **fsetpos** (FILE * stream, const fpos_t * pos);
- Можно добавить своих указателей на позиции чтения/записи.
- append — указатель автоматически становится в конец файла

fseek

- Если записи в файле заранее известной постоянной длины, то удобнее по файлу перемещаться позиционно
- int **fseek** (FILE * **stream**, long **offset**, int **whence**);
- перемещаем им position indicator в потоке stream
- сдвигаем на offset байт
- whence — относительно
  - SEEK_SET — начала файла
  - SEEK_CUR — текущей позиции в файле
  - SEEK_END — конца файла

fseek (fout, 0, SEEK_END)
встали на конец файла

fread, fwrite

- Если записи в файле заранее известной постоянной длины struct Student , то удобнее читать и писать данные фиксированной длины
- size_t **fread** (void * **ptr**,
  size_t **size**, size_t **nmemb**,
  FILE * **stream**);
- size_t **fwrite** (const void * **ptr**,
  size_t **size**, size_t **nmemb**,
  FILE * **stream**);
- ptr — указатель на память откуда читаем/пишем nmemb элементов, каждый размером size
  Возвращает сколько байт прочитано/записано

- #define GROUP_SIZE 20
- struct Student { ... };
- struct Student arr [20];

  arr [0] = ivan;
  arr [1] = alex;
  arr [2] = natasha;

  size_t res = fwrite(arr, sizeof(struct Student), 3, fin);