# Отчет по лабораторной работе № 8

## по курсу "Операционные системы"

Студент группы М8О-208Б-23 Соловьева Надежда Сергеевна

Работа выполнена

Преподаватель: Егор Живалев

1. **Тема**: Диагностика ПО
2. **Цель работы**:
    - Приобретение практических навыков диагностики работы программного обеспечения
3. **Задание**: При выполнении лабораторных работ по курсу ОС необходимо продемонстрировать ключевые системные вызовы, которые в них используются и то, что их использование соответствует варианту ЛР.
   По итогам выполнения всех лабораторных работ отчет по данной ЛР должен содержать краткую сводку по исследованию написанных программ.
   Средства диагностики:
   Для ОС Windows:
   ● Windbg
   - http://windbg.info/doc/1-common-cmds.html
   ● Sysinternals Suite
   - Handle.exe
   - Procmon.exe
   - Procexp.exe
   Для ОС *nix:
   ● strace

**Описание используемых утилит:**
Strace — это утилита Linux, отслеживающая системные вызовы, которые представляют собой механизм трансляции, обеспечивающий интерфейс между процессом и операционной системой. Использование данной утилиты позволяет понять, что процесс пытается сделать в данное время. Strace может быть полезен при отладке программ.
Для удобства работы с протоколом утилиты можно использовать следующие ключи:
-o file – Перенаправить протокол утилиты в файл file
-e trace=filters – Указать выражения, по которым будут фильтроваться системные вызовы. Например -e trace=write,%process задаёт фильтрацию по системным вызовам write и по группе системных вызовов, связанных с межпроцессорным взаимодействием.
-f – Отслеживать системные вызовы в дочерних процессах
-y – Заменить в протоколе все файловые дескрипторы на имена соответствующих им файлов (где возможно).
-p file – Отслеживать только обращения к файлу file
-k – Отображать стек вызовов
Ltrace – это утилита Linux, отслеживающая системные вызовы, связанные с динамическими библиотеками. Использование данной утилиты очень похоже на использование strace.

Рассмотрим на примере Лабораторной работы №1.

4. **Листинг диагностики**:

```
hopeee@hopeeenad:~/repository/OC/build/lab1$ strace -f -e
trace="%process,read,write,dup2,pipe" -o log_lab1.txt ./parent
Введите название файла: result.txt
Введите команду (или пустую строку для завершения):
133
-144 12 -32
24 89
89 g5
Ошибка: обнаружен посторонний символ.
```

```
hopeee@hopeeenad:~/repository/OC/build/lab2$ strace -f -e
trace="%process,read,write" -o log_lab2.txt ./lab2_exe 8 2 2 5
Matrix size: 8
Convolution window size: 2
Convolution filter count: 2
MAX threads: 5

Original matrix:
   1    2    3    4    5    6    7    8
   9   10   11   12   13   14   15   16
  17   18   19   20   21   22   23   24
  25   26   27   28   29   30   31   32
  33   34   35   36   37   38   39   40
  41   42   43   44   45   46   47   48
  49   50   51   52   53   54   55   56
  57   58   59   60   61   62   63   64

Matrix after convolution:
 160  176  192  208  224  240
 288  304  320  336  352  368
 416  432  448  464  480  496
 544  560  576  592  608  624
 672  688  704  720  736  752
 800  816  832  848  864  880

Used threads: 5
Time spent: 0.00723 seconds
```

```
hopeee@hopeeenad:~/repository/OC/build/lab3$ strace -f -e
trace="%process,read,write,dup2,mmap" -o log_lab3.txt ./lab3_exe
Enter the output file name: result.txt
133
-144 12 -32
24 89
89 g5
Error: invalid input
End of work. You can try again)
```

```
hopeee@hopeeenad:~/repository/OC/build/lab4$ ltrace -o log_lab4.txt ./dynamic
Режимы:
0 - Смена реализаций,
1 - Расчет числа 'e',
2 - Перевод числа X из 10СС в 2СС или 3СС,
3 - Выход.
Введите команду (режим аргумент): 1
Ошибка: аргумент должен быть положительным целым числом.
Введите команду (режим аргумент): 1 3
Результат E(3) = 2.370370
Введите команду (режим аргумент): 2 8
Результат перевода: 1000
Введите команду (режим аргумент): 0
Библиотека переключена.
Введите команду (режим аргумент): 2 8
Результат перевода: 22
Введите команду (режим аргумент): 3
Завершение работы
```

```
hopeee@hopeeenad:~/repository/OC/build/lab5$ strace -o log_lab5.txt
./controller
> create 1 2
> Error: Parent not found
> create 1
> Ok: 6581
> exit
Terminating worker with PID: 6581
```

5. **Результаты диагностики**: lab1 (logs/log_lab1.txt), lab2 (logs/log_lab2.txt), lab3 (logs/log_lab3.txt), lab4 (logs/log_lab4.txt), lab5 (logs/log_lab5.txt)
6. **Анализ log_lab1.txt (основные моменты)**:

```
1777  execve("./parent", ["./parent"], 0x7ffec4052b98 /* 26 vars */) = 0
```

Процесс с PID 1777 запускает исполняемый файл parent. execve заменяет текущий процесс на новый, указанный в аргументах. Возвращаемое значение 0 указывает на успешное выполнение.

```
1777  read(3,
"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) =
832
```

Процесс читает 832 байта из файлового дескриптора 3. Содержимое, начинающееся с \177ELF, указывает на то, что читается ELF-файл, вероятно, исполняемый.

```
1777  clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f2452208a10) = 1778
```

Процесс создает новый дочерний процесс с PID 1778 с помощью системного вызова clone. Установка флагов предполагает, что родительский процесс будет уведомлен о завершении дочернего через сигнал SIGCHLD.

```
1778  dup2(3, 0) = 0
```

Дочерний процесс перенаправляет файловый дескриптор 3 на стандартный ввод (файловый дескриптор 0) с помощью dup2.

```
1778  execve("../lab1/child", ["child"], 0x7ffc8cab4bd8 /* 26 vars */) = 0
```

Дочерний процесс запускает исполняемый файл child. Это замещает текущий процесс child.

```
1777  write(1, "\320\222\320\262..."..., 44) = 44
```

Процесс пишет 44 байта в стандартный вывод (файловый дескриптор 1). В данном случае выводятся символы в кодировке UTF-8.

```
1777  read(0,  <unfinished ...> и 1778  read(0,  <unfinished ...>
```

Оба процесса пытаются читать из стандартного ввода. read в состоянии <unfinished ...> указывает, что операция чтения еще не завершена и процесс ожидает данных.

```
1777  <... read resumed>"result.txt\n", 1024) = 11
```

Чтение из стандартного ввода возобновлено, и было прочитано 11 байт, включая строку "result.txt\n".

```
1777  write(4, "result.txt\n", 11) = 11
```

Процесс пишет строку "result.txt\n" в файловый дескриптор 4.

```
1778  write(2, "\320\236\321\210..."..., 70) = 70
```

Дочерний процесс пишет 70 байт в стандартный поток ошибок (файловый дескриптор 2), указывая на то, что произошла ошибка.

```
1778  exit_group(1) = ?
```

Дочерний процесс завершает работу с кодом 1, что указывает на ненормальное завершение.

```
1777  --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=1778,
si_uid=1000, si_status=1, si_utime=0, si_stime=2} ---
```

Основной процесс получает сигнал SIGCHLD, уведомляющий о завершении дочернего процесса с PID 1778. Код завершения дочернего процесса — 1.

```
1777  exit_group(0) = ?
```

Основной процесс завершает свою работу с кодом 0, что указывает на успешное завершение.

7. **Вывод:** В процессе выполнения лабораторной работы научились использовать такие инструменты диагностики как strace и ltrace. Утилиты strace и ltrace – простые и надежные инструменты для отслеживания системных вызовов. Они полезны при отладке и тестировании программ. Несмотря на то, что на первый взгляд протоколы этих утилит кажутся чересчур объёмными и непонятными, при помощи различных ключей можно отфильтровать их «на свой вкус». Данные

утилиты пригодились мне для изучения принципов работы системных вызовов и поиска ошибок в моих лабораторных работах. Личная оценка: 9/10.