Лабораторная работа №1

Тема: Последовательный сервер. Использование каналов
Язык программирования: C
ОС: Linux

Разработать приложение клиент-сервер для передачи текстовых файлов.

Клиент:

- считывает полное имя файла из стандартного потока ввода;
- передает имя файла серверу;
- считывает данные файла (или сообщение об ошибке) из канала;
- записывает полученные данные в стандартный поток вывода.

Сервер:

- считывает полное имя файла из канала;
- производит попытку открытия файла для чтения. Если попытка оказывается успешной, считывает файл и записывает его в канал. В противном случае сервер возвращает клиенту сообщение об ошибке.

Сервер представляет собой длительно функционирующий процесс (демон), не являющийся родственным клиенту. Сервер создает канал с именем /tmp/serv.fifo и открывает его на чтение. Запускаемые впоследствии клиенты открывают этот канал на запись и отправляют демону команды и необходимые данные.

Каждый клиент при запуске создает свой собственный канал, полное имя которого определяется его идентификатором процесса (PID). Клиент отправляет свой запрос в канал сервера, причем в запросе содержится идентификатор процесса клиента и имя файла, отправку которого клиент запрашивает у сервера.

Проверить использование сервера из интерпретатора команд:

- узнать PID текущей копии интерпретатора команд;
- создать именованный канал для клиента

```
$ mkfifo /tmp/fifo.$Pid
```

- отослать запрос на сервер

```
$ echo "$Pid /tmp/123.txt" > /tmp/serv.fifo
```

- считать ответ сервера

```
$ cat < /tmp/fifo.$Pid
```

Исследовать возможность реализации атаки типа «отказ в обслуживании».

Лабораторная работа №2

Тема: Параллельный сервер. Потоки и их синхронизация
Язык программирования: C
ОС: Linux

Разработать многопоточное приложение, использующее очереди сообщений и средства синхронизации потоков.

Формат вызова:

```
$ gen_prog [options]
```

необязательные опции программы:

- -q 3 | --queue=3 — размер очереди сообщений;
- -t 2 | --threads=2 — максимальное число потоков-обработчиков.

Описание программы:

- поток A (поставщик) считывает текстовый файл с заданиями, формирует запросы и помещает их на обработку в очередь сообщений;
- число элементов очереди сообщений задается в виде необязательного параметра при запуске программы (по умолчанию 5);
- поток B (потребитель) читает из очереди сообщений и запускает отдельные потоки-обработчики для выполнения задания;
- задания потокам-обработчикам передаются через одну или несколько структур данных в памяти;
- максимальное число одновременно работающих потоков-обработчиков задается в виде необязательного параметра при запуске программы (по умолчанию 3);
- поток-обработчик выводит информацию в поток stderr посимвольно.

Формат текстового файла с заданиями. Файл состоит из строк следующей структуры (tab-separated CSV file):

```
число_групп число_символов_в_группе символ задержка
```

где

- разделителем полей служит символ табуляции;
- символ — символ для вывода;
- число_символов_в_группе — число подряд выводимых символов;
- задержка — время ожидания в миллисекундах перед выводом следующей группы символов;
- число_групп — число групп выводимых символов.

Лабораторная работа №3

Тема: Разработка клиент-серверного приложения (TCP)
Язык программирования: C
ОС: Linux/Windows

Разработать приложение клиент-сервер для передачи файлов. Сервер является приемником файла, клиент — источником. По умолчанию клиент получает файл из стандартного потока ввода stdin, сервер выдает файл в стандартный поток вывода stdout. Сообщения о возможных ошибках выводятся в поток ошибок stderr.

Приложения должны быть кросс-платформенными, т.е. одни и те же исходные файлы могут быть собраны как для Linux, так и для Windows.

**Клиент**

Формат вызова:

```
$ tcpsnd [options]
```

необязательные опции программы:

- -f name | --file=name — получить данные для передачи из файла с именем name;
- -a 127.0.0.1 | --address=127.0.0.1 — IP адрес сервера;
- -p 7089 | --port=7089 — номер порта на сервере.

**Сервер**

Формат вызова:

```
$ tcprcv [options]
```

необязательные опции программы:

- -f name | --file=name — сохранить полученные данные в файл с именем name;
- -p 7089 | --port=7089 — номер порта на сервере.

**Тестовые примеры**

На Windows-машине:

```
$ tcprcv –p 8115
```

На Linux-машине:

```
$ ls –l /etc | tcpsnd –a 192.168.100.10 –p 8115
```

На Linux-машине:

```
$ tcprcv –p 8115 ––file=/tmp/file.zip
```

На Windows-машине:

```
$ tcpsnd –a 192.168.100.121 –p 8115 < d:/file.zip
```

Лабораторная работа №4

Тема: Разработка клиент-серверного приложения (UDP)
Язык программирования: C
ОС: Linux

Разработать приложение для реализации эхо-сервера и клиента, используя протокол UDP.

Клиент считывает строку из стандартного потока ввода stdin и отсылает ее на сервер. Сервер возвращает полученную строку назад клиенту. Клиент выдает ответ сервера в стандартный поток вывода stdout и завершает свою работу. Сообщения о возможных ошибках в сервере и клиенте выводятся в поток ошибок stderr.

Необходимо реализовать последовательный сервер, который после запуска продолжает свое функционирование до тех пор, пока не будет получен сигнал SIGUSR1.

**Клиент**

Формат вызова:

```
$ udpecho [options]
```

необязательные опции программы:

- -a 127.0.0.1 | --address=127.0.0.1 — IP адрес сервера;
- -p 1234 | --port=1234 — номер порта на сервере.

**Сервер**

Формат вызова:

```
$ echoserv [options]
```

необязательные опции программы:

- -p 2345 | --port=2345 — номер порта на сервере.