Задание

Скомпилируйте *tinyhttpd*, перейдя в директорию с его исходниками и выполнив команду `make`. Потестируйте его, убедитесь, что работает (как делали на лекции). Запустите его под strace (см. материалы лекции 2 и `man strace`). Опишите все системные вызовы, которые он использует (см. задание к лекции 02). Перечислите системные вызовы, которые выполняются при обработке отдельного запроса.

Решение

Собираем с помощью make-файла сервер, далее вводим в терминал команду `strace -c httpd&`, находим PID процесса strace с помощью `ps`, посылаем ему SIGINT и получаем такой список системных вызовов (вызовы, которые использует *httpd*)

- execve
- mmap
- socket
- mprotect
- open
- write
- munmap
- bind — привязывает к сокету локальный адрес
- listen — слушает соединения на сокете
- read
- fstat
- access

- close
- rt_sigaction — используется для изменения выполняемого процессом действия при получении определённого сигнала
- rt_sigprocmask — используется для выборки и/или изменения маски сигналов вызывающей нити. Маска сигналов представляет собой набор сигналов, чья доставка в настоящее время заблокирована вызывающим процессом
- getrlimit — получает ограничения использования ресурсов
- arch_prtcl
- set_tid_address — устанавливает указатель на идентификатор нити
- set_robust_list — устанавливает лист robust futexes
- brk

Теперь надо выяснить, какие вызовы используются при обработке запроса.

Запоминаем количество вызовов для каждого сисвызова (зачем — будет ясно совсем скоро). Создаём с помощью screen отдельное окно. Возвращаемся из screen в исходную консоль, снова запускаем команду `strace -c httpd&`. Переходим в созданный ранее screen. Здесь вводим команду `telnet localhost 80`, дожидаемся соединения, вводим запрос `GET /index.html`, получаем OK-ответ.

Отсоединяем экран, попадаем к серверу, снова находим PID процесса strace, посылаем ему SIGINT. Снова видим список системных вызовов.

Есть некоторые совсем новые:

- accept — принимает соединение на сокете

- resvfrom — получает сообщение из сокета
- stat — получает информацию о файле
- sendto — отправляет сообщение в сокет

Для остальных же сверяем количество вызовов сейчас с количеством вызовов, которое требовалось для запуска http сервера (то, что мы запоминали выше). Таким образом находим ещё несколько сисвызовов, которые нужны были для обработки запроса:

- mmap
- open
- munmap
- read
- fstat
- close
- brk

Вот так.

Я ещё пробовал запускать тот клиент, который *simpleclient.c* из той же папки, что и *httpd.c*. Но я, видимо, не до конца разобрался с кодом программ, так как мне пришлось послать SIGINT strace, чтобы клиент завершился (с сообщением, что "получил символ A от сервера"). И в этом случае, для обработки запроса от клиента, потребовалось только два дополнительных системных вызова (которые выполнялись по одному разу):

- accept
- recvfrom

Да, и в клиенте пришлось поменять порт в строчке

```
address.sin_port = htons(9734);
```

на 80 вместо 9734.