Лабораторная работа № 3
Использование модели Select

3.1 Цель работы

Целью третьей лабораторной работы является ознакомление студентов с возможностью организации параллельного обслуживания клиентов с использованием неблокирующих сокетов и модели select.

3.2 Задание на лабораторную работу

В лабораторной работе необходимо разработать сервер, использующий неблокирующие сокеты и модель select. Сервер должен принимать/отправлять файлы по запросу клиентов. Количество клиентов выбрать равное 3.

3.3 Методические указания

3.3.1 Неблокирующие сокеты

Сначала разберёмся, что такое ***неблокирующие*** сокеты. Сокеты, которые мы до сих пор использовали, являлись ***блокирующими*** (blocking). Это означает, что на время выполнения операции с таким сокетом ваша программа блокируется. Например, если вы вызвали recv, а данных на вашем конце соединения нет, то в ожидании их прихода ваша программа "засыпает". Аналогичная ситуация может наблюдаться при вызове других функций. Это поведение можно изменить, переведя сокет в неблокирующий режим. Для этого существует соответствующая функция:

Таблица 3.1

Описание функции в *nix	Описание функции в Windows
#include <unistd.h> #include <fcntl.h>	#include <winsock2.h>
fcntl(sockfd ,F_SETFL, O_NONBLOCK);	int ioctlsocket(SOCKET s, long cmd, // указать FIONBIO u_long *argp);

Вызов любой функции с таким сокетом будет возвращать управление немедленно. Причём если затребованная операция не была выполнена до конца, функция вернёт -1(SOCKET_ERROR в Windows) и запишет в ***errno*** (WSAGetLastError) значение ***EWOULDBLOCK*** (WSAEWOULDBLOCK для Windows). Чтобы дождаться завершения операции, мы можем опрашивать

все наши сокеты в цикле, пока какая-то функция не вернёт значение, отличное от ***EWOULDBLOCK*** (WSAEWOULDBLOCK для Windows). Как только это произойдёт, мы можем запустить на выполнение следующую операцию с этим сокетом и вернуться к нашему опрашивающему циклу. Такая тактика работоспособна, но очень неэффективна, поскольку процессорное время тратится впустую на многократные (и безрезультатные) опросы.

3.3.2 Select

Модель ***select*** позволяет отслеживать состояние нескольких дескрипторов сокетов одновременно.

Таблица 3.2

Описание функции в *nix	Описание функции в Windows
#include <sys/time.h> #include <sys/types.h> #include <unistd.h>	#include <winsock2.h>
int select(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);	int select(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);

Функция ***select*** работает с тремя множествами дескрипторов, каждое из которых имеет тип ***fd_set***. В множество ***readfds*** записываются дескрипторы сокетов, из которых нам требуется читать данные (слушающие сокеты добавляются в это же множество). Множество ***writefds*** должно содержать дескрипторы сокетов, в которые мы собираемся писать, а ***exceptfds*** - дескрипторы сокетов, которые нужно контролировать на возникновение ошибки. Если какое-то множество вас не интересуют, вы можете передать вместо указателя на него NULL. Что касается других параметров, в ***n*** нужно записать максимальное значение дескриптора по всем множествам плюс единица, а в ***timeout*** - величину таймаута. Структура ***timeval*** имеет следующий формат.

Таблица 3.3

Описание структуры в *nix	Описание структуры в Windows
#include <sys/time.h> #include <sys/types.h> #include <unistd.h>	#include <winsock2.h>

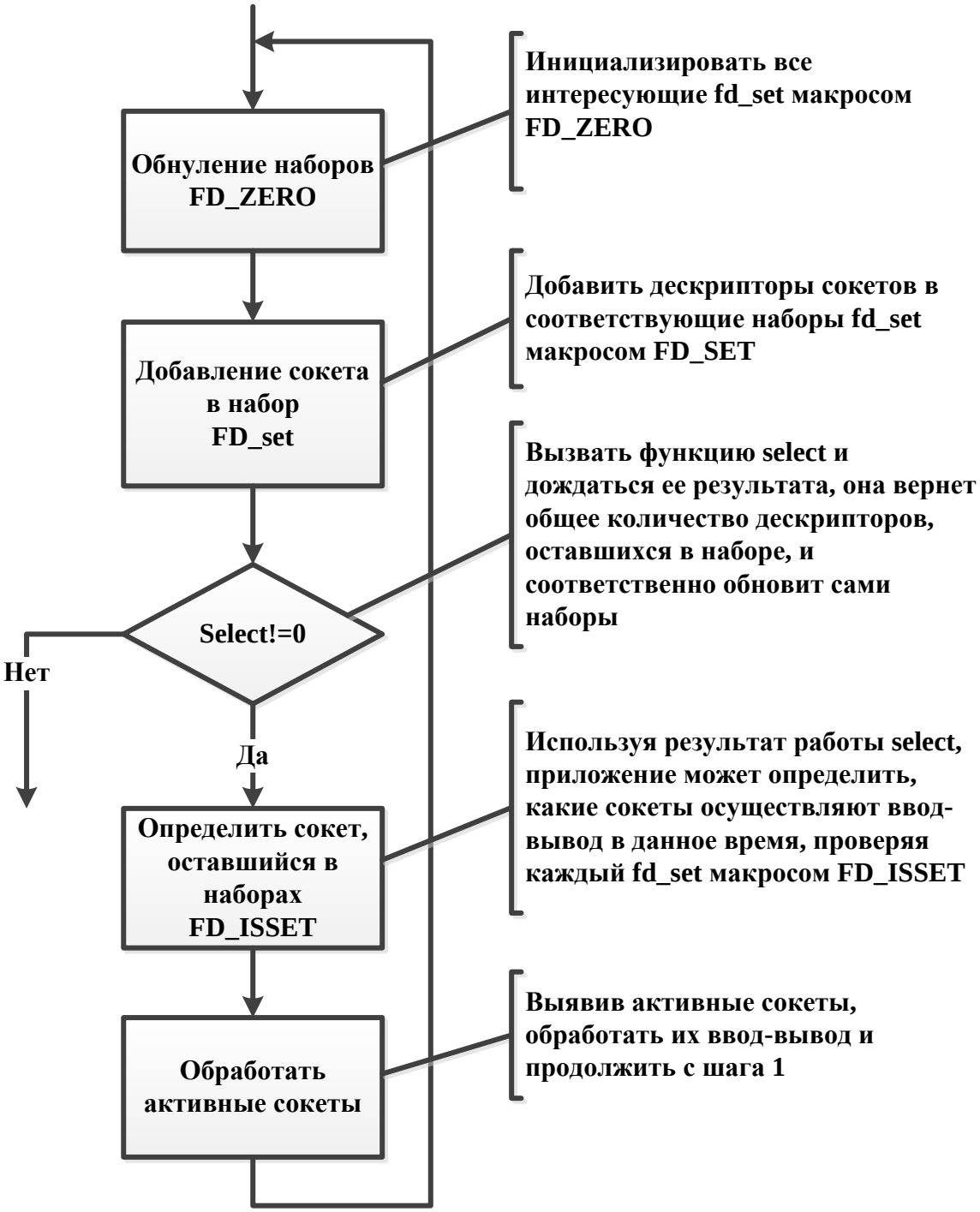
struct timeval {
int tv_sec; // секунды
int tv_usec; // микросекунды
};
	struct timeval {
	long tv_sec; // секунды
	long tv_usec; // микросекунды
	};

Поле "микросекунды" смотрится впечатляюще. Но на практике вам не добиться такой точности измерения времени при использовании ***select***. Реальная точность окажется в районе 100 миллисекунд.

Теперь займёмся множествами дескрипторов. Для работы с ними предусмотрены макросы ***FD_XXX***, показанные выше; их использование полностью скрывает от нас детали внутреннего устройства ***fd_set***. Рассмотрим их назначение.

- **FD_ZERO**(fd_set *set) - очищает множество set
- **FD_SET**(int fd, fd_set *set) - добавляет дескриптор fd в множество set
- **FD_CLR**(int fd, fd_set *set) - удаляет дескриптор fd из множества set
- **FD_ISSET**(int fd, fd_set *set) - проверяет, содержится ли дескриптор fd в множестве set

Если хотя бы один сокет готов к выполнению заданной операции, ***select*** возвращает ненулевое значение, а все дескрипторы, которые привели к "срабатыванию" функции, записываются в соответствующие множества. Это позволяет нам проанализировать содержащиеся в множествах дескрипторы и выполнить над ними необходимые действия. Если сработал таймаут, ***select*** возвращает ноль, а в случае ошибки -1(SOCKET_ERROR для Windows). Расширенный код записывается в ***errno*** (WSAGetLastError для Windows).



По завершению работы функция select удаляет из каждой структуры ***fd_set*** описатели сокетов, не участвующих в операциях ввода-вывода. Этим объясняется необходимость использовать макросы ***FD_ISSET*** на шаге 4, чтобы определить, является ли конкретный сокет частью набора.

3.4 Контрольные вопросы и задания

1. Описать различия работы с сокетами в блокирующем и неблокирующем режиме.
2. Описать способ перевода сокета в неблокирующий режим
3. Объяснить назначение и применение функции select
4. Объяснить назначение и применение макросов FD_XXX
5. Объяснить алгоритм модели select
6. Измерить скорость передачи при одновременной работе сервера с одним, двумя и тремя клиентам