Семестр 4 (2019), занятие 4 (часть 2)
Неблокирующий ввод-вывод на примере именованных каналов

Именованные каналы

Именованные каналы представляют собой механизм локального межпроцессного взаимодействия, обеспечивающий высокопроизводительную и детерминированную передачу данных. Они представляют собой файлы специального типа и могут быть созданы с помощью служебной программы `mkfifo`.

Схема эксперимента

В рамках эксперимента запускается распределенное приложение. Это приложение состоит из трех компонентов (двух клиентов и одного сервера). Каждый компонент запускается в отдельном терминале. Для межпроцессного взаимодействия используются два именованных канала `a.dat` и `b.dat`. Первый (второй) клиент считывает данные из своего стандартного потока ввода и записывает их в именованный канал `a.dat` (`b.dat`). Сервер считывает данные из именованных каналов `a.dat` и `b.dat`, и печатает полученные данные в свой стандартный поток вывода.

Клиент 1 (терминал 1).

```
$ mkfifo a.dat
$ cat /dev/stdin > a.dat
Hello,<Enter>
!<Enter>
...
```

Клиент 2 (терминал 2).

```
$ mkfifo b.dat
$ cat /dev/stdin > b.dat
world<Enter>
...
```

Сервер (терминал 3).

```
$ ./server
Hello,world!...
```

Блокирующий ввод-вывод

Структура сервера для демонстрации блокирующего ввода-вывода.

```
int main() {
    int fd[] = { open("a.dat", O_RDONLY),
                 open("b.dat", O_RDONLY) };

    for(;;)
        for(int i = 0; i < 2; i ++)
            process(fd[i]);

    return 0;
}
```

Неблокирующий ввод-вывод

Структура сервера для демонстрации неблокирующего ввода-вывода.

```
int main() {
    int fd[] = { open("a.dat", O_RDONLY),
                 open("b.dat", O_RDONLY) };

    fcntl(fd[0], F_SETFL, O_NONBLOCK);
    fcntl(fd[1], F_SETFL, O_NONBLOCK);

    for(;;)
        for(int i = 0; i < 2; i ++) {
            process(fd[i]);
            sleep(1);
        }

    return 0;
}
```

Мультиплексирование ввода-вывода

Структура сервера для демонстрации использования мультиплексирования ввода-вывода.

```
int main() {
    pollfd fd[] = { {open("a.dat", O_RDONLY), POLLIN, 0},
                    {open("b.dat", O_RDONLY), POLLIN, 0} };

    fcntl(fd[0].fd, F_SETFL, O_NONBLOCK);
    fcntl(fd[1].fd, F_SETFL, O_NONBLOCK);

    for(;;) {
        poll(fd, 2, 2 * 1000);

        for(int i = 0; i < 2; i ++)
            if(fd[i].revents & POLLIN)
                process(fd[i].fd);
    }

    return 0;
}
```

Системный вызов `poll`

Системный вызов `poll` ожидает пока хотя бы один из множества файловых дескрипторов не будет готов для выполнения операции ввода-вывода.

Множество файловых дескрипторов задается в виде массива структур типа `struct pollfd`. Входной параметр `fds` представляет указатель на начало этого массива, а входной параметр `nfds` задает количество элементов этого массива. Входной параметр `timeout` задает время ожидания в миллисекундах готовности файлового дескриптора. Отрицательное значение этого параметра соответствует «бесконечному» времени ожидания.

При успешном завершении системного вызова возвращается положительное число равное количеству файловых дескрипторов готовых для выполнения операции ввода-вывода. Если было возвращено нулевое значение соответствует ситуации, когда время ожидания истекло и ни один файловый дескриптор не перешел в состояние готовности. В случае ошибки возвращается значение `-1`, а в глобальную переменную `errno` помещается код ошибки.

```
#include <poll.h>

int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

Структура `struct pollfd` содержит три поля. Поля `fd` и `events` играют роль входных параметров. Через поле `fd` задается значение файлового дескриптора. Поле `events` представляет собой битовую маску событий, предназначенных для наблюдения. В заголовочном файле `poll.h` определены константы для этих событий. Например, событие `POLLIN` означает наличие данных для чтения, а событие `POLLOUT` означает возможность записи. Значение поля `events` можно задать, применяя операцию побитого «или» («|») к этим константам. Поле `revents` является выходным параметром и представляет собой битовую маску произошедших событий.

```
struct pollfd {
    int   fd;
    short events;
    short revents;
};
```

Мультиплексирование ввода-вывода (полный текст)

```
// server3.cpp
#include <unistd.h>
#include <fcntl.h>
#include <poll.h>

#include <cstdio>

void process(int fd);

int main() {
    pollfd fd[] = { {open("a.dat", O_RDONLY), POLLIN, 0},
                    {open("b.dat", O_RDONLY), POLLIN, 0} };

    if(fd[0].fd == -1 || fd[1].fd == -1) {
        fprintf(stderr, "Can't open file\n");
        return -1;
    }

    if(fcntl(fd[0].fd, F_SETFL, O_NONBLOCK) == -1 ||
       fcntl(fd[1].fd, F_SETFL, O_NONBLOCK) == -1 ) {
        fprintf(stderr, "Can't setup NONBLOCK flag\n");
        return -1;
    }

    for(;;) {
        int r = poll(fd, 2, 2 * 1000);

        if(!r)
            puts("Nothing ...");
        else
            if(r == -1) {
                fprintf(stderr, "Can't perform poll operation\n");
                break;
            }
            else
                for(int i = 0; i < 2; i ++) {
                    if(fd[i].revents & POLLIN)
                        process(fd[i].fd);
                }
    }

    close(fd[0].fd);
    close(fd[1].fd);

    return 0;
}


// process.cpp
#include <unistd.h>
#include <cstdio>

#include <vector>
void process(int fd);

void process(int fd) {
    std::vector<char> bs(80, 0);
    ssize_t s = read(fd, &bs[0], bs.size() - 1);
    if(s > 0)
        printf("%d (%d) %s", fd, (int)s, &bs[0]);
    else
        printf("%d (%d)\n", fd, (int)s);
}
```