Отчёт по лабораторной работе №14

Именованные каналы

Аскеров Александр Эдуардович

Содержание

Список иллюстраций

1 Цель работы

Приобретение практических навыков работы с именованными каналами.

2 Теоретическое введение

Одним из видов взаимодействия между процессами в операционных системах является обмен сообщениями. Под сообщением понимается последовательность байтов, передаваемая от одного процесса другому.

В операционных системах типа UNIX есть 3 вида межпроцессорных взаимодействий: общеюниксные (именованные каналы, сигналы), System V Interface Definition (SVID — разделяемая память, очередь сообщений, семафоры) и BSD (сокеты).

Для передачи данных между неродственными процессами можно использовать механизм именованных каналов (named pipes). Данные передаются по принципу FIFO (First In First Out) (первым записан — первым прочитан), поэтому они называются также FIFO pipes или просто FIFO. Именованные каналы отличаются от неименованных наличием идентификатора канала, который представлен как специальный файл (соответственно имя именованного канала — это имя файла). Поскольку файл находится на локальной файловой системе, данное IPC используется внутри одной системы.

3 Выполнение лабораторной работы

Изучим приведённые в тексте программы server.c и client.c. Взяв данные примеры за образец, напишем аналогичные программы, внеся следующие изменения:

1. Работает не 1 клиент, а несколько (например, два).

Для этого создаём два файла: client.c и client2.c.

```
[aeaskerov@fedora lab14]$ touch client.c client2.c
```

Рис. 3.1: Файлы client.c и client2.c

2. Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд). Используйте функцию sleep() для приостановки работы клиента.

```
13 int
14 main()
15 {
16   int msg, len, i; /* дескриптор для записи в FIFO */
17   long int t;
18
19   for(i=0;i<20;i++)
20   {
21     /* используем фукцию sleep для приостановки работы клиента */
22     sleep(5);
23     t=time(NULL);
24
25     /* баннер */
26     printf("FIFO Client...\n");
```

Рис. 3.2: Функция sleep() для приостановки работы клиента в файлах client.c и client2.c

3. Сервер работает не бесконечно, а прекращает работу через некоторое время (например, 30 сек). Используйте функцию clock() для определения времени работы сервера. Что будет в случае, если сервер завершит работу, не закрыв канал?

```
39   /* используем функцию clock для определения времени работы сервера */
40   clock_t now=time(NULL), start=time(NULL);
41   while(now-start<30)
42   {
```

Рис. 3.3: Цикл while, ограничивающий время работы сервера, и функция clock

Если сервер завершит работу, не закрыв канал, то клиент продолжит ждать сигнал от сервера, но он никогда к нему не поступит.

3.1 Результат работы

В течение 30 секунд всё работает, после чего работа сервера прекращается.

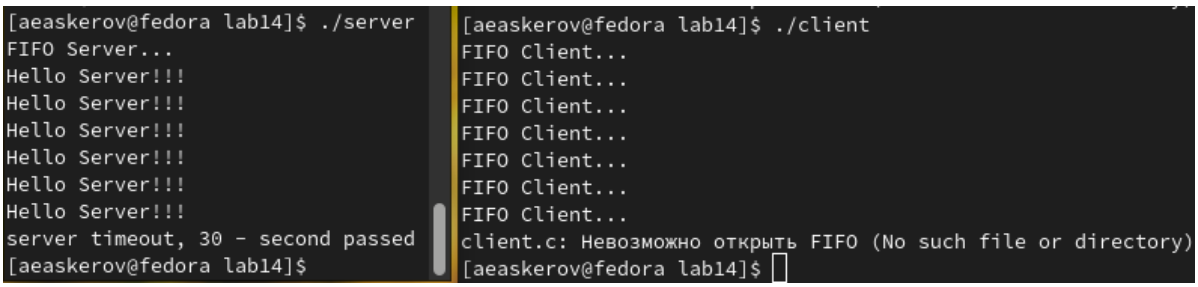

Рис. 3.4: Результат работы

4 Контрольные вопросы

1. В чем ключевое отличие именованных каналов от неименованных?

Именованные каналы отличаются от неименованных наличием идентификатора канала, который представлен как специальный файл (соответственно имя именованного канала – это имя файла). Поскольку файл находится на локальной файловой системе, данное IPC используется внутри одной системы.

2. Возможно ли создание неименованного канала из командной строки?

Чтобы создать неименованный канал из командной строки нужно использовать символ |, служащий для объединения двух и более процессов: процесс 1 процесс 2 процесс 3

3. Возможно ли создание именованного канала из командной строки?

Чтобы создать именованный канал из командной строки нужно использовать либо команду «mknod», либо команду «mkfifo».

4. Опишите функцию языка C, создающую неименованный канал.

Неименованный канал является средством взаимодействия между связанными процессами родительским и дочерним. Родительский процесс создает канал при помощи системного вызова: «int pipe(int fd[2]);». Массив из двух целых чисел является выходным параметром этого системного вызова. Если вызов выполнился нормально, то этот массив содержит два файловых дескриптора. fd[0] является

дескриптором для чтения из канала, fd[1] дескриптором для записи в канал. Когда процесс порождает другой процесс, дескрипторы родительского процесса наследуются дочерним процессом, и, таким образом, прокладывается трубопровод между двумя процессами. Естественно, что один из процессов использует канал только для чтения, а другой только для записи. Поэтому, если, например, через канал должны передаваться данные из родительского процесса в дочерний, родительский процесс сразу после запуска дочернего процесса закрывает дескриптор канала для чтения, а дочерний процесс закрывает дескриптор для записи. Если нужен двунаправленный обмен данными между процессами, то родительский процесс создает два канала, один из которых используется для передачи данных в одну сторону, а другой в другую.

5. Опишите функцию языка C, создающую именованный канал.

Файлы именованных каналов создаются функцией mkfifo() или функцией mknod: 1. «int mkfifo(const char pathname, mode tmode);», где первый параметр путь, где будет располагаться FIFO (имя файла, идентифицирующего канал), второй параметр определяет режим работы с FIFO (маска прав доступа к файлу), 2. «mknod (namefile, IFIFO | 0666, 0)», где namefile имя канала, 0666 к каналу разрешен доступ на запись и на чтение любому запросившему процессу), 3. «int mknod(const char pathname, mode t mode, dev t dev);».Функция mkfifo() создает канал и файл соответствующего типа. Если указанный файл канала уже существует, mkfifo() возвращает 1. После создания файла канала процессы, участвующие в обмене данными, должны открыть этот файл либо для записи, любо для чтения.

6. Что будет в случае прочтения из fifo меньшего числа байтов, чем находится в канале? Большего числа байтов?

При чтении меньшего числа байтов, чем находится в канале или FIFO, возвращается требуемое число байтов, остаток сохраняется для последующих чтений.

При чтении большего числа байтов, чем находится в канале или FIFO, возвращается доступное число байтов. Процесс, читающий из канала, должен соответствующим образом обработать ситуацию, когда прочитано меньше, чем заказано.

7. Аналогично, что будет в случае записи в fifo меньшего числа байтов, чем позволяет буфер? Большего числа байтов?

Запись числа байтов, меньшего емкости канала или FIFO, гарантированно атомарно. Это означает, что в случае, когда несколько процессов одновременно записывают в канал, порции данных от этих процессов не перемешиваются. При записи большего числа байтов, чем это позволяет канал или FIFO, вызов write(2) блокируется до освобождения требуемого места. При этом атомарность операции не гарантируется. Если процесс пытается записать данные в канал, не открытый ни одним процессом на чтение, процессу генерируется сигнал SIGPIPE, а вызов write(2) возвращает 0 с установкой ошибки (errno=ERRPIPE) (если процесс не установил обработки сигнала SIGPIPE, производится обработка по умолчанию процесс завершается).

8. Могут ли два и более процессов читать или записывать в канал?

Количество процессов, которые могут параллельно присоединяться к любому концу канала, не ограничено. Однако если два или более процесса записывают в канал данные одновременно, каждый процесс за один раз может записать максимум PIPE BUF байтов данных. Предположим, процесс (назовем его A) пытается записать X байтов данных в канал, в котором имеется место для Y байтов данных. Если X больше, чем Y, только первые Y байтов данных записываются в канал, и процесс блокируется. Запускается другой процесс (например. B); в это время в канале появляется свободное пространство (благодаря третьему процессу, считывающему данные из канала). Процесс B записывает данные в канал. Затем, когда выполнение процесса A возобновляется, он записывает оставшиеся X Y байтов данных в канал. В результате данные в канал записываются поочередно двумя

процессами. Аналогичным образом, если два (или более) процесса одновременно попытаются прочитать данные из канала, может случиться так, что каждый из них прочитает только часть необходимых данных.

9. Опишите функцию write (тип возвращаемого значения, аргументы и логику работы). Что означает 1 (единица) в вызове этой функции в программе server.c (строка 42)?

Функция write записывает байты count из буфера buffer в файл, связанный с handle. Операции write начинаются с текущей позиции указателя на файл (указатель ассоциирован с заданным файлом). Если файл открыт для добавления, операции выполняются в конец файла. После осуществления операций записи указатель на файл(если он есть) увеличивается на количество действительно записанных байтов. Функция write возвращает число действительно записанных байтов. Возвращаемое значение должно быть положительным, но меньше числа count (например, когда размер для записи count байтов выходит за пределы пространства на диске). Возвращаемое значение 1 указывает на ошибку; errno устанавливается в одно из следующих значений: EACCES файл открыт для чтения или закрыт для записи, EBADF неверный handle p файла, ENOSPC на устройстве нет свободного места. Единица в вызове функции write в программе server.c означает идентификатор (дескриптор потока) стандартного потока вывода.

10. Опишите функцию strerror.

Прототип функции strerror: «char * strerror(int errornum);». Функция strerror интерпретирует номер ошибки, передаваемый в функцию в качестве аргумента errornum, в понятное для человека текстовое сообщение (строку). Откуда берутся эти ошибки? Ошибки эти возникают при вызове функций стандартных Си библиотек. То есть хорошим тоном программирования будет использование этой функции в паре с другой, и если возникнет ошибка, то пользователь или программист поймет, как исправить ошибку, прочитав сообщение функции strerror.

Возвращенный указатель ссылается на статическую строку с ошибкой, которая не должна быть изменена программой. Дальнейшие вызовы функции strerror перезапишут содержание этой строки. Интерпретированные сообщения об ошибках могут различаться, это зависит от платформы и компилятора.

5 Выводы

Приобретены практические навыки работы с именованными каналами.