Отчет по лабораторной работе №14

Дисциплина: Операционные системы

Кашкин Иввн Евгеньевич

Содержание

Список иллюстраций

Список таблиц

Цель работы

- Приобретение практических навыков работы с именованными каналами

Задание

Изучите приведённые в тексте программы server.c и client.c. Взяв данные примеры за образец, напишите аналогичные программы, внеся следующие изменения: 1. Работает не 1 клиент, а несколько (например, два). 2. Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд). Используйте функцию sleep() для приостановки работы клиента. 3. Сервер работает не бесконечно, а прекращает работу через некоторое время (например, 30 сек). Используйте функцию clock() для определения времени работы сервера.

Теоретическое введение

Этапы разработки приложений - Процесс разработки программного обеспечения обычно разделяется на следующие этапы: - планирование, включающее сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения; - проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языка программирования; - непосредственная разработка приложения: - кодирование - по сути создание исходного текста программы (возможно в нескольких вариантах); - анализ разработанного кода; - сборка, компиляция и разработка исполняемого модуля; - тестирование и отладка, сохранение произведённых изменений; - документирование. Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: vi, vim, mceditor, emacs, geany и др. После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль.

Компиляция исходного текста и построение исполняемого файла

Стандартным средством для компиляции программ в ОС типа UNIX является GCC (GNU Compiler Collection). Это набор компиляторов для разного рода языков программирования (C, C++, Java, Фортран и др.). Работа с GCC производится при помощи одноимённой управляющей программы gcc, которая интерпретирует аргументы командной строки, определяет и осуществляет запуск нужного компилятора для входного файла. Файлы с расширением (суффиксом) .c воспринимаются gcc как программы на языке C, файлы с расширением .cc или .C как файлы на языке C++, а файлы с расширением .o считаются объектными.

Выполнение лабораторной работы

1. Зашёл в каталог лабораторной работы и создал сами программы (рис. [-@fig:001])

Рис. 0.1: Создание

2. Написал командный файл для файла common.h (рис. [-@fig:002])

```
#ifndef __COMMON_H__
#define __COMMON_H__

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <time.h>

#define FIFO_NAME "/tmp/fifo"
#define MAX_BUFF 80

#endif
```

Рис. 0.2: common.h

3. Написал командный файл server.c (рис. [-@fig:004])

```c
#include "common.h"

int main(){
  int readfd;
  int n;
  char buff[MAX_BUFF];

  printf("FIFO Server...\n");

  if (mknod(FIFO_NAME, S_IFIFO | 0666, 0)<0)
    {
      fprintf (stderr, "%s: Невозможно создать FIFO (%s)\n", __FILE__, strerror(errno));
      exit(-1);
    }
  if ((readfd = open(FIFO_NAME, O_RDONLY)) <0)
    {
      fprintf(stderr, "%s:  Невозможно открыть FIFO (%s)\n", __FILE__,  strerror(errno));
      exit(-2);
    }
  clock_t start = time(NULL);

  while(time(NULL)-star < 30)
    {
      while ((n = read(readfd,buff,MAX_BUFF))>0)
        {
          if (write(1, buff, n) != n)
            {
              fprintf (stderr, "%s: Ошибка вывода (%s)\n", __FILE__,  strerror(errno));
              exit(-3);
            }
        }
    }
```

Рис. 0.3: server.c

```c
] close (readfd);
  if (unlink(FIFP_NAME)<0)
    {
      fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n", __FILE__,  strerror(errno));
              exit(-3);
    }
  exit(0);
}
```

Рис. 0.4: server.c

- Написал командный файл client.c (рис. [-@fig:005])

```c
#include "common.h"

int main(){
  int writefd;
  int msglen;

  printf("FIFO Clirnt...\n");

  for (int i=0; i<4; i++){
    if ((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
      {
        Fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n", __FILE__,  strerror(errno));
              exit(-1);
      }

    long int ttime = time(NULL);
    char* text = ctime(&ttime);

    msglen = strlen(text);
    if (write(writefd, text, msglen) != msglen){
      fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n", __FILE__,  strerror(errno));
              exit(-2);
    }
    sleep(5);
  }
  close(writefd);
  exit(0);
}$
```

Рис. 0.5: client.c

4. Создал Makefile (рис. [-@fig:006])

```
all: server client

server: serer.c common.h
        gcc server.c -o server

client: client.c common.h
        gcc client.c -o client

clean:
        -rm server client *.o
```

Рис. 0.6: Makefile

- Создал с помошью Makefile нужные программы для работы (рис. [-@fig:007])

```
[ivanekashkin@iekashkin lab14]$ make all
gcc client.c -o client
```

Рис. 0.7: Makefile

5. Проверка выполнения скрипта (рис. [-@fig:008])

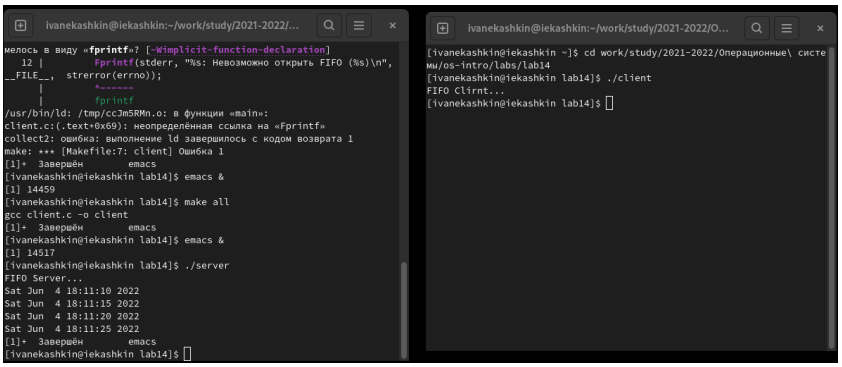

Рис. 0.8: Ввод программы

Выводы

- В ходе выполнения данной лабораторной работы я приобрёл практические навыки работы с именованными каналами.

Контрольные вопросы

1. 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. Чтобы создать неименованный канал из командной строки нужно использовать символ |, служащий для объединения двух и более процессов: процесс_1 | процесс_2 | процесс_3...
3. Чтобы создать именованный канал из командной строки нужно использовать либо команду «mknod », либо команду «mkfifo ».
4. 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. Файлы именованных каналов создаются функцией mkfifo() или функцией mknod:

- «int mkfifo(const char *pathname, mode_t mode);», где первый параметр путь, где будет располагаться FIFO (имя файла, идентифицирующего канал), второй параметр определяет режим работы с FIFO (маска прав доступа к файлу),
- «mknod (namefile, IFIFO | 0666, 0)», где namefile имя канала, 0666 к каналу разрешен доступ на запись и на чтение любому запросившему процессу),
- «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. 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. 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. 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. 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 функ-

ции write в программе server.c означает идентификатор (дескриптор потока) стандартного потока вывода.

10. 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.

Список литературы

::: {#Лабораторная работа No 13. Средства, применяемые при разработке программного обеспечения в ОС типа UNIX/Linux} :::