Московский авиационный институт (национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Операционные системы»

Студентка:	А. Довженко
Преподаватель:	Е. С. Миронов
Группа:	08-207
Вариант:	2
Дата:	
Оценка:	
Подпись:	

Москва, 2017

Лабораторная работа №2

1 Описание

Процесс – абстракция, описывающая выполняющуюся программу. В моем случае, каждый процесс представляет собой вычисление некоторого n-ого члена последовательности Фибоначчи. Передача информации между родительскими и дочерними процессами осуществляется с помощью каналов (pipe). Если нарисовать дерево из рекурсивных вызовов, то можно увидеть, что все процессы, кроме тех что в корне и листьях дерева, являются одновременно и родительскими, и дочерними. Это значит, что считывать информацию из потока они будут как родительские процессы, а записывать как дочерние. Благодаря этому можно использовать только 2 файловых дескриптора.
Для корректной работы программы, использующей больше одного процесса, необходимо решить три проблемы: передача информации между процессами, обеспечение совместной работы без создания взаимных помех, определение правильной последовательности работы процессов. Как сказано выше, информация передается по каналам. Совместная работа обеспечивается благодаря системным вызовам read, write и close, которые используют файловые дескрипторы в определенной последовательности для корректной передачи полученных чисел по каналу. Правильная последовательность работы процессов осуществляется при помощи waitpid. Этот системный вызов преостанавливает выполнение текущего процесса до тех пор, пока не завершится дочерний процесс.

Использованные системные вызовы:

pidt fork(void); – создает дочерний процесс. Если возвращает 0, то созданный процесс – ребенок, если > 0, то – родитель.
exit(int status); – выходит из процесса с заданным статусом.
pidt waitpid(pidt pid, int *status, int options); – приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится, или до появления сигнала, который либо завершает текущий процесс, либо требует вызвать функцию-обработчик.
int pipe(int *fd); – предоставляет средства передачи данных между двумя процессами.
int close(int fd); – закрывает файловый дескриптор.
int read(int fd, void *buffer, int nbyte); – читает nbyte из файлового дескриптора fd в буффер buffer.
int write(int fd, void *buffer, int nbyte); – записывает количество байтов в 3 аргументе из буфера в файл с дискриптором fd.

2 Исходный код

```c
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>

#define P1_READ 0
#define P2_WRITE 1

int fd[2];

int Fib(int n)
{
    pid_t pid1, pid2;
    int buf1, buf2, res, status, bufread1, bufread2;

    if (n == 0) {
        return 0;
    } else if (n == 1 || n == 2) {
        return 1;
    }

    pid1 = fork();
    if (pid1 == 0) { // child process 1 (fib(n-1))
        buf1 = Fib(n - 1);
        close(fd[P1_READ]);
        if(write(fd[P2_WRITE], &buf1, sizeof(buf1)) == -1) {
            perror("write");
        }
        exit(0);
    } else if (pid1 < 0) {
        perror("fork");
    } else if (pid1 > 0) {
        pid2 = fork();
        if (pid2 == 0) { // child process 2 (fib(n-2))
            buf2 = Fib(n - 2);
            close(fd[P1_READ]);
            if(write(fd[P2_WRITE], &buf2, sizeof(buf2)) == -1) {
                perror("write");
            }
            exit(0);
        } else if (pid2 < 0) {
            perror("fork");
        }
    }
```

```
48
49      if (waitpid(pid1, &status, 0) == -1) {
50          perror("waitpid");
51      }
52      if (waitpid(pid2, &status, 0) == -1) {
53          perror("waitpid");
54      }
55
56      if(read(fd[P1_READ], &bufread1, sizeof(bufread1)) == -1) {
57          perror("read");
58      }
59      if(read(fd[P1_READ], &bufread2, sizeof(bufread2)) == -1) {
60          perror("read");
61      }
62      res = bufread1 + bufread2;
63
64      return res;
65  }
66
67  int main(void)
68  {
69      int n = 0;
70
71      if (pipe(fd) == -1) {
72          perror("pipe");
73      }
74
75      printf("Enter a sequence number: ");
76      scanf("%d", &n);
77      if (n < 0) {
78          printf("Number must be > 0.\n");
79      } else {
80          printf("%d\n", Fib(n));
81      }
82
83      return 0;
84  }
```

3 Консоль

```
karma@karma:~/mai_study/OS/lab2$ ./run
Enter a sequence number: 0
0
karma@karma:~/mai_study/OS/lab2$ ./run
Enter a sequence number: 1
1
karma@karma:~/mai_study/OS/lab2$ ./run
Enter a sequence number: 2
1
karma@karma:~/mai_study/OS/lab2$ ./run
Enter a sequence number: -1
Number must be >0.
karma@karma:~/mai_study/OS/lab2$ ./run
Enter a sequence number: 9
34
karma@karma:~/mai_study/OS/lab2$ ./run
Enter a sequence number: 14
377
karma@karma:~/mai_study/OS/lab2$ ./run
Enter a sequence number: 18
2584
karma@karma:~/mai_study/OS/lab2$ ./run
Enter a sequence number: 20
6765
```

4 Выводы

Вычисление чисел Фибоначчи с помощью пайпов – довольно странное занятие, потому что рано или поздно на любой машине закончатся доступные пользователю процессы, и вычисление n-ого члена, если он достаточно большой, не будет выполнено. Еще это не очень эффективно. На моем компьютере после вычисления 20 члена начинаются проблемы с доступом к ресурсу. Интересно, что подсчет 20 члена при любом запуске заканчивается удачно, в то время как подсчет 21 члена от запуска к запуску меняется с удачного на неудачный и наоборот. Видимо есть какие-то ограничения процессов для пользователя. Такая ошибка может возникнуть либо из-за большого количества потоков, либо из-за ограничения используемых файловых дескрипторов. Последних я использую всего 2, поэтому можно предположить, что дело в большом количестве потоков.

Редко встретишь программу, которая выполняется в одном процессе или потоке. Поэтому нам, как программистам, необходимо уметь работать с ними. Очевидно, что в сложных программах используется большее количество пайпов чем один. Находясь в ситуации нравственного выбора (писать хороший код или не очень), важно сделать правильное архитектурное решение.