Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Операционные системы»

Межпроцессное взаимодействие

Студент: В. М. Ватулин
Преподаватель: А. А. Соколов
Группа: М8О-206Б-19
Дата: 17.04.2021
Оценка:
Подпись:

Москва, 2021

1 Постановка задачи

Цель работы:

Приобретение практических навыков в:

- Управление процессами в ОС
- Обеспечение обмена данных между процессами посредством каналов

Задание (вариант 20):

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Правило фильтрации: строки длины больше 10 символов отправляются в pipe2, иначе в pipe1. Дочерние процессы инвертируют строки.

2 Общие сведения о программе

Программа компилируется из файла main.c. В программе используются следующие системные вызовы:

1. **pipe** — принимает массив из двух целых чисел, в случае успеха массив будет содержать два файловых дескриптора, которые будут использоваться для конвейера, первое число в массиве предназначено для чтения, второе для записи, а так же вернется 0. В случае неуспеха вернется -1.

2. **fork** — создает новый процесс, который является копией родительского процесса, за исключением разных pid. В случае успеха fork() возвращает 0 для ребенка, число больше 0 для родителя – pid ребенка, в случае ошибки возвращает -1.

3. **open** — создает или открывает файл, если он был создан. В качестве аргументов принимает путь до файла, режим доступа (запись, чтение и т.п.), модификатор доступа (при создании можно указать права для файла). Возвращает в случае успеха файловый дескриптор – положительное число, иначе возвращает -1.

4. **close** — принимает файловый дескриптор в качестве аргумента, удаляет файловый дескриптор из таблицы дескрипторов, в случае успеха вернет 0, в случае неуспеха вернет -1.

5. **read** — предназначена для чтения какого-то числа байт из файла, принимает в качестве аргументов файловый дескриптор, буфер, в который будут записаны данные и число байт. В случае успеха вернет число прочитанных байт, иначе -1.

6. **write** — предназначена для записи какого-то числа байт в файл, принимает в качестве аргументов файловый дескриптор, буфер, из которого будут считаны данные для записи и число байт. В случае успеха вернет число записанных байт, иначе -1.

7. **prctl** – манипулирует аспектами поведения родительского процесса или потока. Принимает аспект поведения в виде числа, далее идут varargs, количество и значение которых зависит от первого аргумента.

3 Общий метод и алгоритм решения

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы pipe и fork.
2. Создать каналы связи для каждого из дочерних процессов
3. Написать код для создания дочерних процессов
4. Написать функцию работы родительского процесса
5. Написать функцию работы дочернего процесса
6. Написать обработку ошибок
7. Написать тесты

4 Исходный код

main.c

```
1
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <string.h>
5  #include <stdlib.h>
6  #include <signal.h>
7  #include <sys/prctl.h>
8
9  #include <fcntl.h>
10 #include <sys/stat.h>
11
12 #define PATH_MAX 4096
13 #define INPUT_BUFFER 4096
14 #define MAX_PROCESSES 2
15 #define READ 0
16 #define WRITE 1
17 #define BOUNDARY 10
18
19 #define STDIN 0
20
21 size_t my_read(char *buff, size_t max_bytes, int fd) {
22     char temp;
23     size_t i;
24     for (i = 0; i < max_bytes - 1; ++i) {
25         if (read(fd, &temp, sizeof(char)) == 0 || temp == '\n') {
26             break;
27         }
28         buff[i] = temp;
29     }
30     buff[i] = '\0';
31     return i;
32 }
33
34 void parent_death(int sig) {
35     exit(0);
36 }
37
38 void parentjob(int fd[MAX_PROCESSES][2]) {
39     for (int i = 0; i < MAX_PROCESSES; i++) {
40         close(fd[i][READ]);
41     }
42     char filename[PATH_MAX + 1];
43     for (int i = 0; i < MAX_PROCESSES; i++) {
44         //fgets(filename, PATH_MAX + 1, stdin);
45         my_read(filename, PATH_MAX + 1, STDIN);
46         write(fd[i][WRITE], filename, PATH_MAX + 1);
```

```
47      }
48      char input[INPUT_BUFFER + 1];
49      //while (fgets(input, INPUT_BUFFER + 1, stdin) != NULL) {
50      while (my_read(input, INPUT_BUFFER + 1, STDIN) != 0) {
51          if (strlen(input) - 1 <= BOUNDARY) {
52              write(fd[0][WRITE], input, INPUT_BUFFER + 1);
53          }
54          else {
55              write(fd[1][WRITE], input, INPUT_BUFFER + 1);
56          }
57      }
58  }
59
60  void childjob(int fd[2], pid_t parent_pid) {
61      prctl(PR_SET_PDEATHSIG, SIGTERM);
62      signal(SIGTERM, parent_death);
63      if (getppid() != parent_pid) {
64          parent_death(SIGTERM);
65      }
66      close(fd[WRITE]);
67      char filename[PATH_MAX + 1];
68      read(fd[READ], filename, PATH_MAX + 1);
69      //FILE *fp = fopen(filename, "w");
70      int file_des = open(filename, O_WRONLY | O_CREAT, S_IWUSR | S_IRUSR);
71      char input[INPUT_BUFFER + 1];
72      while (1) {
73          read(fd[READ], input, INPUT_BUFFER + 1);
74          size_t input_len = strlen(input);
75          for (int i = 0; i < input_len / 2; i++) {
76              char temp = input[i];
77              input[i] = input[input_len - 1 - i];
78              input[input_len - 1 - i] = temp;
79          }
80          //fputs(input, fp);
81          write(file_des, input, strlen(input));
82          write(file_des, "\n", 1);
83      }
84  }
85
86  int main() {
87      int fd[MAX_PROCESSES][2];
88      for (int i = 0; i < MAX_PROCESSES; i++) {
89          pipe(fd[i]);
90      }
91      pid_t parent_pid = getpid();
92      pid_t temp_pid = 1;
93      unsigned int id = 0;
94      for (int i = 0; i < MAX_PROCESSES; i++) {
95          if (temp_pid != 0) {
```

```
96              id = i;
97              temp_pid = fork();
98              if (temp_pid == -1) {
99                  perror("fork error");
100                 exit(1);
101             }
102         }
103         else {
104             break;
105         }
106     }
107     if (temp_pid == 0) {
108         childjob(fd[id], parent_pid);
109     }
110     else {
111         parentjob(fd);
112     }
113
114     return 0;
115 }
```

5 Пример работы

```
eri412@Eri-PC:~/Desktop/study/OS/OSlab2$ ./a.out
child1
child2
qwe
rty
qweqweqeqeqwqe
12311
asdadasdasdasd5345
65363636345353
sadadxzc
eri412@Eri-PC:~/Desktop/study/OS/OSlab2$ cat child1
ewq
ytr
11321
czxdadas
eri412@Eri-PC:~/Desktop/study/OS/OSlab2$ cat child2
eqwqeqeqewqewq
5435dsadsadsadadsa
35354363636356
```

6 Вывод

В процессе работы над данной лабораторной работой я научился основам работы с конвейерами и процессами в Си, а также минимально узнал, как работают сигналы в операционной системе. Процессы занимают важную роль в разработке ПО, так как программы зачастую состоят из нескольких, относительно обособленных, подпрограмм, то есть процессов. Конвейеры как один из способов обмена данными между процессами таже играют немаловажную роль.