Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Операционные системы»

Межпроцессное взаимодействие

Студент: В. М. Ватулин
Преподаватель: А. А. Соколов
Группа: М8О-206Б-19
Дата: 19.04.2021
Оценка:
Подпись:

Москва, 2021

1 Постановка задачи

Цель курсового проекта

1. Приобретение практических навыков в использовании знаний, полученных в течении курса
2. Проведение исследования в выбранной предметной области

Задание

Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Произвести анализ и сделать вывод на основании данных, полученных при работе программного прототипа.

Вариант на «удовлетворительно».

Необходимо написать три программы. Далее будем обозначать эти программы A, B, C. Программа A принимает из стандартного потока ввода строки, а далее их отправляет программе C. Отправка строк должна производится построчно. Программа C печатает в стандартый вывод, полученную строку от программы A. После получения программа C отправляет программе A сообщение о том, что строка получена. До тех пор пока программа A не примет «сообщение о получение строки» от программы C, она не может отправялять следующую строку программе C. Программа B пишет в стандартный вывод количество отправленных символов программой A и количество принятых символов программой C. Данную информацию программа B получает от программ A и C соответственно. Способ организация межпроцессного взаимодействия выбирает студент.

2 Общие сведения о программе

Код курсового проекта состоит из трех файлов: A.c, B.c и C.c, названия соответствуют программам из условия. При запуске программы A создаются 4 пайпа для связи между всеми программами, затем просходит два форка для создания дочерних программ (B и C). После форка в каждой программе закрываются ненужные стороны пайпов, образы дочерних программ заменяются на соответствующие программы с помощью $execl()$. Программы B и C используют $prctl()$ для того, чтобы они завершались в случае неожиданной смерти родителя. Далее основная программа A принимает ввод по строкам, отправляет длину ввода в B, а само сообщение в C. C отправляет длину принятого сообщения в B, выводит принятое сообщение и уведомляет программу A об успехе. Программа B, приняв длину от программ A и C сразу же выводит её.

3 Общий метод и алгоритм решения

Для реализации поставленной задачи необходимо:

1. Вспомнить принцип работы fork, pipe, prctl, execl
2. Написать код для создания программ B и C в программе A
3. Написать основной цикл программы A
4. Написать программу C
5. Написать программу B
6. Написать обработку ошибок

4 Исходный код

A.c

```
1
2  #include <stdlib.h>
3  #include <stdio.h>
4  #include <string.h>
5  #include <signal.h>
6
7  #include <unistd.h>
8
9  #define BUFF_SIZE 256
10
11 #define READ 0
12 #define WRITE 1
13 #define MAX_ARG_LEN 8
14
15
16 int main() {
17     int fd_A2C[2];
18     int fd_A2B[2];
19     int fd_C2A[2];
20     int fd_C2B[2];
21
22     if (pipe(fd_A2C) || pipe(fd_A2B) || pipe(fd_C2A) || pipe(fd_C2B)) {
23         perror("Error");
24         exit(EXIT_FAILURE);
25     }
26
27     pid_t parent_pid = getpid();
28
29     pid_t proc_C_pid = fork();
30     if (proc_C_pid == -1) {
31         perror("Error");
32         exit(EXIT_FAILURE);
33     }
34     else if (proc_C_pid == 0) {
35         close(fd_A2B[READ]);
36         close(fd_A2B[WRITE]);
37         close(fd_A2C[WRITE]);
38         close(fd_C2B[READ]);
39         close(fd_C2A[READ]);
40
41         char ppid[MAX_ARG_LEN + 1];
42         snprintf(ppid, MAX_ARG_LEN + 1, "%d", parent_pid);
43
44         char read_A[MAX_ARG_LEN + 1];
45         snprintf(read_A, MAX_ARG_LEN + 1, "%d", fd_A2C[READ]);
46
```

```
47          char write_B[MAX_ARG_LEN + 1];
48          snprintf(write_B, MAX_ARG_LEN + 1, "%d", fd_C2B[WRITE]);
49
50          char write_A[MAX_ARG_LEN + 1];
51          snprintf(write_A, MAX_ARG_LEN + 1, "%d", fd_C2A[WRITE]);
52
53          execl("./C", "./C", ppid, read_A, write_B, write_A, (char *) NULL);
54          kill(parent_pid, SIGABRT);
55      }
56
57      pid_t proc_B_pid = fork();
58      if (proc_B_pid == -1) {
59          perror("Error");
60          kill(proc_C_pid, SIGABRT);
61          exit(EXIT_FAILURE);
62      }
63      else if (proc_B_pid == 0) {
64          close(fd_A2C[READ]);
65          close(fd_A2C[WRITE]);
66          close(fd_C2A[READ]);
67          close(fd_C2A[WRITE]);
68          close(fd_A2B[WRITE]);
69          close(fd_C2B[WRITE]);
70
71          char ppid[MAX_ARG_LEN + 1];
72          snprintf(ppid, MAX_ARG_LEN + 1, "%d", parent_pid);
73
74          char read_A[MAX_ARG_LEN + 1];
75          snprintf(read_A, MAX_ARG_LEN + 1, "%d", fd_A2B[READ]);
76
77          char read_C[MAX_ARG_LEN + 1];
78          snprintf(read_C, MAX_ARG_LEN + 1, "%d", fd_C2B[READ]);
79
80          execl("./B", "./B", ppid, read_A, read_C, (char *) NULL);
81          kill(parent_pid, SIGABRT);
82      }
83
84      close(fd_C2B[READ]);
85      close(fd_C2B[WRITE]);
86      close(fd_C2A[WRITE]);
87      close(fd_A2C[READ]);
88      close(fd_A2B[READ]);
89
90      char buff[BUFF_SIZE + 1];
91      while (fgets(buff, BUFF_SIZE + 1, stdin) != NULL) {
92          size_t input_len = strlen(buff) - 1;
93          printf("A sent string: %s", buff);
94          write(fd_A2B[WRITE], &input_len, sizeof(size_t));
95          write(fd_A2C[WRITE], buff, BUFF_SIZE + 1);
```

```
96          int temp;
97          read(fd_C2A[READ], &temp, sizeof(int));
98      }
99
100     kill(proc_C_pid, SIGTERM);
101     kill(proc_B_pid, SIGTERM);
102
103     close(fd_A2C[WRITE]);
104     close(fd_A2B[WRITE]);
105     close(fd_C2A[READ]);
106
107     return 0;
108 }
```

C.c

```
1
2  #include <stdlib.h>
3  #include <stdio.h>
4  #include <string.h>
5  #include <signal.h>
6
7  #include <unistd.h>
8  #include <sys/prctl.h>
9
10 #define BUFF_SIZE 256
11
12 int read_A;
13 int write_B;
14 int write_A;
15
16 void parent_death(int sig) {
17     exit(EXIT_FAILURE);
18 }
19
20 void parent_end(int sig) {
21     close(read_A);
22     close(write_B);
23     close(write_A);
24     exit(EXIT_SUCCESS);
25 }
26
27 int main(int argc, char **argv) {
28     pid_t parent_pid = atoi(argv[1]);
29     prctl(PR_SET_PDEATHSIG, SIGABRT);
30     signal(SIGABRT, parent_death);
31     if (getppid() != parent_pid) {
32         parent_death(SIGABRT);
33     }
34     signal(SIGTERM, parent_end);
```

```
35
36      read_A = atoi(argv[2]);
37      write_B = atoi(argv[3]);
38      write_A = atoi(argv[4]);
39
40      char buff[BUFF_SIZE + 1];
41      while (1) {
42          read(read_A, buff, BUFF_SIZE + 1);
43          printf("C got string: %s", buff);
44          size_t input_len = strlen(buff) - 1;
45          write(write_B, &input_len, sizeof(size_t));
46          int temp = 1;
47          write(write_A, &temp, sizeof(int));
48      }
49
50      return 0;
51  }
```

B.c

```
1
2   #include <stdlib.h>
3   #include <stdio.h>
4   #include <string.h>
5   #include <signal.h>
6
7   #include <unistd.h>
8   #include <sys/prctl.h>
9
10  #define BUFF_SIZE 256
11
12  int read_A;
13  int read_C;
14
15  void parent_death(int sig) {
16      exit(EXIT_FAILURE);
17  }
18
19  void parent_end(int sig) {
20      close(read_A);
21      close(read_C);
22      exit(EXIT_SUCCESS);
23  }
24
25  int main(int argc, char **argv) {
26      pid_t parent_pid = atoi(argv[1]);
27      prctl(PR_SET_PDEATHSIG, SIGABRT);
28      signal(SIGABRT, parent_death);
29      if (getppid() != parent_pid) {
30          parent_death(SIGABRT);
```

```
31      }
32      signal(SIGTERM, parent_end);
33
34      read_A = atoi(argv[2]);
35      read_C = atoi(argv[3]);
36
37      size_t from_A;
38      size_t from_C;
39      while (1) {
40          read(read_A, &from_A, sizeof(size_t));
41          printf("B got length %zu from A\n", from_A);
42          read(read_C, &from_C, sizeof(size_t));
43          printf("B got length %zu from C\n", from_C);
44      }
45
46      return 0;
47  }
```

5 Пример работы

Пример работы программы:

```
eri412@Eri-PC:~/Desktop/study/OS/OSkp$ ./A
qwerty
A sent string: qwerty
C got string: qwerty
B got length 6 from A
B got length 6 from C

A sent string:
C got string:
B got length 0 from A
B got length 0 from C
i love linux <3
A sent string: i love linux <3
B got length 15 from A
C got string: i love linux <3
B got length 15 from C
eri412@Eri-PC:~/Desktop/study/OS/OSkp$ ps
PID TTY          TIME CMD
44745 pts/0    00:00:00 bash
44754 pts/0    00:00:00 ps
```

6 Вывод

В процессе работы над данным курсовым проектом я освежил основы межпроцессного взаимодействия с использованием технологии pipe и сигналов. Pipe является несложной технологией для межпроцессного взаимодействия и предлагает удобный интерфейс.

Также я выяснил более полно, как работает *execl*() - системный вызов замены образа процесса на другую программу - и таблицы файловых дескрипторов у процессов. Использование *execl*() упрощает написание программы, однако выяснил я это намного позже написания второй лабораторной работы, в которой обе программы были написаны в одном файле. Разделение кода на подобные логические единицы упрощает разработку и понимание кода.