Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу

«Операционные системы»

ВАРИАНТ НА УДОВЛЕТВОРИТЕЛЬНО

Студент: Суханов Е.А.
Группа: М8О–206Б–19
Вариант: на удовлетворительно
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

Постановка задачи

Цель работы

- Приобретение практических навыков в использовании знаний, полученных в течении курса;
- 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: Необходимо написать 3-и 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 производиться построчно. Программа C печатает в стандартный 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 отправлять 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.

Взаимодействие между программами будет происходить с помощью однонаправленных каналов.

Общие сведения о программе.

Программы компилируются из файлов a.c, b.c, c.c и io.c . Также используется заголовочные файлы: stdio.h, unistd.h, stdlib.h, signal.h, stdbool.h. В программе используются следующие системные вызовы:

pipe – создает канал и возвращает два файловых дескриптора, для общения по нему;

close – закрывает файловый дескриптор;

fork – создает дочерний процесс и продолжает выполнение текущей программы в нем;

execl – загружает в текущий процесс другую програму;

kill – посылает сигнал другому процессу.

Общий метод и алгоритм решения

1. Программа A создает 4 канала. Первые два канала направлены из процесса A в процессы B и C; Вторая пара каналов отвечает за отправку сообщений из процесса C в процессы A и B. Затем программа A создает процессы B и C и отдает им соответствующие файловые дескрипторы.

   Далее програма A считывает строку и отправляет два сообщения программе C: размер строки и саму строку. Программе B отправляется только размер строки;
2. Программа C считывает данные из канала. Сначала считывается размер, выделяется память, считывается соответствующее сообщение. Отправляется информация о длине считанной строки программе B. Выводится строка;
3. Программа B считывает сначала информацию по каналу из A, затем из C. Затем выводит эту информацию.

Библиотека io.h позволяет упростить io операции. Она была написана в результате выполнения других ЛР.

Основные файлы программы

a.c:

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include "io.h"

// from A
int pipeAC[2];
int pipeAB[2];
// from C
int pipeCA[2];
int pipeCB[2];

pid_t pid_b, pid_c;

void pipe_wrapper(int fds[2]){
  if(pipe(fds) == 0)
    return;

  perror("Ошибка создания pipe");
  exit(-1);
}

void close_wrapper(int fd) {
  if(close(fd) == -1) {
    perror("Ошибка закрытия файлового дескриптора");
    exit(-1);
  }
}

void init_pipes() {
  pipe_wrapper(pipeAC);
  pipe_wrapper(pipeAB);

  pipe_wrapper(pipeCA);
  pipe_wrapper(pipeCB);
}

void fork_b_wrapper() {
  pid_t pid = fork();
  if(pid == -1) {
    perror("Ошибка fork для процесса b");
    exit(-1);
  }
  else if(pid == 0) {
    char fd1[3], fd2[3];
    sprintf(fd1, "%d", pipeAB[0]);
    sprintf(fd2, "%d", pipeCB[0]);
```

```
    close_wrapper(pipeAB[1]);
    close_wrapper(pipeCB[1]);

    if(execl("b", "b", fd1, fd2, NULL) == -1) {
      perror("Ошибка execl для процесса b");
    }
  }
  pid_b = pid;
  close_wrapper(pipeAB[0]);
  close_wrapper(pipeCB[0]);
}

void fork_c_wrapper() {
  pid_t pid = fork();
  if(pid == -1) {
    perror("Ошибка fork для процесса c");
    exit(-1);
  }
  else if(pid == 0) {
    char fd1[3], fd2[3], fd3[3];
    sprintf(fd1, "%d", pipeAC[0]);
    sprintf(fd2, "%d", pipeCA[1]);
    sprintf(fd3, "%d", pipeCB[1]);

    close_wrapper(pipeAC[1]);
    close_wrapper(pipeCA[0]);

    if(execl("c", "c", fd1, fd2, fd3, NULL) == -1) {
      perror("Ошибка execl для процесса c");
    }
  }
  pid_c = pid;
  close_wrapper(pipeAC[0]);
  close_wrapper(pipeCA[1]);
  close_wrapper(pipeCB[1]);
}

void init_processes() {
  // process B
  fork_b_wrapper();
  // process C
  fork_c_wrapper();
}

void run() {
  int to_c = pipeAC[1];
  int to_b = pipeAB[1];
  int from_c = pipeCA[0];

  char* str;
```

```
  int len;
  while(str = get_line(&len), len > 0) {
    if(write_str(to_c, (char*)&len, sizeof(len)) == -1){
      perror("Ошибка записи");
      exit(-1);
    }
    if(write_str(to_c, str, len + 1) == -1){
      perror("Ошибка записи");
      exit(-1);
    }
    if(write_str(to_b, (char*)&len, sizeof(len)) == -1){
      perror("Ошибка записи");
      exit(-1);
    }
    free(str);

    //Ожидание ответа о получении сообщения
    int msg = 0;
    if(read_str(from_c, (char*)&msg, sizeof(msg)) == -1) {
      perror("Ошибка записи");
      exit(-1);
    }
  }
}

void deinit() {
  kill(pid_c, SIGTERM);
  kill(pid_b, SIGTERM);
}

int main() {
  init_pipes();
  init_processes();
  run();
  deinit();
  return 0;
}
```

b.c:

```
#include <stdio.h>
#include <stdlib.h>
#include "io.h"

int main(int argc, char* argv[]) {
  int from_a, from_c;
  from_a = atoi(argv[1]);
  from_c = atoi(argv[2]);
```

```
  int len_a, len_c;
  while(read_str(from_a, (char*)&len_a, sizeof(len_a)) == 0 &&
     read_str(from_c, (char*)&len_c, sizeof(len_c)) == 0) {
    printf("Процесс A отправил: %d символов\n"
       "Процесс B отправил: %d символов\n\n",
       len_a, len_c);
  }

  return 0;
}
```

c.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "io.h"

int main(int argc, char* argv[]) {
  int from_a, to_a, to_b;
  from_a = atoi(argv[1]);
  to_a = atoi(argv[2]);
  to_b = atoi(argv[3]);


  bool run = true;
  while(run){
    int len;
    if(read_str(from_a, (char*)&len, sizeof(len)) == -1){
      perror("Ошибка записи");
      exit(-1);
    }
    char* str = malloc(len+1);
    if(read_str(from_a, str, len+1) == -1){
      perror("Ошибка записи");
      exit(-1);
    }
    if(write_str(1, str, len) == -1){
      perror("Ошибка записи");
      exit(-1);
    }
    free(str);

    //Отправить принятый размер
    if(write_str(to_b, (char*)&len, sizeof(len)) == -1) {
      perror("Ошибка записи");
      exit(-1);
    }
    //Отправить сообщение, что все ОК
```

```
    int msg = 0;
    if(write_str(to_a, (char*)&msg, sizeof(msg)) == -1) {
      perror("Ошибка записи");
      exit(-1);
    }
  }

  return 0;
}
```

Пример работы

```
reterer@serv:~/OS/os_cp/src$ ./a
Hello world
Hello world
Процесс A отправил: 12 символов
Процесс B отправил: 12 символов

How are you?
How are you?
Процесс A отправил: 13 символов
Процесс B отправил: 13 символов

sooooo loooooooooooooooooooooooooooooooooooooooooong
sooooo loooooooooooooooooooooooooooooooooooooooooong
Процесс A отправил: 55 символов
Процесс B отправил: 55 символов
```

Вывод

Каналы хорошо подходят для многопроцессорного общения, но они требуют частого использования системных вызовов. Но в то же время их очень просто использовать, в отличии, от общей памяти, где могут возникать гонки данных.