Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

Студент: Люгге Т.В.

Группа: М8О-201Б-21

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2023

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/MonkeDLyugge/LabOS/tree/main/lab6\_8

**Постановка задачи**

Необходимо написать 3-и программы. Далее будем обозначать эти программы A, B, C.

Необходимо создать программу A, которая читает строки из стандартного ввода и посылает их по одной программе C. После отправки каждой строки, программа C отправляет сообщение об успешном получении строки обратно программе A. Программа A не должна отправлять следующую строку до получения подтверждения от программы C.

Программа B отслеживает количество отправленных символов программой A и количество принятых символов программой C. Для этого программа B получает информацию от программ A и C соответственно, и выводит эту информацию в стандартный вывод.

**Общие сведения о программе**

Программы написаны на языке C++ для Unix подобной операционной системы на базе ядра Linux. Для связи между процессами используется pipe

**Общий метод и алгоритм решения**

Программа А создает два дочерних процесса B и C, затем считывает строки из стандартного потока ввода, после чего передает строки процессу C

Процесс С пишет полученную строку, подтвердив получение строки процессу А

А отправляет размер отправленной строки программе B

C отправляет размер полученной строки программе B

**Исходный код**

|  |
| --- |
| **A.cpp** |
| #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  #include <signal.h>  #include <string.h>  #include "../include/get\_line.h"  int id1, id2;  int pipeAC[2];  int pipeAB[2];  int pipeCA[2];  int pipeCB[2];  void sig\_handler(int signal) {  kill(id1, SIGUSR1);  kill(id2, SIGUSR1);  close(pipeAC[0]);  close(pipeAC[1]);  close(pipeCA[0]);  close(pipeCA[1]);  close(pipeAB[0]);  close(pipeAB[1]);  close(pipeCB[0]);  close(pipeCB[1]);  exit(0);  }  int main(){  if (signal(SIGINT, sig\_handler) == SIG\_ERR) {  printf("[%d] ", getpid());  perror("Error signal ");  return -1;  }  pipe(pipeAC);  pipe(pipeAB);  pipe(pipeCA);  pipe(pipeCB);  id1 = fork();  if (id1 == -1)  {  perror("fork1");  exit(-1);  }  else if(id1 == 0)  {  char name[] = "./B";  char pAB[3] = "";  sprintf(pAB, "%d", pipeAB[0]);  char pCB[3] = "";  sprintf(pCB, "%d", pipeCB[0]);  close(pipeAC[0]);  close(pipeAC[1]);  close(pipeCA[0]);  close(pipeCA[1]);  close(pipeAB[1]);  close(pipeCB[1]);  execl(name, name, pAB, pCB, NULL);  }  else  {  id2 = fork();  if (id2 == -1)  {  perror("fork2");  exit(-1);  }  else if(id2 == 0)  {  char name[] = "./C";  char pAC[3] = "";  sprintf(pAC, "%d", pipeAC[0]);  char pCA[3] = "";  sprintf(pCA, "%d", pipeCA[1]);  char pCB[3] = "";  sprintf(pCB, "%d", pipeCB[1]);  close(pipeAC[1]);  close(pipeCA[0]);  close(pipeAB[0]);  close(pipeAB[1]);  close(pipeCB[0]);  execl(name, name, pAC, pCA, pCB, NULL);  }  else  {  char\* line = NULL;  int size;  while((size = get\_line(&line, STDIN\_FILENO)) != 0)  {  write(pipeAC[1], &size, sizeof(int));  write(pipeAC[1], line, size\*sizeof(char));  int ok;  read(pipeCA[0], &ok, sizeof(int));  write(pipeAB[1], &size, sizeof(int));  }  free(line);  kill(id1, SIGUSR1);  kill(id2, SIGUSR1);  }  }  close(pipeAC[0]);  close(pipeAC[1]);  close(pipeCA[0]);  close(pipeCA[1]);  close(pipeAB[0]);  close(pipeAB[1]);  close(pipeCB[0]);  close(pipeCB[1]);  } |

|  |
| --- |
| **B.cpp** |
| #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  #include <signal.h>  int pipeAB;  int pipeCB;  void sig\_handler(int signal) {  close(pipeAB);  close(pipeCB);  exit(0);  }  int main(int argc, char \*argv[]){  if (signal(SIGUSR1, sig\_handler) == SIG\_ERR) {  printf("[%d] ", getpid());  perror("Error signal ");  return -1;  }  pipeAB = atoi(argv[1]);  pipeCB = atoi(argv[2]);  int sizeA;  int sizeB;  while(read(pipeAB, &sizeA, sizeof(int)) > 0 && read(pipeCB, &sizeB, sizeof(int)) > 0){  printf("[B] Get A: %d; Get C: %d\n", sizeA, sizeB);  }  } |
| **C.cpp** |
| #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  #include <signal.h>  int pipeAC;  int pipeCA;  int pipeCB;  void sig\_handler(int signal) {  close(pipeAC);  close(pipeCA);  close(pipeCB);  exit(0);  }  int main(int argc, char \*argv[]){  if (signal(SIGUSR1, sig\_handler) == SIG\_ERR) {  perror("[C] Error signal ");  return -1;  }  pipeAC = atoi(argv[1]);  pipeCA = atoi(argv[2]);  pipeCB = atoi(argv[3]);  int sizeA;  while(read(pipeAC, &sizeA, sizeof(int))> 0){  char line[sizeA+1];  line[sizeA] = '\0';  read(pipeAC, line, sizeA \* sizeof(char));  printf("[C] Get from A: %s\n", line);  int ok = 1;  write(pipeCA, &ok, sizeof(int));  write(pipeCB, &sizeA, sizeof(int));  }  } |

**Демонстрация работы программ**

microhacker@microhacker-HLYL-WXX9:~/Desktop/LabOS$ ./A hello

[C] Get from A: hello [B] Get A: 5; Get C: 5

world

[C] Get from A: world [B] Get A: 5; Get C: 5

qwertyuioplkmasnfdkja

[C] Get from A: qwertyuioplkmasnfdkja

1. Get A: 21; Get C: 21

f

1. Get from A: f

[B] Get A: 22; Get C: 22

**Выводы**

Данная курсовая работа основывается на знаниях полученных в ходе изучения курса. По итогу мы получили нейколько программ, которые взаимодействуют друг с другом с помощью pipe. Задача курсового проекта не сложна в реализации, но ее реализация обобщает и закрепляет полученные в курсе знания.