МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Новосибирский государственный технический университет»

НГТУ НЭТИ | **Факультет прикладной математики и информатики**

Кафедра теоретической и прикладной информатики

Лабораторная работа №3
по дисциплине «Управление ресурсами в вычислительных системах»

Синхронизация процессов

Бригада 9 БЕГИЧЕВ АЛЕКСАНДР

Группа ПМ-92 ШИШКИН НИКИТА

Вариант 9

Преподаватели СИВАК МАРИЯ АЛЕКСЕЕВНА

СТАСЫШИН ВЛАДИМИР МИХАЙЛОВИЧ

Новосибирск, 2022

Цель работы

Практическое освоение механизма синхронизации процессов и их взаимодействие посредством програмных каналов.

Вариант 9

Исходный процесс создает два программных канала К1 и К2 и порождает два процесса Р1 и Р2, каждый из которых готовит данные для обработки их основным процессом. Подготавливаемые данные процесс Р2 помещает в канал К1, затем они оттуда читаются процессом Р1, переписываются в канал К2, дополняются своими данными. Схема взаимодействия процессов, порядок передачи данных в канал и структура подготавливаемых данных изображены ниже:

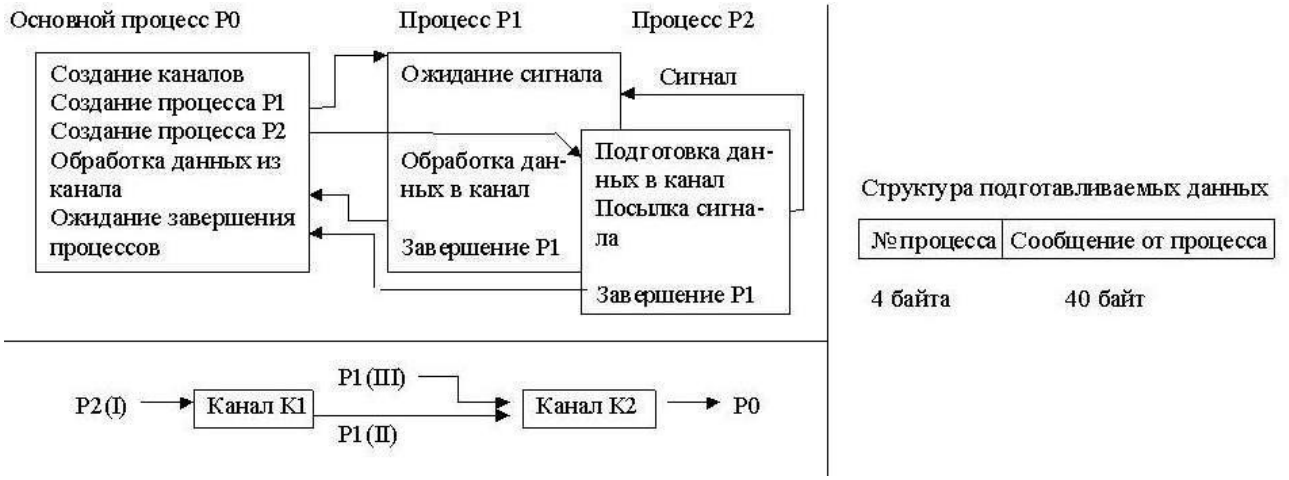

Рис. 1: Схема из методического пособия.

Обработка данных основным процессом заключается в чтении информации из программного канала К2 и печати её. Кроме того, посредством выдачи сообщений необходимо информировать обо всех этапах работы программы (создание процесса, завершение посылки данных в канал и т.д.).

Использованные функции для решения задачи

- `pipe()` – создаёт однонаправленный канал данных.
- `fork()` – создаёт новый процесс путём копирования текущего.
- `getpid()` – возвращает ID текущего процесса.
- `exit()` – вызывает прерывание текущего процесса.
- `waitpid()` – ожидает смену состояния (прерывание) процесса-потомка по его ID. Если состояние изменилось до вызова функции, то при вызове функции оно и вернётся.
- `close()` – закрывает описатель файла.
- `read()` – считывает поток байтов из файла (с помощью описателя) в буффер.
- `write()` – записывает поток байтов из буфера в файл по его описателю.

Листинги

main.h

```h
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

#define DATA_SIZE 44 // Размер структуры Data

// Структура с передаваемыми данными
typedef struct {
    pid_t pid;      // id процесса
    char msg[40];   // сообщение
} Data;

// Структура для наглядного
// представления каналов
typedef struct {
    int rcv; // получатель
    int snd; // отправитель
} Pipe;

```

main.c

```c
#include "main.h"

int main(int argc, char *argv[]) {
    Pipe pipe1, pipe2;
    Data data;
    pid_t pid1, pid2;

    // Создаём каналы
    pipe((int *) &pipe1);
    pipe((int *) &pipe2);


    // Первый дочерний процесс (т.н. P2)
    // ------------------------------------------------------------
    if ((pid1 = fork()) == -1) {
        perror("Fork failed\n");
        return EXIT_FAILURE;
    }

    if (pid1 == 0) {
        pid1 = getpid();
        printf("P1 with pid %d is created.\n", pid1);

        // Мы знаем, что это сообщение полностью не поместится в data.msg
        // Как раз не хватит места для имени убийцы :D
        const char *msg = " His name is DIO BRANDO.";

        // Вытаскиваем данные из канала K1
        read(pipe1.rcv, &data, DATA_SIZE);
        printf("P1 with pid %d have got data from pipe K1 from process with pid %d.\n", pid1, data.pid);

        // Дополняем полученные данные
        strncat(data.msg, msg, DATA_SIZE - 4 - strlen(data.msg));
        data.pid = pid1;

        // Отправляем данные по каналу K2
        write(pipe2.snd, &data, DATA_SIZE);
        printf("P1 with pid %d have sent data to pipe K2.\n", pid1);

        // Закрываем каналы
        close(pipe1.rcv);
        close(pipe1.snd);
        close(pipe2.rcv);
        close(pipe2.snd);

        exit(EXIT_SUCCESS);
    }


    // Второй дочерний процесс (т.н. P2)
    // ------------------------------------------------------------
    if ((pid2 = fork()) == -1) {
        perror("Fork failed\n");
        return EXIT_FAILURE;
    }

    if (pid2 == 0) {
        pid2 = getpid();
        printf("P2 with pid %d is created.\n", pid2);

        // Вот такое сообщение мы собираемся отправить
        const char *msg = "There is imposter among us.";

        // Заполняем данные в форму
        data.pid = pid2;
        strcpy(data.msg, msg);

        // Отправляем данные по каналу K1
        write(pipe1.snd, &data, DATA_SIZE);
        printf("P2 with pid %d have sent data to pipe K1.\n", pid2);

        // Закрываем каналы
        close(pipe1.rcv);
        close(pipe1.snd);
        close(pipe2.rcv);
        close(pipe2.snd);

        exit(EXIT_SUCCESS);
    }


    // В дело вступает родительский процесс (т.н. P0)
    // ------------------------------------------------------------

    // Вытаскиваем данные из канала K2
    read(pipe2.rcv, &data, DATA_SIZE);

    printf("P0 with pid %d have got data from pipe K2 from process with pid %d.\n", pid2, data.pid);
    printf("- Got message: %s\n", data.msg);

    // Закрываем каналы
    close(pipe1.rcv);
    close(pipe1.snd);
    close(pipe2.rcv);
    close(pipe2.snd);

    // Ждём завершения процессов-потомков
    waitpid(pid1, NULL, 0);
    waitpid(pid2, NULL, 0);

    return EXIT_SUCCESS;
}
```

Makefile

```
SHELL=/bin/bash

CC = gcc
CFLAGS = -c -g
LDFLAGS = -W

PROGRAM_NAME = lab3

SRCS := $(wildcard *.c)
OBJS := $(SRCS:%.c=%.o)
BINS := $(SRCS:%.c=%~)

.INTERMEDIATE: $(OBJS)

all: remove_bin link

remove_bin:
  rm -rvf $(PROGRAM_NAME)

link: $(OBJS)
  $(CC) $(LDFLAGS) $(OBJS) -o $(PROGRAM_NAME)

%~: %.o
  @echo "Linking "$<" to "$(PROGRAM_NAME)"..."
  $(CC) $(LDFLAGS) $< -o $(PROGRAM_NAME)

%.o: %.c
  @echo "Compiling "$<" to object file "$@"..."
  $(CC) $(CFLAGS) $< -o $@

clean:
  @echo "Cleaning up..."
  rm -rvf $(PROGRAM_NAME)
```

Запуск программы

Мы собираем прогрмму с помощью Make и запускаем:

```
alexander@Punch:~/D/u/w/s/o/l/c-src
➤ make
rm -rvf lab3
removed 'lab3'
Compiling main.c to object file main.o...
gcc -c -g main.c -o main.o
gcc -W main.o -o lab3
rm main.o
alexander@Punch:~/D/u/w/s/o/l/c-src
➤ ./lab3
P1 with pid 17914 is created.
P2 with pid 17915 is created.
P2 with pid 17915 have sent data to pipe K1.
P1 with pid 17914 have got data from pipe K1 from process with pid 17915.
P1 with pid 17914 have sent data to pipe K2.
P0 with pid 17915 have got data from pipe K2 from process with pid 17914.
- Got message: There is imposter among us. His name is
```

Рис. 2: Скриншот консоли со сборкой и выполнением программы.

У нас не получилось придумать тесты, которые будут блокировать создание дочерних процессов или каналов.