|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра теоретической и прикладной информатики | | |
|  | | |
| Лабораторная работа №3 | | |
| по дисциплине «Управление ресурсами в вычислительных системах». | | |
| «СИНХРОНИЗАЦИЯ ПРОЦЕССОВ» | | |
|  | | |
|  | | |
|  | Факультет: | ФПМИ |
| Группа: | ПМИ-12 |
| Студент: | Курдюков Иван, Омельницкая Екатерина |
| Преподаватели: | Стасышин В.М.,  Сивак М.А. |
|  |  |
|  |  |
|  |  |
|  |  |
| Новосибирск, 2024 | | |

1. **Цель работы**

Практическое освоение механизма синхронизации процессов и их взаимодействия посредством программных каналов.

1. **Задание**

Исходный процесс создает два программных канала К1 и К2 и порождает новый процесс Р1, а тот, в свою очередь, еще один процесс Р2, каждый из которых готовит данные для обработки их основным процессом. Подготавливаемые данные процесс Р1 помещает в канал К1, а процесс Р2 в канал К2, откуда они процессом Р1 копируются в канал К1 и дополняются новой порцией данных. Схема взаимодействия процессов, порядок передачи данных в канал и структура подготавливаемых данных показаны ниже:



Обработка данных основным процессом заключается в чтении информации из программного канала К1 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.).

1. **Описание используемых структур**

**Функции работы с каналами**

*int pipe(int filedes[2]);*

Возвращает два дескриптора файла: один для записи данных в канал, другой – для чтения.

*ssize\_t read(int \_fd, void \*\_buf, size\_t \_n)*

Выполняет чтение из файла по дескриптору \_fd не более \_n байт в память \_buf. Возвращает количество фактически прочитанных байт.

*ssize\_t write(int \_\_fd, const void \*\_\_buf, size\_t \_\_n)*

Записывает в файл по дескриптору \_fd не более \_n байт из памяти \_\_buf. Возвращает количество фактически записанных байт.

*int close(int \_\_fd)*

Закрывает дескриптор файла или канала.

**Функции работы с сигналами**

*int kill(pid\_t \_\_pid, int \_\_sig)*

Отправляет сигнал \_sig процессу с указанным pid, если pid положителен.

**Функции работы с процессами**

*pid\_t getpid()*

Возвращает идентификатор текущего процесса.

*pid\_t wait(void \*\_\_stat\_loc)*

Ожидает поступления сигнала SIGCHLD от дочернего процесса о его завершении. Возвращает pid завершившегося процесса.

1. **Спецификация**

Программа разработана и протестирована на компьютере с Cygwin. Cygwin — UNIX-подобная среда и интерфейс командной строки для Microsoft Windows. В качестве компилятора используется GCC версии 11.2.0.

Директория, содержащая файл: C:\cygwin64\home\Ivan

Название файла с программой: upres3lab\_2.c

Название файла для вывода: out.txt

Компиляция программы: gcc -o <результат компиляции> upres3lab\_2.c

Запуск программы: ./<результат компиляции>

1. **Описание алгоритма на языке Си**
2. **Создание каналов:** В начале программы создаются два канала, *K1* и *K2*, с помощью функции *pipe()*. Каждый канал состоит из двух файловых дескрипторов: один для чтения (*RD*) и один для записи (*WR*).
3. **Создание процесса P1:** После создания каналов процесс *main()* создаёт новый процесс, *P1*, с помощью функции *fork()*. После успешного создания *P1* становится дочерним процессом.
4. **Взаимодействие через каналы в P1:** Процесс *P1* закрывает файловый дескриптор для чтения канала *K1* (чтобы не получать данные от родительского процесса). Отправляет сообщение " sky is cloudy" через канал *K1* с помощью функции *write()*. Создаёт процесс *P2*, также с помощью *fork()*.
5. **Взаимодействие через каналы в P2:** Процесс *P2* закрывает файловый дескриптор для чтения канала *K2*. Записывает сообщение " weather's nice!" в канал *K2* с помощью функции write(). Отправляет сигнал *SIGUSR2* процессу *P1*, чтобы активировать его.
6. **Продолжение выполнения P1:** Процесс *P1* закрывает файловый дескриптор для записи канала *K2* (так как он не нужен). Отправляет сигнал *SIGUSR1* процессу *P2*, чтобы активировать его. Ожидает завершения процесса *P2*. После завершения процесса *P2*, считывает данные из канала *K2*, записывает их в канал *K1* и отправляет дополнительное сообщение в канал *K1*.
7. **Продолжение выполнения родительского процесса (P0):** Родительский процесс закрывает конец для записи дескриптора К1. (так как он более не нужен). Затем он открывает файл для записи данных. Затем обрабатывает данные из канала К1. Закрывает файл. Ожидает завершения процесса Р1. Закрывает все концы файловых дескрипторов (так как они не нужны). Прекращает работу.
8. **Код программы на языке Си**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <signal.h>

volatile sig\_atomic\_t flag = 0;

void signal\_handler(int signum) {

    if (signum == SIGUSR1 ||signum == SIGUSR2) {

        flag = 1;

    }

}

int main() {

    int k1[2], k2[2]; // сюда программный канал будет возвращать два дескриптора файла (один на запись второй на чтение)

    pid\_t pid1, pid2;

    signal(SIGUSR1, SIG\_IGN);

    signal(SIGUSR2, SIG\_IGN);

    if (pipe(k1) == -1 || pipe(k2) == -1) {

        perror("pipe");

        exit(EXIT\_FAILURE);

    }

    printf("Исходный процесс создал два программных канала К1 и К2\n");

    pid1 = fork(); // породили новый процесс р1

    if (pid1 < 0)

    { // проверяем создался ли процесс

        perror("fork (pid1)");

        exit(EXIT\_FAILURE);

    }

    else if (pid1 == 0)

    { // Р1 процесс

        close(k1[0]); // закрываем конец для чтения первого канала, он нам не понадобится ни в Р1 ни в Р2

        printf("Создан процесс Р1\n");

        // создаем второй дочерний процесс внутри первого дочернего

        pid2 = fork();

        if (pid2 < 0)

        {

            perror("fork (pid2)");

            exit(EXIT\_FAILURE);

        }

        else if (pid2 == 0)

        { // Р2 процесс

            printf("Создан процесс Р2\n");

            close(k2[0]);

            printf("Child: Waiting for signal from parent...\n");

            signal(SIGUSR1, signal\_handler);

            pause();

            if (flag) { printf("Child: Received SIGUSR1 signal from parent\n"); }

            // Готовим данные и помещаем их в канал К2

            FILE \*inp = fopen("in.txt", "r");

            if (inp == NULL) {

                perror("Unable to open input file");

                return 1;

            }

            char data[40];

            fgets(data, sizeof(data), inp);

            fgets(data, sizeof(data), inp);

            fclose(inp);

            int id = getpid();

            char input[44];

            char ID[4];

            sprintf(ID, "%i", id);

            for (int i = 0; i < 4; i++)

            {

                input[i] = ID[i];

            }

            for (int i = 0; i < 40; i++)

            {

                input[i + 4] = data[i];

            }

            write(k2[1], input, sizeof(data));

            printf("Процесс Р2 завершил посылку данных в канал К2\n");

            close(k2[1]);

            kill(getppid(), SIGUSR2); // отправляем сигнал процессу Р1

            printf("Процесс Р2 отправил сигнал P1\n");

            exit(0);

        }

        else

        { // процесс р1

            // Готовим данные и помещаем их в канал К1

            FILE \*in = fopen("in.txt", "r");

            if (in == NULL) {

                perror("Unable to open input file");

                return 1;

            }

            char data[40], input[44], ID[4];

            fgets(data, sizeof(data), in);

            int id = getpid();

            int param = sprintf(ID, "%i", id);

            for (int i = 0; i < 4; i++)

            {

                input[i] = ID[i];

            }

            for (int i = 0; i < 40; i++)

            {

                input[i+4] = data[i];

            }

            write(k1[1], input, sizeof(input));

            printf("Процесс Р1 завершил посылку данных в канал К1\n");

            printf("Parent: Sending SIGUSR1 signal to child\n");

            kill(pid1, SIGUSR1);

            printf("Parent: Ready to receive SIGUSR2 signal\n");

            flag = 0;

            signal(SIGUSR2, signal\_handler);

            pause();

            if (flag){printf("Parent: Received SIGUSR2 signal from child\n");}

            read(k2[0], input, sizeof(input));

            write(k1[1], input, sizeof(input));

            while (fgets(data, sizeof(data), in) != NULL) {}

            for (int i = 0; i < 4; i++)

            {

                input[i] = ID[i];

            }

            for (int i = 0; i < 40; i++)

            {

                input[i +4] = data[i];

            }

            write(k1[1], input, sizeof(input));

            wait(NULL); // Ожидаем завершения процесса P2

            close(k2[1]); // закрываем все оставшиеся концы каналов и файл

            fclose(in);

            close(k2[0]);

            close(k1[1]);

            exit(0);

        }

    }

    else

    { // Основной процесс

        close(k1[1]);

        printf("Основной процесс начинает обработку данных\n");

        // Чтение информации из канала К1 и печать

        char buffer[44];

        FILE \*out = fopen("out.txt", "w");

        ssize\_t bytesRead;

        while (bytesRead = read(k1[0], buffer, sizeof(buffer)))

        {

            if (bytesRead > 0)

            {

                buffer[bytesRead] = '\0';

                printf("Информация из канала К1: %s\n", buffer);

                fprintf(out, "%s\n", buffer);

            }

        }

        printf("Основной процесс завершил обработку данных\n");

        wait(NULL);

        close(k1[0]);

        close(k2[0]);

        close(k2[1]);

    }

    return 0;

}

1. **Тестирование**

