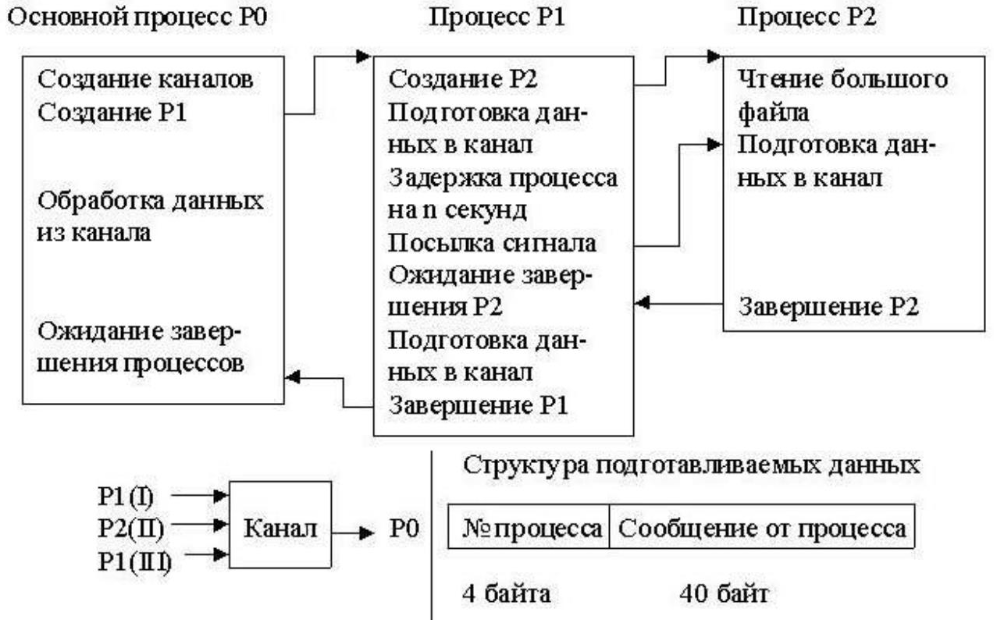
|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
| Лабораторная работа № 3 | | |
| по дисциплине «Управление ресурсами в вычислительных системах» | | |
| **Синхронизация процессов** | | |
|  | | |
|  | Бригада 7 | Грушев андрей |
| Группа ПМ-05 | болдырев сергей |
| Вариант 7 | хабарова анастасия |
|  |  |
|  |  |
|  |  |
| Преподаватели | стасышин владимир михайлович |
|  | сивак мария алексеевна |
| Новосибирск, 2023 | | |

# Условие

Исходный процесс создает программный канал К1 и порождает новый процесс Р1, а тот, в свою очередь, порождает ещё один процесс Р2. Подготовленные данные последовательно помещаются процессами-сыновьями в программный канал и передаются основному процессу. Файл, читаемый процессом Р2, должен быть достаточно велик с тем, чтобы его чтение не завершилось ранее, чем закончится установленная задержка в n секунд. После срабатывания будильника процесс Р1 посылает сигнал процессу Р2, прерывая чтение файла. 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 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.).

# Используемые программные средства

**signal(int sig, func)** – системный вызов, позволяющий процессу самостоятельно определить свою реакцию на получение сигнала. Реакцией процесса, осуществившего системный вызов с аргументом func, при получении сигнала sig будет вызов функции func().

**pipe(fd)** – системный вызов, возвращающий два дескриптора файла: для записи данных в канал и для чтения.

**fork(void)** – системный вызов, порождающий новый дочерний процесс.

**wait(int \*status)** – системный вызов, с помощью которого выполняется ожидание завершения процесса-потомка родительским процессом.

**exit(int status)** – системный вызов, предназначенный для завершения функционирования процесса. Аргумент status является статусом завершения, который передается родительскому процессу, если он выполнял системный вызов wait.

**fprintf(FILE\* stream, const char \*format, …)** – форматированный вывод в файл на который указывает stream.

**printf(const char\* format, …)** – форматированный вывод в файл стандартного вывода.

**sprintf(char\* str, const char\* format, …)** – форматированный вывод в символьную строку на которую указывает str.

**read(int fd, void \*buf, size\_t count)** – производит запись count байтов файлового описателя fd в буфер, адрес которого начинается с buf.

**write(int fd, const void \*buf, size\_t count)** – производит запись в описатель файла. Возвращает количество записанных байтов в случае успешного завершения, иначе возвращает -1.

**getpid(void)** – возвращает идентификатор текущего процесса.

**alarm(n)** – системный вызов, обеспечивающий посылку процессу сигнала SIGALARM через n секунд.

**pause()** – системный вызов, позволяющий приостановить процесс до тех пор, пока не будет получен какой-либо сигнал.

**kill(int pid, int sig)** – системный вызов, посылающий сигнал, специфицированный аргументом sig, процессу, который имеет идентификатор pid или группе процессов.

**open(const char \*pathname, int flags)** – преобразовывает путь к файлу в описатель файла. Возвращает файловый описатель, который не открыт процессом. Функция с флагом O\_TRUNC урезает длину файла до нуля, если файл существует, является обычным файлом и режим позволяет запись в этот файл.

# Алгоритм решения

1. Исходный процесс создает программный канал и порождает новый процесс.
2. Дочерний процесс порождает новый процесс, помещает подготовленные данные в канал.
3. Внучатый процесс помещает подготовленные данные в канал и завершается.
4. Дочерний процесс возобновляется после получения сигнала, помещает подготовленные данные в канал и завершается.
5. Родительский процесс при каждом обновлении содержимого программного канала считывает из него информацию и выводит на экран.

# Спецификация

Код программы расположен на сервере НГТУ в директории **/home/NSTU/pmi-b0507/upres/lab3**. Файлы с кодом на языке **C** – **main.c**.

Для корректной работы программы необходимо, чтобы в папке с исполняемым файлом находился файл большой размерности с наименованием **big\_file**. Для заполнения файла большой размерности существует вспомогательная программа **recorder.c**. Для получения исполняемого файла необходимо, находясь в директории с файлом **recorder.c,** выполнить команду:

**gcc recorder.с -o [имя\_исполняемого\_файла]**,

либо воспользоваться make-файлом при помощи команды:  
**make recorder**,

которая создаст исполняемый файл **recorder.o**.

Для получения основного исполняемого файла необходимо, находясь в директории с файлом **main.c,** выполнить команду:

**gcc main.с -o [имя\_исполняемого\_файла]**,

либо воспользоваться make-файлом при помощи команды:  
**make main**,

которая создаст исполняемый файл **main.o**.

Также можно воспользоваться командой **make all,**

которая создаст исполняемые файлы **main.o** и **recorder.o**.

Запуск программы происходит при помощи команды:

**./[имя\_исполняемого\_файла]**,

например:

**./main.o**

Формат вывода результата:

**[идентификатор процесса]: [сообщение от процесса]**

**…**

# Тестирование программы:

|  |  |  |
| --- | --- | --- |
| № | Входные данные | Результаты работы программы |
| 1 | ./main.o | 45649: Process is created  45650: Process start reading  45649: Process 45650 ended with status 2 |

# Make-файлы

Файл **makefile:**

main: main.c

    gcc main.c -o main.o

recorder: recorder.c

    gcc recorder.c -o recorder.o

all: main.c recorder.c

    gcc main.c -o main.o

    gcc recorder.c -o recorder.o

clean:

    rm main.o

    rm recorder.o

# Листинг программы

main.c:

#include <stdio.h>

#include <errno.h>

#include <string.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <fcntl.h>

#include <signal.h>

#include <stdbool.h>

/\*\*

 \* Функция-обработчик сигнала SIGLARM

\*\*/

void wakeup()

{

    signal(SIGALRM, wakeup);

}

int main(int argc, char\*\* argv)

{

    int k1[2];

    pipe(k1); //Канал для передачи сообщений между процессами

    signal(SIGALRM, wakeup); //Назначение обработчика сигнала SIGALRM

    pid\_t p1 = fork();

    if (p1 == -1)

    {

        fprintf(stderr, "P0: Unable to fork P1.\n");

        exit(EXIT\_FAILURE);

    }

    else if (p1 > 0)    //Процесс P0

    {

        char recievedMsg[40]; //Полученное сообщение

        int i, status, spid; //spid - sender process id

        for (i = 0; i < 3; i++) //Планируется получить 3 сообщения

        {

            read(k1[0], &spid, sizeof(int));

            read(k1[0], &recievedMsg, 40);

            printf("%d: %s\n", spid, recievedMsg);

        }

        wait(&status);

        exit(EXIT\_SUCCESS);

    }

    else    //Процесс P1

    {

        pid\_t p2 = fork();

        if (p2 == -1)

        {

            fprintf(stderr, "P1: Unable to fork P2.\b");

            exit(EXIT\_FAILURE);

        }

        else if (p2 > 0) // Процесс P1

        {

            char msg[40];

            pid\_t pid = getpid();

            int status;

            sprintf(msg, "Process is created\0");

            write(k1[1], &pid, sizeof(int)); //Отправка сообщения через канал

            write(k1[1], &msg, 40);

            alarm(3); //Будильник на 3 секунды

            pause();

            if (kill(p2, SIGINT) == -1) //Отправка SIGINT дочернему процессу

                fprintf(stderr, "%s\n", strerror(errno));

            wait(&status); //Ожидание завершения дочернего процесса

            sprintf(msg, "Process %d ended with status %d\0", p2, status);

            write(k1[1], &pid, sizeof(int)); //Отправка сообщения через канал

            write(k1[1], &msg, 40);

            exit(EXIT\_SUCCESS);

        }

        else    //  Процесс P2

        {

            int bf = open("big\_file", O\_RDONLY); //Открытие файла большого размера

            int number;

            char msg[40];

            pid\_t pid = getpid();

            sprintf(msg, "Process start reading\0");

            write(k1[1], &pid, sizeof(int)); //Отправка сообщения через канал

            write(k1[1], &msg, 40);

            while(true)

                read(bf, &number, sizeof(int)); //Чтение из файла, пока процесс не получит сигнал

        }

    }

    return 0;

}

recorder.c:

#include <unistd.h>

#include <stdlib.h>

#include <sys/types.h>

int main(int argc, char\*\* argv)

{

    int result = 0;

    int fd = open("big\_file", O\_RDONLY);

    while (read(fd, &result, sizeof(int)) == sizeof(int))

    {

        ;

    }

    close(fd);

    return 0;

}