Министерство образования и науки 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»

Факультет: ПМИ

Группа: ПМ-92

Вариант: 8

Студенты: Иванов В., Кутузов И.

Преподаватели: Стасышин В.М., Сивак М.А.

Новосибирск

2022

1. Цель работы

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

2. Задание

Исходный процесс создает программный канал К1 и порождает два процесса Р1 и Р2, каждый из которых готовит данные для обработки их основным процессом. Подготовленные данные последовательно помещаются процессами-сыновьями в программный канал и передаются основному процессу.

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.).

3. Описание используемых структур

int fork()
Порождение процесса-потомка, точной копии процесса-предка. Процесс-потомок в качестве возвращаемого значения системного вызова fork() получает 0, а процесс-предок - идентификатор процесса-потомка.

void (*signal (int signal, void (*sigfunc) (int func)))(int)
Дает указание выполнить функцию, на которую указывает sigfunc, в случае получения сигнала signal.

int pipe (int filedes[2])
Создает канал и помещает дескрипторы файла для чтения и записи (соответственно) в filedes [0] и filedes [1]. При успехе pipe возвращает значение 0. При отказе, -1.

int pause(void)

После вызова функции pause вызывающий процесс (или подзадача) приостанавливается до тех пор, пока не получит сигнал. Данный сигнал либо остановит процесс, либо заставит его вызвать функцию обработки этого сигнала.

int write(int handle, void *buf, int count)
Переписывает count байт из буфера, на который указывает buf, в файл, соответствующий дескриптору файла handle. Указателю положения в файле дается приращение на количество записанных байт. Если файл открыт в текстовом режиме, то символы перевода строки автоматически дополняются символами возврата каретки.

int kill(pid_t pid, int sig)
Системный вызов kill может быть использован для посылки какого-либо сигнала какому-либо процессу или группе процесса. Если значение pid является положительным, сигнал sig посылается процессу с идентификатором pid. Если pid равен 0, то sig посылается каждому процессу, который входит в группу текущего процесса. Если pid равен -1, то sig посылается каждому процессу, за исключением процесса с номером 1 (init), но есть нюансы, которые описываются ниже. Если pid меньше чем -1, то sig посылается каждому процессу, который входит в группу процесса -pid. Если sig равен 0, то никакой сигнал не посылается, а только выполняется проверка на ошибку.

void exit(int value)
Выполняет немедленное завершение программы. Аргумент параметра value возвращается принимающей стороной (ОС или другой программой) в родительский процесс. Как правило, возвращается значение 0 или EXIT_SUCCESS указывает на успешное завершение программы, и любое другое значение или значение макроса EXIT_FAILURE используется для указания об аварийном завершении программы.

int read(int fd, void *buf, unsigned count)
Считывает count байт из файла, описываемого аргументом fd, в буфер, на который указывает аргумент buf. Указателю положения в файле дается приращение на количество считанных байт.

pid_t wait(int *status)
Приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится, или до появления сигнала, который либо завершает текущий процесс, либо требует вызвать функцию-обработчик.

4. Спецификация

Программа разработана и протестирована на компьютере с операционной системой Linux. В качестве компилятора используется GCC версии 11.2.0.

Рабочая директория: **/home/lenferdetroud/Repos/university-tasks/Unix Resource Management/Process Synchronization**
Название файла с программой: **process_synchronization.c**
Компиляция программы: **gcc -o <результат компиляции> process_synchronization.c**
Запуск программы: **./<результат компиляции>**

5. Описание алгоритма

1. Создаем программный канал
2. Создаем первый дочерний процесс
3. Приостанавливаем первый дочерний процесс для ожидания сигнала от второго
4. Создаем второй дочерний процесс и отправляем сигнал первому
5. Получаем идентификатор второго дочернего процесса; записываем его и сообщение от второго дочернего процесса в программный канал
6. Получаем идентификатор первого дочернего процесса; записываем его и сообщение от первого дочернего процесса в программный канал
7. Внутри второго процесса завершаем работу первого, затем – второго
8. В родительском процессе обрабатываем сообщения: если дочерние процессы были успешно завершены, то читаем данные из программного канала и выводим на экран

6. Код программы

```
#include <signal.h>
#include <wait.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <math.h>

int fd[2], PID1, PID2, PID_CUR, p_number, ret_code, w;
char str_cur[40];

void start(int s) функция, вызываемая signal()
{
    printf("Process P1 (%d) received the signal\n", getpid());
}

int main()
{
    if (pipe(fd) == 0) создание программного канала
        printf("K1 was created\n");
    else
    {
        exit(1);
        printf("Error: Unable to create a channel!\n");
```

```
}

signal(SIGUSR1, start); вызов функции start() отправкой пользовательского сигнала

PID1 = fork(); создание первого процесса

if (PID1 == 0)
{
    printf("P1 was created\n");
    pause(); ожидание сигнала от второго процесса
    PID_CUR = getpid(); получение идентификатора текущего процесса
    char str_cur_1[] = "Hello from P1"; сообщение первого процесса
    write(fd[1], &PID_CUR, sizeof(int)); отправка идентификатора в канал
    write(fd[1], &str_cur_1, sizeof(str_cur_1)); отправка сообщения в канал
    printf("Data sent from P1 to K1\n");
    exit(0); завершение процесса
}

if (PID1 == -1) проверка на создание процесса
{
    exit(1);
    printf("Error: Unable to create P1!\n");
}

PID2 = fork(); создание второго процесса

if (PID2 == 0)
{
    printf("P2 was created\n");
    PID_CUR = getpid();
    char str_cur_2[] = "Hello from P2";
    write(fd[1], &PID_CUR, sizeof(int));
    write(fd[1], &str_cur_2, sizeof(str_cur_2));
    kill(PID1, SIGUSR1); завершение первого процесса
    printf("Data sent from P2 to K1\n");
    printf("A signal is sent from P2 to P1\n");
    exit(0);
}

if (PID2 == -1)
{
    exit(1);
    printf("Error: Unable to create P2!\n");
}

if (PID1 && PID2) обработка данных родительским процессом
{
    read(fd[0], &p_number, sizeof(int)); чтение идентификатора из канала
    read(fd[0], &str_cur, sizeof(str_cur)); чтение сообщения из канала
    printf("%s (%d)\n", str_cur, p_number);
    wait(&ret_code); ожидание завершения второго процесса
    if (WIFEXITED(ret_code) != 0) проверка на завершение второго процесса
        printf("P2 is finished\n");
```

```
        аналогично для первого процесса
        read(fd[0], &p_number, sizeof(int));
        read(fd[0], &str_cur, sizeof(str_cur));
        printf("%s (%d)\n", str_cur, p_number);
        wait(&ret_code);
        if (WIFEXITED(ret_code) != 0)
            printf("P1 is finished\n");
    }

    printf("Parent process finished\n");
    return 0;
}
```

7. Тестирование

Результат выполнения команды `gcc -o a.out process_synchronization.c ; ./a.out`:

```
K1 was created
P1 was created
P2 was created
Data sent from P2 to K1
A signal is sent from P2 to P1
Process P1 (322760) received the signal
Data sent from P1 to K1
Hello from P2 (322761)
P2 is finished
Hello from P1 (322760)
P1 is finished
Parent process finished
```