Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОТЕХНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

**ОТЧЁТ**

к лабораторной работе №2 на тему

**«ПРОЦЕССЫ В ОС LINUX»**

по дисциплине «ОСМУ»

Выполнил студент: К.Г. Хоменок

Проверил: А.Д. Станкевич

Минск 2024

**Цель работы:** изучение вопросов порождения и взаимодействия процессов в ОС LINUX.

**1 Краткие теоретические сведения**

В ОС Linux для создания процессов используется системный вызов fork():

#include <sys/types.h>

#include <unistd.h>

pid\_t fork (void);

В результате успешного вызова **fork()** ядро создаёт новый процесс, который является почти точной копией вызывающего процесса. Другими словами, новый процесс выполняет копию той же программы, что и создавший его процесс, при этом все его объекты данных имеют те же самые значения, что и в вызывающем процессе. Созданный процесс называется дочерним процессом, а процесс, осуществивший вызов **fork()**, называется родительским. После вызова родительский процесс и его вновь созданный потомок выполняются одновременно, при этом оба процесса продолжают выполнение с оператора, который следует сразу же за вызовом **fork()**. Процессы выполняются в разных адресных пространствах, поэтому прямой доступ к переменным одного процесса из другого процесса невозможен.

**2 Порядок выполнения работы**

1 Написать программу, создающую два дочерних процесса использованием двух вызовов **fork()**. Родительский и два дочерних процесса должны выводить на экран свой **pid** и **pid** родительского процесса и текущее время в формате: часы: минуты: секунды: миллисекунды. Используя вызов **system()**, выполнить команду **ps -x** в родительском процессе. Найти свои процессы в списке запущенных процессов.

Написанный код для программы:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#include <sys/time.h>

// Функция для получения текущего времени в формате часы:минуты:секунды:миллисекунды

void get\_current\_time(char \*buffer) {

struct timeval tv;

gettimeofday(&tv, NULL);

long milliseconds = tv.tv\_usec / 1000;

sprintf(buffer, "%ld:%ld:%ld:%ld", (tv.tv\_sec / 3600) % 24, (tv.tv\_sec / 60) % 60, tv.tv\_sec % 60, milliseconds);

}

int main() {

pid\_t pid1, pid2;

char time\_buffer[32];

// Создание первого дочернего процесса

pid1 = fork();

if (pid1 < 0) {

perror("Ошибка при создании первого дочернего процесса");

exit(EXIT\_FAILURE);

} else if (pid1 == 0) {

// Этот код выполняется в первом дочернем процессе

printf("PID первого дочернего процесса: %d, PPID: %d\n", getpid(), getppid());

get\_current\_time(time\_buffer);

printf("Текущее время: %s\n", time\_buffer);

exit(EXIT\_SUCCESS);

} else {

// Код родительского процесса

// Создание второго дочернего процесса

pid2 = fork();

if (pid2 < 0) {

perror("Ошибка при создании второго дочернего процесса");

exit(EXIT\_FAILURE);

} else if (pid2 == 0) {

// Этот код выполняется во втором дочернем процессе

printf("PID второго дочернего процесса: %d, PPID: %d\n", getpid(), getppid());

get\_current\_time(time\_buffer);

printf("Текущее время: %s\n", time\_buffer);

exit(EXIT\_SUCCESS);

} else {

// Этот код выполняется в родительском процессе

printf("PID родительского процесса: %d\n", getpid());

printf("Запуск ps -x...\n");

// Выполнение команды ps -x с помощью system()

system("ps -x");

// Ожидание завершения обоих дочерних процессов

wait(NULL);

wait(NULL);

}

}

return 0;

}

Необходимо проверить работу кода, поэтому запустим скомпилированную программу (рисунок 1).

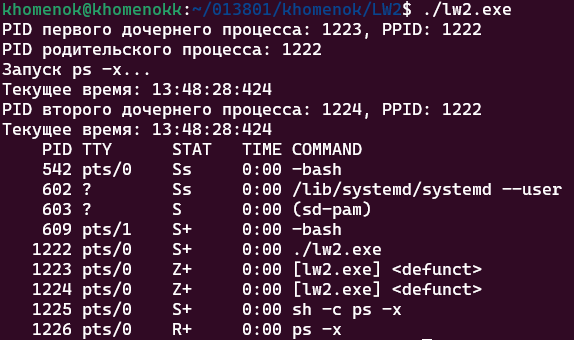


Рисунок 1 – Работа программы

Таким образом, было выполнено общее задание, далее перейдем к индивидуальному.

**3 Индивидуальное задание**

Написать программу синхронизации двух каталогов, например, Dir1 и Dir2. Пользователь задаёт имена Dir1 и Dir2. В результате работы программы файлы, имеющиеся в Dir1, но отсутствующие в Dir2, должны скопироваться в Dir2 вместе с правами доступа. Процедуры копирования должны запускаться в отдельном процессе для каждого копируемого файла. Каждый процесс выводит на экран свой pid, имя копируемого файла и число скопированных байт. Число одновременно работающих процессов не должно превышать N (вводится пользователем).

Написанный код для программы:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <dirent.h>

#include <string.h>

#include <errno.h>

#include <sys/wait.h>

#define MAX\_PATH\_LENGTH 1024

ssize\_t copy\_file(const char \*source, const char \*destination) {

char buffer[4096];

ssize\_t bytes\_read, bytes\_written, total\_bytes = 0;

int source\_fd = open(source, O\_RDONLY);

if (source\_fd == -1) {

perror("Ошибка открытия исходного файла");

exit(EXIT\_FAILURE);

}

int destination\_fd = open(destination, O\_WRONLY | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH);

if (destination\_fd == -1) {

perror("Ошибка открытия файла назначения");

exit(EXIT\_FAILURE);

}

while ((bytes\_read = read(source\_fd, buffer, sizeof(buffer))) > 0) {

bytes\_written = write(destination\_fd, buffer, bytes\_read);

if (bytes\_written != bytes\_read) {

perror("Ошибка записи в файл назначения");

exit(EXIT\_FAILURE);

}

total\_bytes += bytes\_written;

}

if (bytes\_read == -1) {

perror("Ошибка чтения исходного файла");

exit(EXIT\_FAILURE);

}

close(source\_fd);

close(destination\_fd);

return total\_bytes;

}

void synchronize\_directories(const char \*dir1, const char \*dir2, int max\_processes) {

DIR \*dir = opendir(dir1);

if (!dir) {

perror("Ошибка открытия каталога Dir1");

exit(EXIT\_FAILURE);

}

struct dirent \*entry;

while ((entry = readdir(dir)) != NULL) {

if (entry->d\_type == DT\_REG) { // Регулярный файл

char source\_path[MAX\_PATH\_LENGTH];

char destination\_path[MAX\_PATH\_LENGTH];

snprintf(source\_path, sizeof(source\_path), "%s/%s", dir1, entry->d\_name);

snprintf(destination\_path, sizeof(destination\_path), "%s/%s", dir2, entry->d\_name);

if (access(destination\_path, F\_OK) == -1) { // Проверка, существует ли файл в Dir2

pid\_t pid = fork();

if (pid == -1) {

perror("Ошибка при создании процесса");

exit(EXIT\_FAILURE);

} else if (pid == 0) { // Дочерний процесс

printf("PID процесса: %d, Копируется файл: %s\n", getpid(), entry->d\_name);

ssize\_t bytes\_copied = copy\_file(source\_path, destination\_path);

printf("PID процесса: %d, Файл скопирован: %s, Скопировано байт: %ld\n", getpid(), entry->d\_name, bytes\_copied);

exit(EXIT\_SUCCESS);

} else {

// Родительский процесс

while (waitpid(-1, NULL, WNOHANG) > 0);

if (max\_processes > 0) {

max\_processes--;

} else {

wait(NULL);

}

}

}

}

}

closedir(dir);

}

int main() {

char dir1[MAX\_PATH\_LENGTH], dir2[MAX\_PATH\_LENGTH];

int max\_processes;

printf("Введите имя первого каталога: ");

scanf("%s", dir1);

printf("Введите имя второго каталога: ");

scanf("%s", dir2);

printf("Введите максимальное количество одновременно работающих процессов: ");

scanf("%d", &max\_processes);

synchronize\_directories(dir1, dir2, max\_processes);

return 0;

}

Проверим работоспособность написанного кода для программы индивидуального задания (рисунок 2).

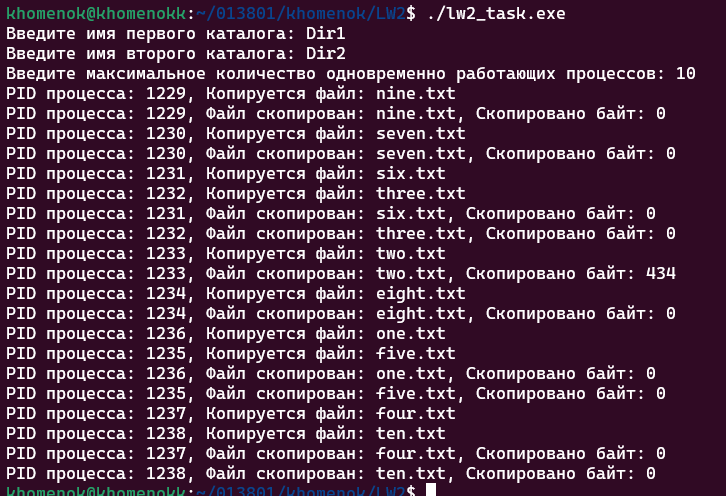


Рисунок 2 – Работоспособность программы

**4 Вывод**

В ходе выполнения лабораторной работы "Процессы в ОС Linux" была изучена теоретическая основа порождения и взаимодействия процессов в операционной системе Linux. Затем были реализованы две программы. Первая программа создает два дочерних процесса с использованием вызовов fork(), выводя на экран их PID, PID родительского процесса и текущее время. Родительский процесс выполняет команду "ps -x" с использованием system(). Вторая программа синхронизирует два каталога, копируя файлы из Dir1 в Dir2, которые отсутствуют в Dir2. Каждая процедура копирования запускается в отдельном процессе, выводя свой PID, имя файла и количество скопированных байт. Пользователь может ограничить число одновременно работающих процессов N.