

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 <time.h>
#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);

    time_t rawtime;
    struct tm *timeinfo;
    time(&rawtime);
    timeinfo = localtime(&rawtime);

    long milliseconds = tv.tv_usec / 1000;
    sprintf(buffer, "%02d:%02d:%02d:%03ld", timeinfo->tm_hour,
timeinfo->tm_min, timeinfo->tm_sec, milliseconds);
}

int main() {
    pid_t pid1, pid2;
    char time_buffer[32];

    pid1 = fork();

    if (pid1 < 0) {
        perror("Ошибка при создании первого дочернего процесса");
        exit(EXIT_FAILURE);
    } else if (pid1 == 0) {
        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 {
        pid2 = fork();

        if (pid2 < 0) {
            perror("Ошибка при создании второго дочернего
процесса");
            exit(EXIT_FAILURE);
        } else if (pid2 == 0) {
            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);
        }
    }
}

```

```

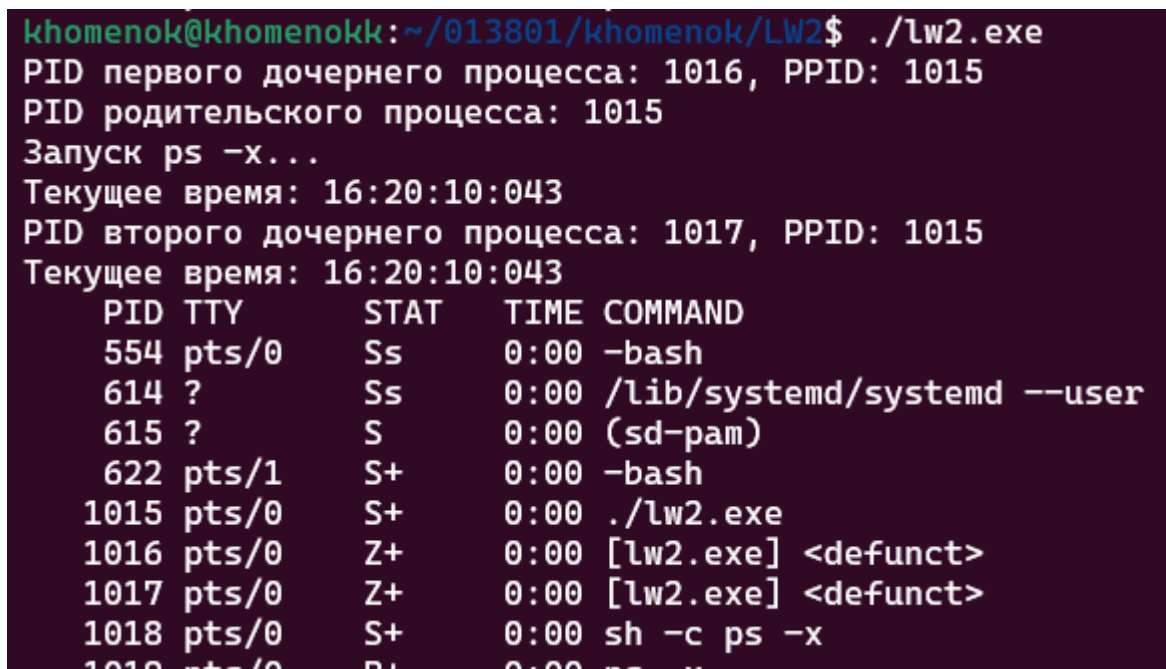
    } else {
        printf("PID родительского процесса: %d\n", getpid());
        printf("Запуск ps -x...\n");

        system("ps -x");

        wait(NULL);
        wait(NULL);
    }
}
return 0;
}

```

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



```

khomenok@khomenokk:~/013801/khomenok/LW2$ ./lw2.exe
PID первого дочернего процесса: 1016, PPID: 1015
PID родительского процесса: 1015
Запуск ps -x...
Текущее время: 16:20:10:043
PID второго дочернего процесса: 1017, PPID: 1015
Текущее время: 16:20:10:043

```

PID	TTY	STAT	TIME	COMMAND
554	pts/0	Ss	0:00	-bash
614	?	Ss	0:00	/lib/systemd/systemd --user
615	?	S	0:00	(sd-pam)
622	pts/1	S+	0:00	-bash
1015	pts/0	S+	0:00	./lw2.exe
1016	pts/0	Z+	0:00	[lw2.exe] <defunct>
1017	pts/0	Z+	0:00	[lw2.exe] <defunct>
1018	pts/0	S+	0:00	sh -c ps -x
1019	pts/0	S+	0:00	ps -x

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--;
                    }
                }
            }
        }
    }
}

```


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.