# ЛАБОРАТОРНАЯ РАБОТА №3 ПРОЦЕССЫ В ОС LINUX

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

**Теоретическая часть**

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.

Следующая короткая программа более наглядно показывает работу вызова ***fork()*** и использование процесса:

***#include <stdio.h>***

***#include <unistd.h>***

***int main ()***

***{***

***pid\_t pid; /\* идентификатор процесса \*/***

***printf (“Пока всего один процесс\n”);***

***pid = fork (); /\* Создание нового процесса \*/***

***printf (“Уже два процесса\n”);***

***if (pid = = 0){***

***printf (“Это Дочерний процесс его pid=%d\n”, getpid());***

***printf (“А pid его Родительского процесса=%d\n”, getppid());***

***}***

***else if (pid > 0)***

***printf (“Это Родительский процесс pid=%d\n”, getpid());***

***else***

***printf (“Ошибка вызова fork, потомок не создан\n”);***

***}***

Для корректного завершения дочернего процесса в родительском процессе необходимо использовать функцию ***wait()*** или ***waitpid()***:

***pid\_t wait(int \*status);   
pid\_t waitpid(pid\_t pid, int \*status, int options);***

Функция ***wait*** приостанавливает выполнение родительского процесса до тех пор, пока дочерний процесс не прекратит выполнение или до появления сигнала, который либо завершает текущий процесс, либо требует вызвать функцию-обработчик. Если дочерний процесс к моменту вызова функции уже завершился (так называемый ***«зомби»***), то функция немедленно возвращается. Системные ресурсы, связанные с дочерним процессом, освобождаются.

Функция ***waitpid ()*** приостанавливает выполнение родительского процесса до тех пор, пока дочерний процесс, указанный в параметре ***pid***, не завершит выполнение, или пока не появится сигнал, который либо завершает родительский процесс, либо требует вызвать функцию-обработчик. Если указанный дочерний процесс к моменту вызова функции уже завершился (так называемый ***«зомби»***), то функция немедленно возвращается. Системные ресурсы, связанные с дочерним процессом, освобождаются. Параметр ***pid*** может принимать несколько значений:

***pid< -1*** означает, что нужно ждать любого дочернего процесса, чей идент ификатор группы процессов равен абсолютному значению ***pid***.

***pid= -1*** означает ожидать любого дочернего процесса; функция **wait** ведет себя точно так же.

***pid = 0*** означает ожидать любого дочернего процесса, чей идентификатор группы процессов равен таковому у текущего процесса.

***pid> 0*** означает ожидать дочернего процесса, чем идентификатор равен ***pid***.

Значение ***options*** создается путем битовой операции ***ИЛИ*** над следующими константами:

**WNOHANG** - означает вернуть управление немедленно, если ни один дочерний процесс не завершил выполнение.

**WUNTRACED -** означает возвращать управление также для остановленных дочерних процессов, о чьем статусе еще не было сообщено.

Каждый дочерний процесс при завершении работы посылает своему процессу-родителю специальный сигнал ***SIGCHLD***, на который у всех процессов по умолчанию установлена реакция "игнорировать сигнал". Наличие такого сигнала совместно с системным вызовом ***waitpid()*** позволяет организовать асинхронный сбор информации о статусе завершившихся порожденных процессов процессом-родителем.

Для перегрузки исполняемой программы можно использовать функции семейства ***exec***. Основное отличие между разными функциями в семействе состоит в способе передачи параметров.

***int execl(char \*pathname, char \*arg0, arg1, ..., argn, NULL);***

***int execle(char \*pathname, char \*arg0, arg1, ..., argn, NULL, char \*\*envp);***

***int execlp(char \*pathname, char \*arg0, arg1, ..., argn, NULL);***

***int execlpe(char \*pathname, char \*arg0, arg1, ..., argn, NULL, char \*\*envp);***

***int execv(char \*pathname, char \*argv[]);***

***int execve(char \*pathname, char \*argv[],char \*\*envp);***

***int execvp(char \*pathname, char \*argv[]);***

***int execvpe(char \*pathname, char \*argv[],char \*\*envp);***

Основное отличие между разными функциями в семействе состоит в способе передачи параметров. Как видно из рис. 1, все эти функции выполняют один системный вызов ***execve****.*

**execl**

**execle**

**execve**

**execvp**

**execv**

**execlp**

Рис. 1. Дерево семейства вызовов ***exec***

**Порядок выполнения работы**

1. Изучить теоретическую часть лабораторной работы.
2. Написать программу, создающую два дочерних процесса с 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.

## Варианты индивидуальных заданий

Все исходные данные ко всем заданиям вводятся как аргументы командной строки!

1. Написать программу нахождения массива значений функции ***y[i]=sin(2\*PI\*i/N)*** ***(i=0,1,2…N-1)*** с использованием ряда Тейлора. Пользователь задаёт значения ***N*** и количество ***n*** членов ряда Тейлора. Для расчета одного члена ряда Тейлора запускается отдельный процесс. Каждый процесс выводит на экран и в файл промежуточных результатов (создать в каталоге ***/tmp)*** свой ***pid***, ***i*** и рассчитанное значение члена ряда. Головной процесс считывает из файла промежуточных результатов значения всех рассчитанных членов ряда Тейлора для каждого ***i***, суммирует их и полученное значение ***y[i]*** записывает в файл результата в виде: ***y[i] =значение***. Проверить работу программы для ***N=256 n=5; N=1024 n=10.***
2. 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,*** полный путь к копируемому файлу и число скопированных байт. Число одновременно работающих процессов не должно превышать ***N*** (вводится пользователем). Скопировать несколько файлов из каталога ***/etc*** в свой домашний каталог. Проверить работу программы для каталога ***/etc и*** домашнего каталога.
3. Написать программу поиска одинаковых по содержимому файлов в двух каталогов, например, ***Dir1*** и ***Dir2***. Пользователь задаёт имена ***Dir1*** и ***Dir2***. В результате работы программы файлы, имеющиеся в ***Dir1***, сравниваются с файлами в ***Dir2*** по их содержимому. Процедуры сравнения должны запускаться в отдельном процессе для каждой пары сравниваемых файлов. Каждый процесс выводит на экран свой ***pid***, имя файла, общее число просмотренных байт и результаты сравнения. Число одновременно работающих процессов не должно превышать ***N*** (вводится пользователем). Скопировать несколько файлов из каталога ***/etc*** в свой домашний каталог. Проверить работу программы для каталога ***/etc и*** домашнего каталога.
4. Написать программу поиска заданной пользователем комбинации из ***m*** байт (***m <255***) во всех файлах заданного каталога. Главный процесс открывает каталог и запускает для каждого файла каталога отдельный процесс поиска заданной комбинации из ***m*** байт. Каждый процесс выводит на экран свой ***pid,*** полный путь к файлу, общее число просмотренных байт и результаты (сколько раз найдена комбинация) поиска (все в одной строке). Число одновременно работающих процессов не должно превышать ***N*** (вводится пользователем). Проверить работу программы для каталога ***/etc и*** строки ***«ifconfig»***.
5. Написать программу подсчета количества слов в файлах заданного каталога его подкаталогов. Пользователь задаёт имя каталога. Главный процесс открывает каталоги и запускает для каждого файла каталога отдельный процесс подсчета количества слов. Каждый процесс выводит на экран свой ***pid,*** полный путь к файлу, общее число просмотренных байт и количество слов. Число одновременно работающих процессов не должно превышать ***N*** (вводится пользователем). Проверить работу программы для каталога ***/etc***.
6. Написать программу подсчета частоты встречающихся символов в файлах заданного каталога его подкаталогов. Пользователь задаёт имя каталога. Главный процесс открывает каталоги и запускает для каждого файла каталога отдельный процесс подсчета количества символов. Каждый процесс выводит на экран свой ***pid,*** полный путь к файлу, общее число просмотренных байт и частоты встречающихся символов (все в одной строке). Число одновременно работающих процессов не должно превышать ***N*** (вводится пользователем). Проверить работу программы для каталога ***/etc***.
7. Написать программу подсчета количества бит (0 и 1) в файлах заданного каталога и его подкаталогов. Пользователь задаёт имя каталога. Главный процесс открывает каталоги и запускает для каждого файла каталога отдельный процесс подсчета количества бит. Каждый процесс выводит на экран свой ***pid,*** полный путь к файлу, общее число просмотренных байт и количество бит 0 и 1. Число одновременно работающих процессов не должно превышать ***N*** (вводится пользователем). Проверить работу программы для каталога ***/etc***.
8. Написать программу подсчета всех периодов бит (0 и 1) в файлах заданного каталога и его подкаталогов. Пользователь задаёт имя каталога. Главный процесс открывает каталоги и запускает для каждого файла каталога отдельный процесс подсчета всех периодов бит. Каждый процесс выводит на экран свой ***pid,*** полный путь к файлу, общее число просмотренных байт и все периоды бит (0 и 1). Число одновременно работающих процессов не должно превышать ***N*** (вводится пользователем). Проверить работу программы для каталога ***/etc***.
9. Написать программу шифрации всех файлов заданного каталога и его подкаталогов. Пользователь задаёт имена начального каталога и каталога для всех зашифрованных файлов как аргумент1, 2 командной строки. Главный процесс открывает каталоги и запускает для каждого файла каталога и отдельный процесс шифрации. Каждый процесс выводит на экран свой ***pid,*** полный путь к файлу, общее число зашифрованных байт. Шифрация по алгоритму сложения по модулю 2 бит исходного файла и файла ключа. Число одновременно работающих процессов не должно превышать ***N*** (вводится пользователем как аргумент командной строки). Проверить работу программы для каталога ***/etc***.
10. Написать программу шифрации всех файлов заданного каталога и его подкаталогов. Пользователь задаёт имена начального каталога и каталога для всех зашифрованных файлов как аргумент1, 2 командной строки. Главный процесс открывает каталоги и запускает для каждого файла каталога и отдельный процесс шифрации. Каждый процесс выводит на экран свой ***pid,*** полный путь к файлу, общее число зашифрованных байт. Шифрация по алгоритму перестановки символов. Новое местоположение символа x определяется по формуле x= (текущее положение\*ключ) mod размер файла. Число одновременно работающих процессов не должно превышать ***N*** (вводится пользователем как аргумент командной строки). Проверить работу программы для каталога ***/etc***.
11. Написать программу подсчета хэша для каждого подкаталога (хэша всех файлов подкаталога). Пользователь задаёт имя начального каталога. Для хэша можно использовать любой алгоритм, дающий приемлемые результаты. Главный процесс открывает каталог запускает для каждого подкаталога отдельный процесс подсчета хэша. Каждый процесс выводит на экран свой ***pid,*** полный путь к подкаталогу, общее число просмотренных байт в подкаталоге и хэш подкаталога. Число одновременно работающих процессов не должно превышать ***N*** (вводится пользователем). Проверить работу программы для каталога ***/etc***.
12. Написать программу поиска заданного слова для начального каталога и всех подкаталогов (начальный каталог, заданное слово -аргументы 1,2 командной строки). Главный процесс открывает каталог запускает для каждого подкаталога отдельный процесс поиска заданного слова. Каждый процесс выводит на экран свой ***pid,*** полный путь к файлу, общее число найденных слов в файле. Число одновременно работающих процессов не должно превышать ***N*** (вводится пользователем). Проверить работу программы для каталога ***/etc***.