**Лабораторная работа №2.**

**Создание и управление процессов в UNIX-подобных ОС.**

Цель лабораторной работы: научиться создавать процессы и потоки, а также управлять ими

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 любой дочерний процесс, чей идентификатор группы процессов равен абсолютному значению pid.
* pid = -1 означает ожидать любой дочерний процесс; функция wait ведет себя точно так же.
* pid = 0 означает ожидать любой дочерний процесс, чей идентификатор группы процессов равен таковому у текущего процесса.
* pid > 0 означает ожидать дочерний процесс, чей 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);

**Задание**

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.
3. Выполнить индивидуальное задания.

**Варианты индивидуальных заданий:**

1. Написать программу нахождения массива K последовательных значений функции y[i]=sin(2\*PI\*i/N) (где i=0, 1, 2...K-1) с использованием ряда Тейлора. Пользователь задаёт значения K, N и количество n членов ряда Тейлора. Для расчета каждого члена ряда Тейлора запускается отдельный поток. Каждый поток выводит на экран свой pid и рассчитанное значение ряда. Головной процесс суммирует все члены ряда Тейлора, и полученное значение y[i] записывает в файл.
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, имя копируемого файла и число скопированных байт. Число одновременно работающих процессов не должно превышать N (вводится пользователем).
3. Написать программу поиска одинаковых по их 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).
4. Написать программу поиска заданной пользователем комбинации из m байт (m < 255) во всех файлах текущего каталога. Пользователь задаёт имя 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, имя файла, общее число просмотренных байт и результаты поиска. Число одновременно работающих процессов не должно превышать N (вводится пользователем).
5. Разработать программу «интерпретатор команд», которая воспринимает команды, вводимые с клавиатуры, (например, ls -l /bin/bash) и осуществляет их корректное выполнение. Для этого каждая вводимая команда должна выполняться в отдельном процессе с использованием вызова exec(). Предусмотреть контроль ошибок.
6. Создать дерево процессов по индивидуальному заданию. Каждый процесс постоянно, через время t, выводит на экран следующую информацию: номер процесса/потока, pid, ppid текущее время (мсек). Время t=(номер процесса/потока по дереву)\*200 (мсек).

**Варианты индивидуальных заданий:**

Написать программу, которая будет реализовывать следующие функции:

* сразу после запуска получает и сообщает свой ID и ID родительского процесса;
* перед каждым выводом сообщения об ID процесса и родительского процесса эта информация получается заново;
* порождает процессы, формируя генеалогическое дерево согласно варианту, сообщая, что "процесс с ID таким-то породил процесс с таким-то ID";
* перед завершением процесса сообщить, что "процесс с таким-то ID и таким- то ID родителя завершает работу";
* один из процессов должен вместо себя запустить программу, указанную в варианте задания.

На основании выходной информации программы предыдущего пункта изобразить генеалогическое дерево процессов (с указанием идентификаторов процессов). Объяснить каждое выведенное сообщение и их порядок в предыдущем пункте.

В столбце **fork** описано генеалогическое древо процессов: каждая цифра указывает на относительный номер (не путать с pid) процесса, являющегося родителем для данного процесса. Например, строка 0 1 1 1 3 означает, что первый процесс не имеет родителя среди ваших процессов (порождается и запускается извне), второй, третий и четвертый - порождены первым, пятый - третьим.

В столбце exec указан номер процесса, выполняющего вызов **exec**, команды для которого указаны в последнем столбце. Запускайте команду обязательно с какими-либо параметрами.

|  |  |  |  |
| --- | --- | --- | --- |
| № | fork | exec |  |
| 1 | 0 1 1 1 3 3 5 | 1 | ls |
| 2 | 0 1 2 2 3 4 6 | 2 | Ps |
| 3 | 0 1 1 2 2 5 6 | 3 | pwd |
| 4 | 0 1 1 1 2 5 3 | 4 | whoami |
| 5 | 0 1 1 2 2 3 3 | 5 | df |
| 6 | 0 1 1 2 4 4 4 | 6 | ls |
| 7 | 0 1 1 1 3 3 2 | 7 | Ps |
| 8 | 0 1 2 2 3 4 5 | 1 | pwd |
| 9 | 0 1 1 2 2 5 6 | 2 | whoami |
| 10 | 0 1 1 1 2 5 5 | 3 | time |
| 11 | 0 1 1 2 2 3 4 | 4 | ls |
| 12 | 0 1 1 2 4 4 5 | 5 | ps |
| 13 | 0 1 1 1 3 3 5 | 6 | pwd |
| 14 | 0 1 2 2 3 4 6 | 7 | whoami |
| 15 | 0 1 1 2 2 5 5 | 1 | date |
| 16 | 0 1 1 1 2 5 4 | 2 | Is |
| 17 | 0 1 1 2 2 3 3 | 3 | Ps |
| 18 | 0 1 1 2 4 4 6 | 4 | pwd |
| 19 | 0 1 1 2 2 5 5 | 5 | whoami |
| 20 | 0 1 1 1 2 5 4 | 6 | free |

**Контрольные вопросы**

1. Работа с процессами в языке C.
2. Использование функций семейства exec.

**Источники**

Алексеев И.Г. Учебно-методическое пособие Операционные системы и системное программирование: для студ. спец. «Программное обеспечение информационных технологий»/ И.Г Алексеев, П.Ю. Бранцевич – Мн.: БГУИР, 2009. – 73 с.