**Министерство образования Республики Беларусь**

**Учреждение образования**

**«Белорусский государственный университет**

**информатики и радиоэлектроники»**

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЕНИЯ

Кафедра интеллектуальных информационных технологий

**Отчет**

По дисциплине: Операционные системы

Лабораторная работа №2

Выполнили:

Липский Р. В.,

Жолнерчик И. А.,

гр. 121701

Проверил:

Цирук В. А.

**Минск 2022**

**Цель**:получить знания о процессах в операционной системе Linux.

**Задание:**

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.

**Задания:**

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.

#include <stdio.h>#include <unistd.h>#include <sys/wait.h>#include <sys/time.h>#include <time.h>int main() { int number = 1; printf("1. (%d -> %d)\n", getppid(), getpid()); if (fork() == 0) { number = 2; printf("2. (%d -> %d)\n", getppid(), getpid()); } else { if (fork() == 0) { number = 3; printf("3. (%d -> %d)\n", getppid(), getpid()); } } time\_t now = time(NULL); struct tm \*tm\_struct = localtime(&now); struct timeval t; gettimeofday(&t, NULL); printf("%d. time: %d:%d:%d:%ld\n", number, tm\_struct->tm\_hour, tm\_struct->tm\_min, tm\_struct->tm\_sec, t.tv\_usec/1000); while(wait(NULL) > 0); printf("STP (%d -> %d)\n", getppid(), getpid()); return 0;}

**1. Индивидуальное задание**

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).

#include <stdio.h>  
#include <unistd.h>  
#include <sys/wait.h>  
#include <sys/time.h>  
#include <time.h>  
  
int main() {  
 int number = 1;  
 printf("1. STR (%d -> %d)\n", getppid(), getpid());  
  
 if (fork() == 0) {  
 // 2nd  
 number = 2;  
 printf("2. (%d -> %d)\n", getppid(), getpid());  
 if (fork() == 0) {  
 // 4th  
 number = 4;  
 printf("4. (%d -> %d)\n", getppid(), getpid());  
  
 if (fork() == 0) {  
 // 5th  
 number = 5;  
 printf("5. (%d -> %d)\n", getppid(), getpid());  
 } else {  
 if (fork() == 0) {  
 // 6th  
 number = 6;  
 printf("6. (%d -> %d)\n", getppid(), getpid());  
 execl("/bin/ls", "/bin/ls", "-l", NULL);  
 } else {  
 if (fork() == 0) {  
 // 7th  
 number = 7;  
 printf("7. (%d -> %d)\n", getppid(), getpid());  
 }  
 }  
 }  
 }  
 } else {  
 if (fork() == 0) {  
 // 3rd  
 number = 3;  
 printf("3. (%d -> %d)\n", getppid(), getpid());  
 }  
 }  
  
 while (1) {  
 time\_t now = time(NULL);  
 struct tm \*tm\_struct = localtime(&now);  
 struct timeval t;  
 gettimeofday(&t, NULL);  
 printf("%d. time: %d:%d:%d:%ld\n", number, tm\_struct->tm\_hour, tm\_struct->tm\_min, tm\_struct->tm\_sec,  
 t.tv\_usec / 1000);  
 usleep(number \* 200000);  
 }  
  
 printf("STP (%d -> %d)\n", getppid(), getpid());  
 return 0;  
}

**2. Индивидуальное задание:**

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.

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | 0 1 1 2 4 4 4 | 6 | ls |

|  |  |  |  |
| --- | --- | --- | --- |
| 10 | 0 1 1 1 2 5 5 | 3 | time |

*2. Индивидуальное задание 6:*

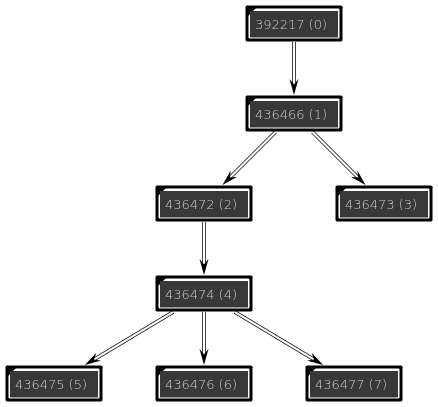
Исходный код:

#include <stdio.h>  
#include <unistd.h>  
#include <sys/wait.h>  
  
int main() {  
 // 1st  
 printf("1. STR (%d -> %d)\n", getppid(), getpid());  
  
 if (fork() == 0) {  
 // 2nd  
 printf("2. (%d -> %d)\n", getppid(), getpid());  
 if (fork() == 0) {  
 // 4th  
 printf("4. (%d -> %d)\n", getppid(), getpid());  
  
 if (fork() == 0) {  
 // 5th  
 printf("5. (%d -> %d)\n", getppid(), getpid());  
 } else {  
 if (fork() == 0) {  
 // 6th  
 printf("6. (%d -> %d)\n", getppid(), getpid());  
 execl("/bin/ls", "/bin/ls", "-l", NULL);  
 } else {  
 if (fork() == 0) {  
 // 7th  
 printf("7. (%d -> %d)\n", getppid(), getpid());  
 }  
 }  
 }  
 }  
 } else {  
 if (fork() == 0) {  
 // 3rd  
 printf("3. (%d -> %d)\n", getppid(), getpid());  
 }  
 }  
  
 while(wait(NULL) > 0); // ожидание завершения процессов-детей  
 printf("STP (%d -> %d)\n", getppid(), getpid());  
 return 0;  
}

Вывод:

1. STR (392217 -> 436466) // Запуск родительского процесса  
3. (436466 -> 436473) // Запуск 3-ого процесса  
2. (436466 -> 436472) // Запуск 2-ого процесса (после 2-ого, поскольку запуск происходит практически параллельно)  
STP (436466 -> 436473) // Остановка 3-его процесса, поскольку у него больше нет задач  
4. (436472 -> 436474) // Запуск 4-ого процесса  
5. (436474 -> 436475) // Запуск 5-ого процесса  
STP (436474 -> 436475) // Остановка 5-ого процесса, поскольку у него больше нет задач  
6. (436474 -> 436476) // Запуск 6-ого процесса  
7. (436474 -> 436477) // Запуск 7-ого процесса  
STP (436474 -> 436477) // Остановка 7-ого процесса, поскольку у него больше нет задач  
total 26 // Начало вывода 6-ого процесса  
-rw-rw-r-- 1 rostislav rostislav 26032 Oct 1 01:02 build.ninja // Вывод происходит после 7, поскольку  
-rw-rw-r-- 1 rostislav rostislav 12653 Oct 1 01:02 CMakeCache.txt // 7 процесс завершился быстрее  
drwxrwxr-x 5 rostislav rostislav 11 Oct 1 01:02 CMakeFiles // чем отработал ls -l  
-rw-rw-r-- 1 rostislav rostislav 1665 Oct 1 01:02 cmake\_install.cmake  
-rwxrwxr-x 1 rostislav rostislav 17880 Oct 1 01:46 OS\_2\_C  
drwxrwxr-x 3 rostislav rostislav 3 Oct 1 01:02 Testing // Конец вывода 6-ого процесса  
STP (436472 -> 436474) // Остановка 4-ого процесса, поскольку все его дети закончили работу  
STP (436466 -> 436472) // Остановка 2-ого процесса, поскольку все его дети закончили работу  
STP (392217 -> 436466) // Остановка 1-ого процесса, поскольку все его дети закончили работу  
// Сообщения об остановке 6 процесса не было, поскольку он был заменён на ls -l

**Генеалогическое древо процессов:**

****

*2. Индивидуальное задание 10:*

Исходный код:

#include <stdio.h>  
#include <unistd.h>  
#include <sys/wait.h>  
  
int main() {  
 // 1st  
 printf("STR (%d -> %d)\n", getppid(), getpid());  
  
 if (fork() == 0) {  
 // 2nd  
 printf("2. (%d -> %d)\n", getppid(), getpid());  
 if (fork() == 0) {  
 // 5th  
 printf("5. (%d -> %d)\n", getppid(), getpid());  
 if (fork() == 0) {  
 // 6th  
 printf("6. (%d -> %d)\n", getppid(), getpid());  
 execl("/bin/time", "/bin/time", "-p", "/bin/ls");  
 } else {  
 if (fork() == 0) {  
 // 7th  
 printf("7. (%d -> %d)\n", getppid(), getpid());  
 }  
 }  
 }  
 } else {  
 if (fork() == 0) {  
 // 3rd  
 printf("3. (%d -> %d)\n", getppid(), getpid());  
 } else {  
 if (fork() == 0) {  
 // 4th  
 printf("4. (%d -> %d)\n", getppid(), getpid());  
 }  
 }  
 }  
  
 while(wait(NULL) > 0); // ожидание завершения процессов-детей  
 printf("STP (%d -> %d)\n", getppid(), getpid());  
 return 0;  
}

Вывод:

STR (9898 -> 27442) // Запуск родительского процесса  
2. (27442 -> 27447) // Запуск 2-ого процесса  
3. (27442 -> 27448) // Запуск 3-ого процесса  
5. (27447 -> 27449) // Запуск 5-ого процесса  
4. (27442 -> 27450) // Запуск 4-ого процесса  
STP (27442 -> 27450) // Остановка 4-ого процесса, поскольку у него больше нет задач  
6. (27449 -> 27451) // Запуск 6-ого процесса  
STP (27449 -> 27451) // Остановка 6-ого процесса, поскольку у него больше нет задач  
7. (27449 -> 27452) // Запуск 7-ого процесса  
STP (27449 -> 27452) // Остановка 7-ого процесса, поскольку у него больше нет задач  
STP (27447 -> 27449) // Остановка 5-ого процесса, поскольку все его дети завершили работу  
STP (27442 -> 27447) // Остановка 2-ого процесса, поскольку все его дети завершили работу  
// Вывод команды /bin/ls (команда требует чуть больше времени на выполнение, чем наш код):   
build.ninja CMakeFiles OS\_2\_10\_C  
CMakeCache.txt cmake\_install.cmake Testing  
STP (9898 -> 27442) // Остановка родительского процесса  
// Вывод /bin/time (требует ещё чуть больше времени, чем /bin/ls):   
real 0.00  
user 0.00  
sys 0.00  
// Сообщения о завершение 3-его процесса нет, поскольку он был заменён /bin/time

Генеалогическое древо процессов:

