Лабораторная работа по курсу "Операционные системы"

Процессы в ОС Linux (I)

Цель работы: знакомство с системными вызовами для создания процессов; исследование состояния гонок при совместном доступе к файлу родительского и дочернего процессов.

Системные вызовы для управления процессами

Идентификаторы процессов. Процессы в ОС Linux имеют уникальные номера - идентификаторы процессов (PID), являющиеся целыми числами, назначаемыми процессам при их создании. В программах на C/C++, предпочтительнее использовать для PID не тип **int**, а тип **pid_t ,** описанный в файле **<sys/types.h>**. Для получения программой PID собственного процесса используется системный вызов **getpid ,** для получения PID родительского процесса - **getppid .**

Для просмотра активных процессов предназначены следующие команды:

- **ps** (*Process Status*). Введенная без опций, команда **ps** показывает только те процессы, которые были запущены из данного терминального окна.
- **top** показывает активные процессы в динамике.
- **pstree** показывает активные процессы в виде дерева.

Код завершения процесса. В нормальной ситуации процесс завершается либо системным вызовом **exit** либо возвратом из функции **main**. Код завершения - двухбайтное целое число, возвращаемое процессом своему родителю. Это аргумент функции **exit** или значение, возвращаемое функцией **main**. Старший байт кода возврата равен коду возврата программы (*пользовательский код возврата*). В младший байт ОС записывает причину завершения процесса (*системный код возврата*; при нормальном завершении процесса он равен нулю).

Создание процессов осуществляется при помощи системного вызова **fork**, который создает дочерний процесс, являющийся копией своего родительского процесса. При неудаче **fork** возвращает -1. При успешном выполнении **fork** возвращает 0 в созданный дочерний процесс и возвращает PID дочернего процесса в родительский процесс.

Задание 1.

1.1. Выполните программу **pr1.c**. *Поясните в отчете, сколько процессов будет выполнено? Сколько сообщений будет напечатано? Нарисуйте дерево процессов. Почему приглашение оболочки **$** появляется раньше, чем программа завершает работу?*

```
/* pr1.c */
#include <stdio.h>
#include <unistd.h>
 int main()
 {     fork(); printf("A\n");
       fork(); printf("B\n");
       return 0;
 }
```

1.2. В программе **pr1.c** приостановите все процессы, например, при помощи **getchar()**.
Введите команду **ps** со следующими опциями:

$ps -e -o pid,ppid,start_time,command
$ps -f -A
$ps -la **($ps -mAl)**

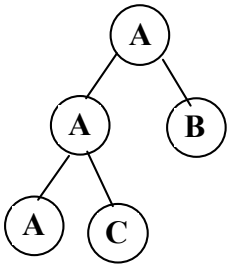
*Законспектируйте информацию об использованных Вами опциях команды **ps**. Для последней команды расшифруйте значения колонок листинга, в колонке "состояние" – обозначения для различных соятояний процесса.* Определите, какой процесс является родителем процесса **pr1**, и какой - родителем его родителя и т.д.

Введите команду **$top .**

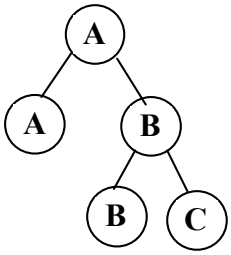
Введите команду **$pstree** и зарисуйте часть дерева процессов для **pr1 .** Рядом с процессами укажите их PID (для этого введите **pstree** с соответствующим флагом).

1.3. Модифицируйте программу **pr1.c**, удалив один **fork**. После оставшегося **fork** разделите код для родительского и дочернего процессов и выведите следующую информацию:
Для родительского процесса - *"Родительский процесс с PID=... PID дочернего процесса=..."*
Для дочернего процесса - *"Дочерний процесс с PID=... PID родительского процесса=..."*
Затем после **fork** задержите выполнение родительского процесса на 3 единицы времени (при помощи библиотечной функции **sleep**), а дочернего - на 10 единиц. Распечатайте в дочернем процессе PID его родителя дважды: до и после **sleep**. Объясните в отчете Ваши результаты.

1.4. Модифицируйте программу **pr1.c**: создайте дерево процессов в соответствии со своим вариантом (равен № ПК). Буквы внутри кружков обозначают условное имя процесса.



Нечетные варианты



Четные варианты

Переключение процесса на выполнение другой программы. Семейство функций **exec** содержит набор функций, которые переключают процесс на выполнение другой программы, т.е. изменяют образ процесса. При этом прекращается выполнение текущей программы и начинается выполнение в этом же процессе другой программы, начиная с ее точки входа. Функции семейства **exec** различаются набором аргументов:

- Функции, которые содержат в имени букву **p** (**execvp** и **execlp**), принимают имя программы и ищут программу по имени во всех путях, содержащихся в PATH; в функции, не содержащие в имени букву **p**, должно быть передано полное имя пути программы.
- Функции, содержащие в имени букву **v** (**execv**, **execvp** и **execve**), принимают список параметров для новой программы как массив указателей на строки с последним нулевым элементом. Каждая строка соответствует одному параметру. В функции, содержащие в имени букву **l** (**execl**, **execlp**, **execle**), параметры передаются как параметры соответствующей функции **exec**.
- Функции, которые содержат в имени букву **e** (**execve** и **execle**), принимают дополнительный параметр - массив переменных окружения. Этот параметр должен быть массивом указателей на строки, с последним нулевым элементом. Каждая строка должна иметь вид "ПЕРЕМЕННАЯ=значение".

Т.к. **exec** заменяет вызывающую программу на другую, то **exec** никогда не возвращает управление в вызывающую программу в случае успеха. Таким образом, код, находящийся после **exec**, будет выполнен только в случае ошибки в вызове **exec**. Список параметров, передаваемых в вызываемую программу, является аналогом параметров командной строки. Первым параметром (***argv[0]**) в нем должно быть имя программы.

Один из вариантов запуска новых программ - в основной программе создать дочерний процесс (**fork**) и в нем переключиться на другую программу (**exec**), как в задании 3:

Задание 2.

2.1. Выполните программу **pr2.c**, которая в дочернем процессе вызывает программу **ls**. Определите, какой оператор(ы) не выполняется при успешном выполнении **exec**.

```
/* pr2.c */
#include <stdio.h>
#include <unistd.h>
int main()
{char* arg_list[] = {"ls", "-l", "/", NULL};
 if (fork() == 0)
       {
       execvp("ls", arg_list);
       printf("Return after exec\n");
```

```
        }
     return 0;
    }
```

2.2. Модифицируйте программу: замените **execvp** на **execlp**:

```
     execlp("ls", "ls", "-l", "/", NULL);
```

2.3. Сделайте ошибку в вызове **exec**: а) укажите вместо "**ls**" имя несуществующей программы; б) задайте несуществующий каталог для **ls**. Различаются ли при выполнении случаи а) и б) ?

2.4. Модифицируйте программу: загрузите в дочерний процесс (используйте **execl**) любую вашу программу (например, печатающую *"Hello, world"* из лабораторной работы 2.

Совместный доступ к файлам родительского и дочернего процессов

Каждый дочерний процесс наследует от своего родительского процесса все файлы, открытые родителем до вызова **fork**. Это означает, что любой дочерний процесс может использовать соответствующие дескрипторы файлов, значения которых были установлены родительским процессом во время операции открытия файлов. При выполнении **fork** значения этих дескрипторов копируются в область данных дочернего процесса, и после этого дочерний процесс может использовать эти дескрипторы для доступа к соответствующим файлам параллельно с родительским процессом. Дочерний процесс также наследует текущий каталог родительского процесса и его права доступа к файлам.

Если родитель и его дочерний процесс имеют одновременный доступ к одним и тем же файлам, то может возникать условие ***гонок***, или ***состязаний***.

Задание 3. В данной программе **sharfile.c** родительский и дочерний процессы параллельно копируют один и тот же входной файл в один и тот же выходной файл. Добавьте в программу контроль количества входных параметров, а также функцию **perror** для каждого системного вызова **open**, **creat**, **read** и **write** (за исключением **write** на терминал).

```
/* Program sharfile.c Usage: progname source_file destination_file
 * for example: sharfile sharfile.c shar.bak */
#include <stdio.h>
#include <fcntl.h>
main (int argc,char *argv[])
{int fdrd,fdwt;
char c;
char parent = 'P';
char child ='C';
int pid;
unsigned long i;
if (argc != 3) exit (1);
if ((fdrd = open(argv[1], O_RDONLY)) == -1) exit (1);
if ((fdwt = creat(argv[2], 0666)) == -1) exit (1);
printf("Parent: creating a child process\n");
pid = fork ();
if (pid == 0)
   {
   printf("Child process starts, id = %d\n",getpid());
   for (;;)
          {
          if (read (fdrd, &c, 1) != 1) break;
          for (i=0; i<50000;i++);   /* Long cycle */
          write(1,&child,1);
          write (fdwt, &c, 1);
          }
   exit (0);
   }
```

```
else
   {
   printf("Parent starts, id= %d\n", getpid());
   for (;;)
          {
          if (read (fdrd, &c, 1) != 1) break;
          for (i=0;i<50000;i++);  /* Long cycle */
          write(1,&parent,1);
          write (fdwt, &c, 1);
          }
   wait (0);
   }
}
```

3.1. Подготовьте произвольный текстовый файл для копирования размером 500-1000 байтов. Можно взять исходный текст данной программы.

3.2. Запустите на выполнение созданный Вами исполнимый файл данной программы при помощи командной строки, содержащей два параметра: имя входного файла и имя выходного файла (имена должны быть разными). Сравните результирующий файл с исходным файлом. Совпадают ли они?

3.3. Повторите п. 3.2 несколько раз. Всегда ли результирующий файл один и тот же? Объясните причину несовпадения входного и выходного файлов. Примите во внимание, что процессы читают и записывают по одному байту каждой операцией **read** или **write**.

3.4. (Дополнительное задание) Добавьте в Вашу программу возможность измерять время (в микросекундах), необходимое для копирования файла. (Используйте функцию **gettimeofday**. Ее описание получите с помощью команды **man**.). Замерьте время копирования файла двумя процессами. Затем для сравнения исключите один процесс из операции копирования и замерьте время, необходимое для копирования файла одним процессом. Запишите в отчет Ваши результаты и выводы.

Порядок выполнения лабораторной работы

1. Выполните задания и занесите в отчет их описание со всеми требуемыми пояснениями.

Требования

1. При подготовке к лабораторной работе (дома) занесите в отчет тексты модифицированных (см. задания) и прокомментированных программ из заданий 1-3.
2. Студент должен знать ответы на следующие вопросы:

Вопросы

1. Какие команды предназначены для вывода списка активных процессов?
2. Каково назначение системных вызовов **getpid**, **getppid**, **fork**, **execxx**?
3. Если родительский процесс завершается раньше дочернего, то чему будет равно PPID дочернего процесса? Почему?
4. Может ли один процесс выполнять две программы?
5. Может ли одна программа выполняться двумя процессами?
6. Как осуществляется совместный доступ к файлу двумя процессами?
7. Каково назначение оператора **for (i=0;i<50000;i++);** в программе **sharfile.c**?

Источники информации

1. Митчел М., Оулдем Дж., Самьюэл А. Программирование для Linux. Профессиональный подход. - М.: Издательский дом "Вильямс", 2003. (Глава 3) (The original book (2001) is available at http://www.newriders.com or http://www.advancedlinuxprogramming.com)
2. Ш. Уолтон. Создание сетевых приложений в среде Linux. - М.: Издательский дом "Вильямс", 2001. (Глава 7)
