



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий
Кафедра вычислительной техники

Отчет по лабораторной работе № 5
по дисциплине
«Инструментальные средства разработки вычислительных
систем»

Тема работы:
«Работа с процессами»

Выполнил: студент группы ИВБО-02-19

К. Ю. Денисов

Принял:

И. Р. Сон

Москва 2022

Цель работы

Цель лабораторной работы изучить программные средства создания процессов, получить навыки управления и синхронизации процессов.

Задание

Написать программу в которой будет продемонстрировано создание и управление процессом-потомком.

Ход работы

Создание процессов

С целью демонстрации процесса создания процессов-потомков была написана программа на языке программирования С, исходный код которой приведен в листинге 1.

Листинг 1 — fork-demo.c

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <sys/types.h>
5  #include <sys/wait.h>
6
7
8  int main(){
9      pid_t t;
10     int a;
11     a=89;
12     int status;
13     printf("Before creating a descendant process a = %d\n",a);
14     t=fork();
15     if(t<0){
16         perror("ERROR: Child process creation failed\n");
17         return 1;
18     }
19     if(t==0){
20         printf("In the child process a = %d\n",a);
21         printf("Child PID = %d\n",getpid());
22         printf("Parrent PID = %d\n",getppid());
23         exit(1);
24     }
25     if (t>0)
```

```

26     {
        a = 10;
        printf("a = 10\n");
28     printf("In the parent process a = %d\n",a);
        printf("Parent PID =  %d\n",getpid());
30     printf("Child PID =  %d\n",t);
        wait(&status);
32     printf("wait() function argument =  0x%04x\n" , status);
    }
34     return 0;
}

```

В данной программе создается дочерний процесс, который представляет собой точную копию своего процесса-предка.

Единственное различие между ними заключается в том, что процесс-потомок в качестве возвращаемого значения системного вызова *fork()* получает 0, а процесс-предок -идентификатор процесса-потомка.

Кроме того, процесс-потомок наследует и весь контекст программной среды, включая дескрипторы файлов, каналы и т.д.

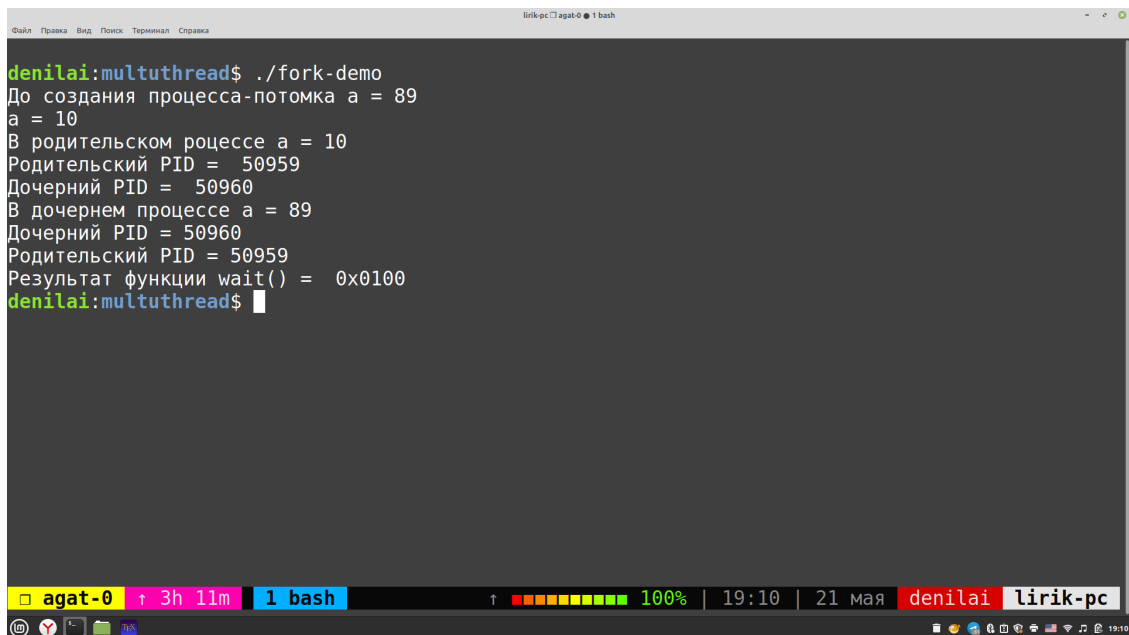
Ожидание завершения процесса-потомка родительским процессом выполняется с помощью системного вызова *wait()* `int wait(int *status)`

Аргумент системного вызова *wait()* представляет собой указатель на целочисленную переменную *status*, которая после завершения выполнения этого системного вызова будет содержать в старшем байте код завершения процесса-потомка, установленный последним в качестве системного вызова *exit()*, а в младшем - индикатор причины завершения процесса-потомка.

Чтобы продемонстрировать, что дочерний процесс наследует все переменные родительского процесса, в начале функции *main()* была объявлена переменная *a*, значение которой изменялось по ходу программы.

В данном случае дочерний процесс завершается с кодом 1, поэтому функция *wait* помещает в лидирующий байт аргумента значение 0x01.

Скомпилируем программу с помощью утилиты *gcc* с указанием исходного файла и выходного бинарного файла: `gcc fork-demo.c -o fork-demo` (см. рисунок 1).



```
denilai:multuthread$ ./fork-demo
До создания процесса-потомка a = 89
a = 10
В родительском процессе a = 10
Родительский PID = 50959
Дочерний PID = 50960
В дочернем процессе a = 89
Дочерний PID = 50960
Родительский PID = 50959
Результат функции wait() = 0x0100
denilai:multuthread$
```

Рисунок 1 — Запуск программы fork-demo

Сигналы

С целью демонстрации обработки сигналов была написана программа signals.c на языке программирования C, исходный код которой приведем в листинге 2.

Листинг 2 — signals.c

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <sys/wait.h>
5  #include <signal.h>
6
7  void sigterm_handler(int sig){
8      printf("\n=== handle SIGTERM ===\n\n");
9  }
10
11  int main (int argc, char* argv[]){
12      int pid = fork();
13      int chpid = getpid();
14      if (pid == -1) {
15          return 1;
16      }
17      if (pid == 0){
18          void sigint_handler(int sig);
19          char s[200];
20          struct sigaction sa;
```

```

22     sa.sa_handler = sigint_handler;
    sa.sa_flags = 0;
24     sigemptyset(&sa.sa_mask);
    if (sigaction(SIGTERM, &sa, NULL) == -1){
26         perror("sigaction");
        exit(1);
28     }

    while (1) {
30         printf("We are in the child process with PID = %d\n", getpid());
        sleep(1);
32     }
    return 0;
34 }
else {
36     while (1){
38         printf("Send SIGKILL signal to child process with PID = %d every 4
seconds\n", pid);
        sleep(3);
40         kill(pid, SIGTERM);
        }
42     wait(NULL);
    }
44 }

```

В данной программе создается дочерний процесс, которому каждые 4 секунды отправляется сигнал SIGTERM. Дочерний процесс обрабатывает данный сигнал с помощью обработчика. Структура sigaction устанавливает обработчик и специальные флаги и маски.

В результате при отправлении сигнала SIGTERM, дочерний процесс вызывает функцию sig_handler().

Скомпилируем программу с помощью утилиты gcc с указанием исходного файла и выходного бинарного файла: gcc signals.c -o signals (см. рисунок 2).

