



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н. Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н. Э. Баумана)

---

ФАКУЛЬТЕТ

«Информатика и системы управления»

КАФЕДРА

«Программное обеспечение ЭВМ и информационные технологии»

## ОТЧЕТ

По лабораторной работе №4

По курсу: «Операционные системы»

Тема: «Процессы. Системные вызовы fork() и exec()»

Студент:

Пронин А. С.

Группа:

ИУ7-52Б

Преподаватель:

Рязанова Н. Ю.

Оценка:

\_\_\_\_\_

Москва

2021

Листинг 1: Программа 1

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4
5 int main()
6 {
7     int childpid_1, childpid_2;
8
9     if ((childpid_1 = fork()) == -1)
10    {
11        perror("Can't fork.\n");
12        return EXIT_FAILURE;
13    }
14    else if (childpid_1 == 0)
15    {
16        sleep(2);
17        printf("\nFirst child: pid=%d; ppid=%d; pgrep=%d\n", getpid(),
18               getppid(), getpgrp());
19        exit(EXIT_SUCCESS);
20    }
21
22    if ((childpid_2 = fork()) == -1)
23    {
24        perror("Can't fork.\n");
25        return EXIT_FAILURE;
26    }
27    else if (childpid_2 == 0)
28    {
29        sleep(3);
30        printf("Second child: pid=%d; ppid=%d; pgrep=%d\n", getpid(), getppid(),
31               getpgrp());
32        exit(EXIT_SUCCESS);
33    }
34
35    printf("Parent: pid=%d; pgrep=%d; child1=%d; child2=%d\n", getpid(),
36           getpgrp(), childpid_1, childpid_2);
37
38    printf("Parent will die now.\n");
39    return EXIT_SUCCESS;
40 }
```

```

arseny@arseny-VirtualBox:~/shared/Lab4$ gcc 1.c
arseny@arseny-VirtualBox:~/shared/Lab4$ ./a.out
Parent: pid = 7554; pgrp = 7554; child1 = 7555; child2 = 7556
Parent will die now.
arseny@arseny-VirtualBox:~/shared/Lab4$ 
First child: pid = 7555; ppid = 705;  pgrp = 7554
Second child: pid = 7556; ppid = 705;  pgrp = 7554

```

Рис. 1: Результат работы программы 1

arseny@arseny-VirtualBox:~/shared/Lab4\$ ps -al													
F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
4	S	1000	787	782	0	80	0	-	146832	ep_pol	tty2	00:00:45	Xorg
0	S	1000	916	782	0	80	0	-	49908	poll_s	tty2	00:00:00	gnome-session-b
0	S	1000	7554	7369	0	80	0	-	624	hrtime	pts/0	00:00:00	a.out
1	S	1000	7555	7554	0	80	0	-	591	hrtime	pts/0	00:00:00	a.out
1	S	1000	7556	7554	0	80	0	-	591	hrtime	pts/0	00:00:00	a.out
0	R	1000	7557	7487	0	80	0	-	5021	-	pts/1	00:00:00	ps

arseny@arseny-VirtualBox:~/shared/Lab4\$ ps -al													
F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
4	S	1000	787	782	0	80	0	-	146812	ep_pol	tty2	00:00:45	Xorg
0	S	1000	916	782	0	80	0	-	49908	poll_s	tty2	00:00:00	gnome-session-b
1	S	1000	7555	705	0	80	0	-	591	hrtime	pts/0	00:00:00	a.out
1	S	1000	7556	705	0	80	0	-	591	hrtime	pts/0	00:00:00	a.out
0	R	1000	7558	7487	0	80	0	-	5021	-	pts/1	00:00:00	ps

Рис. 2: Демонстрация "усыновления"

## Листинг 2: Программа 2

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6
7 void checkStatus(int child_pid, int status);
8
9 int main()
10 {
11     int childpid_1, childpid_2;
12
13     if ((childpid_1 = fork()) == -1)
14     {
15         perror("Can't fork.\n");
16         return EXIT_FAILURE;
17     }
18     else if (childpid_1 == 0)
19     {
20         //sleep(1);
21         printf("First child: pid=%d; ppid=%d; pgrep=%d\n", getpid(), getppid(),
22               getpgrp());
23         exit(EXIT_SUCCESS);
24     }
25
26     if ((childpid_2 = fork()) == -1)
27     {
28         perror("Can't fork.\n");
29         return EXIT_FAILURE;
30     }
31     else if (childpid_2 == 0)
32     {
33         //sleep(2);
34         printf("Second child: pid=%d; ppid=%d; pgrep=%d\n", getpid(), getppid(),
35               getpgrp());
36         exit(EXIT_SUCCESS);
37     }
38
39     printf("Parent: pid=%d; pgrep=%d; child1=%d; child2=%d\n", getpid(),
40           getpgrp(), childpid_1, childpid_2);
41
42     int status;
43     pid_t child_pid;
44
45     printf("Waiting...\n");
46     child_pid = wait(&status);
47     checkStatus(child_pid, status);
```

```

45     printf("Waiting... \n");
46     child_pid = wait(&status);
47     checkStatus(child_pid, status);
48
49
50     printf("Parent will die now. \n");
51     return EXIT_SUCCESS;
52 }
53
54 void checkStatus(int child_pid, int status)
55 {
56     if (WIFEXITED(status))
57         printf("Child with pid = %d has terminated normally. \n\n", child_pid);
58     else if (WEXITSTATUS(status))
59         printf("Child with pid = %d has terminated with code %d. \n", child_pid,
60                WIFEXITED(status));
61     else if (WIFSIGNALED(status))
62     {
63         printf("Child with pid = %d has terminated with an un-intercepted signal. \n",
64                child_pid);
65         printf("Signal number = %d. \n", WTERMSIG(status));
66     }
67     else if (WIFSTOPPED(status))
68     {
69         printf("Child with pid = %d has stopped. \n", child_pid);
70         printf("Signal number = %d. ", WSTOPSIG(status));
71     }
72 }
```

```

arseny@arseny-VirtualBox:~/shared/Lab4$ gcc 2wait.c
arseny@arseny-VirtualBox:~/shared/Lab4$ ./a.out
Parent: pid = 7615; pgrp = 7615; child1 = 7616; child2 = 7617
Waiting...
Second child: pid = 7617; ppid = 7615; pgrp = 7615
Child with pid = 7617 has terminated normally.

Waiting...
First child: pid = 7616; ppid = 7615; pgrp = 7615
Child with pid = 7616 has terminated normally.

Parent will die now.
```

Рис. 3: Результат работы программы 2

### Листинг 3: Программа 3

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6
7 void checkStatus(int child_pid, int status);
8
9 int main()
10 {
11     int childpid_1, childpid_2;
12
13     if ((childpid_1 = fork()) == -1)
14     {
15         perror("Can't fork.\n");
16         return EXIT_FAILURE;
17     }
18     else if (childpid_1 == 0)
19     {
20         printf("First child: pid=%d; ppid=%d; pgid=%d\n", getpid(), getppid(),
21               getpgrp());
22         if (execlp("ls", "ls", NULL) == -1)
23         {
24             perror("First child can't exec");
25             exit(EXIT_FAILURE);
26         }
27         exit(EXIT_SUCCESS);
28     }
29     if ((childpid_2 = fork()) == -1)
30     {
31         perror("Can't fork.\n");
32         return EXIT_FAILURE;
33     }
34     else if (childpid_2 == 0)
35     {
36         printf("Second child: pid=%d; ppid=%d; pgid=%d\n", getpid(), getppid(),
37               getpgrp());
38         if (execl("sort", "sort", "999", "111", "9", "1", "11", "99", "55",
39                   "555", NULL) == -1)
40         {
41             perror("Second child can't exec");
42             exit(EXIT_FAILURE);
43         }
44         exit(EXIT_SUCCESS);
45     }
```

```

45     printf("Parent: pid=%d; pgrp=%d; child1=%d; child2=%d\n", getpid(),
46             getpgrp(), childpid_1, childpid_2);
47
48     int status;
49     pid_t child_pid;
50
51     printf("Waiting...\n");
52     child_pid = wait(&status);
53     checkStatus(child_pid, status);
54
55     printf("Waiting...\n");
56     child_pid = wait(&status);
57     checkStatus(child_pid, status);
58
59     printf("Parent will die now.\n");
60     return EXIT_SUCCESS;
61 }
62 void checkStatus(int child_pid, int status)
63 {
64     if (WIFEXITED(status))
65         printf("Child with pid=%d has terminated normally.\n\n", child_pid);
66     else if (WEXITSTATUS(status))
67         printf("Child with pid=%d has terminated with code %d.\n", child_pid,
68                WIFEXITED(status));
69     else if (WIFSIGNALED(status))
70     {
71         printf("Child with pid=%d has terminated with an un-intercepted signal.\n",
72                child_pid);
73         printf("Signal number=%d.\n", WTERMSIG(status));
74     }
75     else if (WIFSTOPPED(status))
76     {
77         printf("Child with pid=%d has stopped.\n", child_pid);
78         printf("Signal number=%d.", WSTOPSIG(status));
79     }
80 }
```

Листинг 4: Программа sort для потомка

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void printMas(const int *mas, int size);
5 void selectionSort(int *l, int *r);
6
7 int main(int argc, char *argv[])
8 {
9     int n = argc - 1;
10    int mas[n];
11
12    for (int i = 0; i < n; i++)
13        mas[i] = atoi(argv[i + 1]);
14
15    printf("Inputed array = ");
16    printMas(mas, n);
17
18    selectionSort(&mas[0], &mas[n-1]);
19    printf("Sorted array = ");
20    printMas(mas, n);
21
22    return 0;
23 }
24
25 void printMas(const int *mas, int size)
26 {
27     for (int i = 0; i < size; i++)
28     {
29         printf("%d ", mas[i]);
30     }
31     printf("\n");
32 }
33
34 void swap(int *el1, int *el2)
35 {
36     int temp = *el1;
37     *el1 = *el2;
38     *el2 = temp;
39 }
40
41
42
43
44
45
46
47
```

```

48 void selectionSort(int *l, int *r)
49 {
50     for (int *i = l; i <= r; i++)
51     {
52         int minz = *i, *ind = i;
53         for (int *j = i + 1; j <= r; j++)
54         {
55             if (*j < minz)
56             {
57                 minz = *j;
58                 ind = j;
59             }
60         }
61         swap(i, ind);
62     }
63 }
```

```

arseny@arseny-VirtualBox:~/shared/Lab4$ gcc 3.c
arseny@arseny-VirtualBox:~/shared/Lab4$ ./a.out
Parent: pid = 7628; pgrp = 7628; child1 = 7629; child2 = 7630
Waiting...
Second child: pid = 7630; ppid = 7628; pgrp = 7628
First child: pid = 7629; ppid = 7628; pgrp = 7628
Inputed array = 999 111 9 1 11 99 55 555 5
Sorted array = 1 5 9 11 55 99 111 555 999
Child with pid = 7630 has terminated normally.

Waiting...
1.c 2wait.c 3.c 4.c 5.c a.out SelectionSort.c sort
Child with pid = 7629 has terminated normally.

Parent will die now.
```

Рис. 4: Результат работы программы 3

### Листинг 5: Программа 4

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #include <sys/types.h>
6 #include <sys/wait.h>
7
8 #define LEN 50
9 #define TEXT1 "My name is Professional\n"
10#define TEXT2 "There is no meaning in this words\n"
11
12 void checkStatus(int child_pid, int status);
13
14 int main()
15 {
16     int childpid_1, childpid_2;
17     int fd[2];
18
19     if (pipe(fd) == -1)
20     {
21         perror("Can't pipe.\n");
22         return EXIT_FAILURE;
23     }
24
25     if ((childpid_1 = fork()) == -1)
26     {
27         perror("Can't fork.\n");
28         return EXIT_FAILURE;
29     }
30     else if (childpid_1 == 0)
31     {
32         close(fd[0]);
33         write(fd[1], TEXT1, strlen(TEXT1) + 1);
34         exit(EXIT_SUCCESS);
35     }
36
37     if ((childpid_2 = fork()) == -1)
38     {
39         perror("Can't fork.\n");
40         return EXIT_FAILURE;
41     }
42     else if (childpid_2 == 0)
43     {
44         close(fd[0]);
45         write(fd[1], TEXT2, strlen(TEXT2) + 1);
46         exit(EXIT_SUCCESS);
47     }
}
```

```

48
49     printf("Parent: pid=%d; pggrp=%d; child1=%d; child2=%d\n", getpid(),
50            getpgrp(), childpid_1, childpid_2);
51
52     char text1[LEN], text2[LEN];
53
54     close(fd[1]);
55     read(fd[0], text1, LEN);
56     read(fd[0], text2, LEN);
57
58     printf("Text1: %s", text1);
59     printf("Text2: %s", text2);
60
61     int status;
62     pid_t child_pid;
63
64     printf("Waiting...\n");
65     child_pid = wait(&status);
66     checkStatus(child_pid, status);
67
68     printf("Waiting...\n");
69     child_pid = wait(&status);
70     checkStatus(child_pid, status);
71
72     printf("Parent will die now.\n");
73     return EXIT_SUCCESS;
74 }
75
76 void checkStatus(int child_pid, int status)
77 {
78     if (WIFEXITED(status))
79         printf("Child with pid=%d has terminated normally.\n\n", child_pid);
80     else if (WEXITSTATUS(status))
81         printf("Child with pid=%d has terminated with code %d.\n", child_pid,
82                WIFEXITED(status));
83     else if (WIFSIGNALED(status))
84     {
85         printf("Child with pid=%d has terminated with an un-intercepted signal.\n",
86                child_pid);
87         printf("Signal number=%d.\n", WTERMSIG(status));
88     }
89     else if (WIFSTOPPED(status))
90     {
91         printf("Child with pid=%d has stopped.\n", child_pid);
92         printf("Signal number=%d.", WSTOPSIG(status));
93     }
94 }
```

```
arseny@arseny-VirtualBox:~/shared/Lab4$ gcc 4.c
arseny@arseny-VirtualBox:~/shared/Lab4$ ./a.out
Parent: pid = 7641; pgrp = 7641; child1 = 7642; child2 = 7643
Text1: There is no meaning in this words
Text2: My name is Professional
Waiting...
Child with pid = 7643 has terminated normally.

Waiting...
Child with pid = 7642 has terminated normally.

Parent will die now.
```

Рис. 5: Результат работы программы 4

Листинг 6: Программа 5

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #include <signal.h>
6 #include <sys/types.h>
7 #include <sys/wait.h>
8
9 #define LEN 50
10#define TEXT1 "My name is Professional\n"
11#define TEXT2 "There is no meaning in this words\n"
12
13 void checkStatus(int child_pid, int status);
14 void catch_sig(int sig_numb);
15
16 int flag = 0;
17
18 int main()
19 {
20     signal(SIGINT, catch_sig);
21
22     int childpid_1, childpid_2;
23     int fd[2];
24
25     if (pipe(fd) == -1)
26     {
27         perror("Can't pipe.\n");
28         return EXIT_FAILURE;
29     }
30
31     if ((childpid_1 = fork()) == -1)
32     {
33         perror("Can't fork.\n");
34         return EXIT_FAILURE;
35     }
36     else if (childpid_1 == 0)
37     {
38         close(fd[0]);
39         write(fd[1], TEXT1, strlen(TEXT1) + 1);
40         exit(EXIT_SUCCESS);
41     }
42
43     if ((childpid_2 = fork()) == -1)
44     {
45         perror("Can't fork.\n");
46         return EXIT_FAILURE;
47     }
```

```

48     else if (childpid_2 == 0)
49     {
50         close(fd[0]);
51         write(fd[1], TEXT2, strlen(TEXT2) + 1);
52         exit(EXIT_SUCCESS);
53     }
54
55     printf("Parent: pid=%d; pgrp=%d; child1=%d; child2=%d\n", getpid(),
56           getpgrp(), childpid_1, childpid_2);
57     printf("Press \"CTRL+C\", to see message from second child.\n");
58     printf("In other case you will see message from first child.\n\n");
59
60     char text1[LEN], text2[LEN];
61
62     close(fd[1]);
63     read(fd[0], text1, LEN);
64     sleep(2);
65
66     if (flag)
67         printf("Message: %s", text2);
68     else
69         printf("Message: %s", text1);
70
71     int status;
72     pid_t child_pid;
73
74     printf("Waiting...\n");
75     child_pid = wait(&status);
76     checkStatus(child_pid, status);
77
78     printf("Waiting...\n");
79     child_pid = wait(&status);
80     checkStatus(child_pid, status);
81
82     printf("Parent will die now.\n");
83     return EXIT_SUCCESS;
84 }
85
86 void checkStatus(int child_pid, int status)
87 {
88     if (WIFEXITED(status))
89         printf("Child with pid=%d has terminated normally.\n\n", child_pid);
90     else if (WEXITSTATUS(status))
91         printf("Child with pid=%d has terminated with code %d.\n", child_pid,
92               WIFEXITED(status));
93 }
```

```

94     else if (WIFSIGNALED(status))
95     {
96         printf("Child with pid = %d has terminated with an un-intercepted signal.\n",
97               child_pid);
98         printf("Signal number = %d.\n", WTERMSIG(status));
99     }
100    else if (WIFSTOPPED(status))
101    {
102        printf("Child with pid = %d has stopped.\n", child_pid);
103        printf("Signal number = %d.", WSTOPSIG(status));
104    }
105 }
106 void catch_sig(int sig_numb)
107 {
108     flag = 1;
109     printf("\ncatch_sig: %d\n", sig_numb);
110 }
```

```

arseny@arseny-VirtualBox:~/shared/Lab4$ gcc 5.c
arseny@arseny-VirtualBox:~/shared/Lab4$ ./a.out
Parent: pid = 7654; pgrp = 7654; child1 = 7655; child2 = 7656
Press "CTRL+C", to see message from second child.
In other case you will see message from first child.

Message: There is no meaning in this words
Waiting...
Child with pid = 7655 has terminated normally.

Waiting...
Child with pid = 7656 has terminated normally.

Parent will die now.
```

```

arseny@arseny-VirtualBox:~/shared/Lab4$ ./a.out
Parent: pid = 7692; pgrp = 7692; child1 = 7693; child2 = 7694
Press "CTRL+C", to see message from second child.
In other case you will see message from first child.

^C
catch_sig: 2
Message: My name is Proffesional
Waiting...
Child with pid = 7693 has terminated normally.

Waiting...
Child with pid = 7694 has terminated normally.

Parent will die now.
```

Рис. 6: Результат работы программы 5

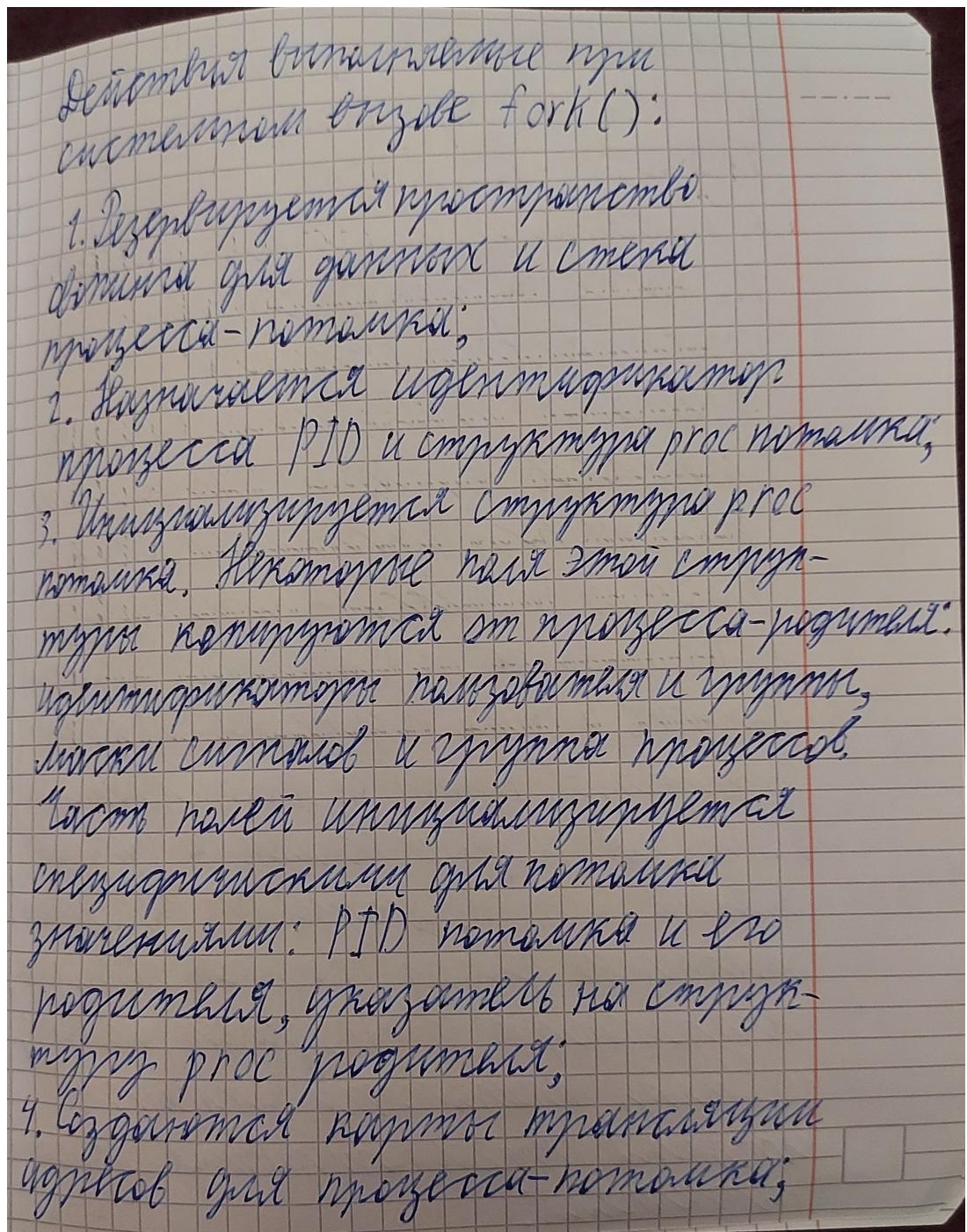


Рис. 7: Действия, выполняемые при системном вызове fork(), стр. 1

5. Выделяется область и память и в ней копируется область и процесса-предка;
6. Изменяются ссылки областей на новые параметры процесса и местоположение стеков;
7. Потомок получает в магнитном поле, которое разделяет сист. память между программой, системой процессор-регистерами;
8. Построение структуры областей и открытие регистрационных номеров;
9. Потомок получает ссылки на различные ресурсы, которые он может вызвать: открытые файлы (потомок наследует дескрипторы) и механизмы распределения.
10. Иниц. запрограмм. компьютером помощьи путём копирования регистрационных номеров.

Рис. 8: Действия, выполняемые при системном вызове fork(), стр. 2

11. Родительский процесс-потомок в очередь готовых процессов;
12. Возвращается РПД в память извлечена из системного вызова в родительский процесс и О- в процесс-потомок.

Рис. 9: Действия, выполняемые при системном вызове fork(), стр. 3

11. Проверяет процесс-помощник в  
очереди готовых процессов;  
12. Выделяется PID в память  
возвращат из системы вызова  
в родительском процессе и о-  
бъект процесс-помощник.

Действия Unix при исп. вызове exec():

1. Разбирает путь к исполняемому  
файлу и получает доступ к нему;
2. Проверяет, имеет ли вызываю-  
щий процесс наследуемую на exec-e  
правильные права;
3. Читает заголовок и проверяет, что  
он соответствует исполняемому;
4. Если для файла установлены биты  
SUID или SGID, то эффективные иденти-  
тификаторы вызывающего процесса  
изменяют на UID и GID, софт-e  
благодаря функции;

Рис. 10: Действия, выполняемые при системном вызове exec(), стр. 1

5. Копируем атрибуты, передаваемые в exec, а также передаваемые среди прост-бо ядер, после чего текущий процесс прос-бо готов к управлению;
6. Ведем прост-бо сканирование файлов для данных и структур;
7. Высвобождаем старое деср. прост-бо и связываем с новым прост-бо сканирования. Если же процесс был создан при помощи vfork, производится возврат старого деср. прост-бо родительскому процессу;
8. Ведем поиск новых пространств адресов для нового текста, данных и структур;

Рис. 11: Действия, выполняемые при системном вызове exec(), стр. 2

9. Установливаем новое ядро ядро-80.  
Если для текста скомпилированной мо-  
дели процесс уже запущен под  
же программу), то она будет  
совместно использоваться с этой  
программой. В других случаях  
ядро-80 должно инициализир-тся  
из штатного. Процесс в сист.  
UNIX обычно разбивается на структуры,  
которые называют стр.  
Составляется в память таблица по адресу  
исходных текстов.

10. Копирует фрагменты и перенес-  
ные среди обратного в новую структуру.

11. Сорганичивает все обратимыи ин-  
струкции в действия, определяющие то  
уничтожение, так как другие под-св  
ит-ов не существует в новой программе.  
Бинарные, которые были созданы ими  
заранее перед выполнением exec, остаются в памяти сист..

Рис. 12: Действия, выполняемые при системном вызове exec(), стр. 3

12. Иници-ем аппаратный комплекс.  
При этом большинство регистров  
составляется 0, а ук-адр  
команд получает значения  
также всегда программы.

Рис. 13: Действия, выполняемые при системном вызове exec(), стр. 4