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»

Лабораторная работа № 4

Тема Оптимизация fork

Студент Садулаева Т.Р.

Группа ИУ7-54Б

Оценка (баллы) _____

Преподаватель Рязанова Наталья Юрьевна

Москва, 2020 г.

Задание 1:

Написать программу, запускающую не мене двух новых процессов системным вызовом fork(). В предке вывести собственный идентификатор (функция getpid()), идентификатор группы (функция getpgrp()) и идентификаторы потомков. В процессе-потомке вывести собственный идентификатор, идентификатор предка (функция getppid()) и идентификатор группы. Убедиться, что при завершении процесса-предка потомок, который продолжает выполняться, получает идентификатор предка (PPID), равный 1 или идентификатор процесса-посредника.

```
include <stdio.h>
int main()

{

    int childpid_1;
    int childpid_2;

    childpid_1 = fork();

    if (childpid_1 == -1)

    {

        perror("Can't fork.\n");

        return 1;

    }
    if (childpid_1 > 0)
    {
        childpid_2 = fork();
        if (childpid_2 == -1)

        {

            perror("Can't fork.\n");
            return 1;

        }

    }
    if (childpid_1 == 0)

    {
        sleep(1);
        printf("\nПроцесс-потомок_1:\n");
        printf("Собственный идентификатор -- %d\n", getpid());
        printf("Идентификатор предка -- %d\n",getppid());
```

```
        printf("Идентификатор группы -- %d\n",getpgrp());
        return 0;

    }

    if (childpid_2 == 0)

    {
        sleep(2);

        printf("\nПроцесс-потомок_2:\n");
        printf("Собственный идентификатор -- %d\n", getpid());
        printf("Идентификатор предка -- %d\n",getppid());
        printf("Идентификатор группы -- %d\n",getpgrp());
        return 0;

    }
    printf("Процесс-предок:\n");
    printf("Собственный идентификатор -- %d\n", getpid());
    printf("Идентификатор группы -- %d\n",getpgrp());
    printf("Идентификатор потомка #1 -- %d\n",childpid_1);
    printf("Идентификатор потомка #2 -- %d\n",childpid_2);
    return 0;
}
```

```
Процесс-предок:
Собственный идентификатор -- 2435
Идентификатор группы -- 2435
Идентификатор потомка #1 -- 2436
Идентификатор потомка #2 -- 2437

Процесс-потомок_1:
Собственный идентификатор -- 2436
Идентификатор предка -- 1250
Идентификатор группы -- 2435

Процесс-потомок_2:
Собственный идентификатор -- 2437
Идентификатор предка -- 1250
Идентификатор группы -- 2435
```

Задание 2:

Написать программу по схеме первого задания, но в процессе-предке выполнить системный вызов wait(). Убедиться, что в этом случае идентификатор процесса потомка на 1 больше идентификатора процесса-предка.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>

int main()

{

	int childpid_1;
	int childpid_2;

	childpid_1 = fork();

	if (childpid_1 == -1)

	{

		perror("Can’t fork.\n");

		return 1;

	}
	if (childpid_1 > 0)
	{
		childpid_2 = fork();
		if (childpid_2 == -1)

		{

			perror("Can’t fork.\n");
			return 1;

		}
	}

	if (childpid_1 == 0)

	{
		sleep(1);
		printf("\nПроцесс-потомок_1:\n");
		printf("Собственный идентификатор -- %d\n", getpid());
		printf("Идентификатор предка -- %d\n",getppid());
		printf("Идентификатор группы -- %d\n",getpgrp());
		return 0;

	}
```

```
	if (childpid_2 == 0)

	{
		sleep(2);
		printf("\nПроцесс-потомок_2:\n");
		printf("Собственный идентификатор -- %d\n", getpid());
		printf("Идентификатор предка -- %d\n",getppid());
		printf("Идентификатор группы -- %d\n",getpgrp());
		return 0;

	}


	printf("Процесс-предок:\n");
	printf("Собственный идентификатор -- %d\n", getpid());
	printf("Идентификатор группы -- %d\n",getpgrp());
	printf("Идентификатор потомка #1 -- %d\n",childpid_1);
	printf("Идентификатор потомка #2 -- %d\n",childpid_2);

	int count = 0;
	while (count < 2)
	{
		int status;
		pid_t child_pid = wait(&status);
		printf("Child has finished: PID = %d\n", child_pid);
		if (WIFEXITED(status))
			printf("Child exited with code #%d\n", WEXITSTATUS(status));
		else if (WIFSIGNALED(status))
			printf("Child exited with non-intercepted signal #%d\n",
WTERMSIG(status));
		else if (WIFSTOPPED(status))
			printf("Child stopped, signal #%d\n", WSTOPSIG(status));
		count++;
	}
	return 0;
}
```

```
Процесс-предок:
Собственный идентификатор -- 4858
Идентификатор группы -- 4858
Идентификатор потомка #1 -- 4859
Идентификатор потомка #2 -- 4860

Процесс-потомок_1:
Собственный идентификатор -- 4859
Идентификатор предка -- 4858
Идентификатор группы -- 4858
Child has finished: PID = 4859
Child exited with code #0

Процесс-потомок_2:
Собственный идентификатор -- 4860
Идентификатор предка -- 4858
Идентификатор группы -- 4858
Child has finished: PID = 4860
Child exited with code #0
```

Задание 3:

Написать программу, в которой процесс-потомок вызывает системный вызов exec(), а процесс-предок ждет завершения процесса-потомка. Следует создать не менее двух потомков.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>

int main()

{

	int childpid_1;
	int childpid_2;

	childpid_1 = fork();

	if (childpid_1 == -1)

	{

		perror("Can’t fork.\n");

		return 1;

	}
	if (childpid_1 > 0)
	{
		childpid_2 = fork();
		if (childpid_2 == -1)

		{

			perror("Can’t fork.\n");
			return 1;

		}
	}

	if (childpid_1 == 0)

	{
		sleep(1);
		printf("\nПроцесс-потомок_1:\n");
		printf("Собственный идентификатор -- %d\n", getpid());
		printf("Идентификатор предка -- %d\n",getppid());
		printf("Идентификатор группы -- %d\n",getpgrp());
		printf("\n");
		if (execlp("ps","ps","-al", 0) == -1)
		{
			printf("Вызов exec отклонен");
			return 2;
		}
		return 0;
```

```
	}

	if (childpid_2 == 0)

	{
		sleep(2);
		printf("\nПроцесс-потомок_2:\n");
		printf("Собственный идентификатор -- %d\n", getpid());
		printf("Идентификатор предка -- %d\n",getppid());
		printf("Идентификатор группы -- %d\n",getpgrp());
		printf("\n");
		if (execlp("ls","ls","-a", 0) == -1)
		{
			printf("Вызов exec отклонен");
			return 2;
		}
		return 0;

	}


	printf("Процесс-предок:\n");
	printf("Собственный идентификатор -- %d\n", getpid());
	printf("Идентификатор группы -- %d\n",getpgrp());
	printf("Идентификатор потомка #1 -- %d\n",childpid_1);
	printf("Идентификатор потомка #2 -- %d\n",childpid_2);

	int count = 0;
	while (count < 2)
	{
		int status;
		pid_t child_pid = wait(&status);
		printf("Child has finished: PID = %d\n", child_pid);
		if (WIFEXITED(status))
			printf("Child exited with code #%d\n", WEXITSTATUS(status));
		else if (WIFSIGNALED(status))
			printf("Child exited with non-intercepted signal #%d\n",
WTERMSIG(status));
		else if (WIFSTOPPED(status))
			printf("Child stopped, signal #%d\n", WSTOPSIG(status));
		count++;
	}
	return 0;
}
```

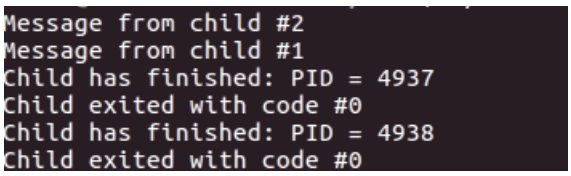
```
Процесс-предок:
Собственный идентификатор -- 4906
Идентификатор группы -- 4906
Идентификатор потомка #1 -- 4907
Идентификатор потомка #2 -- 4908

Процесс-потомок_1:
Собственный идентификатор -- 4907
Идентификатор предка -- 4906
Идентификатор группы -- 4906

F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  4906  4839  0  80   0 -  1088 wait   pts/2    00:00:00 a.out
0 R  1000  4907  4906  0  80   0 -  9000 -      pts/2    00:00:00 ps
1 S  1000  4908  4906  0  80   0 -  1055 hrtime pts/2    00:00:00 a.out
Child has finished: PID = 4907
Child exited with code #0

Процесс-потомок_2:
Собственный идентификатор -- 4908
Идентификатор предка -- 4906
Идентификатор группы -- 4906

.  ..  a2_1.png  a2_2  a2.c  a3.c  a4.c  a5.c  a.c  a.out  second.c
Child has finished: PID = 4908
Child exited with code #0
```

Задание 4:

Написать программу, в которой предок и потомок обмениваются сообщением через программный канал.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#define N 25
#define D 2

int main()

{

	int childpid_1;
	int childpid_2;

	char msg1[N] = "Message from child #1";
	char msg2[N] = "Message from child #2";
	int fd[D];

	pipe(fd);

	childpid_1 = fork();

	if (childpid_1 == -1)

	{

		perror("Can’t fork.\n");

		return 1;

	}
```

```
	if (childpid_1 > 0)
	{
		childpid_2 = fork();
		if (childpid_2 == -1)

		{

			perror("Can’t fork.\n");
			return 1;

		}
	}

	if (childpid_1 == 0)

	{
		close(fd[0]);
		write(fd[1], msg1, N);
		return 0;

	}

	if (childpid_2 == 0)

	{
		close(fd[0]);
		write(fd[1], msg2, N);
		return 0;

	}

	char msg[N];
	for(int i =0; i < 2; i++)
	{
		close(fd[1]);
		read(fd[0], msg, N);
		printf("%s\n", msg);
	}

	int count = 0;
	while (count < 2)
	{
		int status;
		pid_t child_pid = wait(&status);
		printf("Child has finished: PID = %d\n", child_pid);
		if (WIFEXITED(status))
			printf("Child exited with code #%d\n", WEXITSTATUS(status));
		else if (WIFSIGNALED(status))
			printf("Child exited with non-intercepted signal #%d\n",
WTERMSIG(status));
		else if (WIFSTOPPED(status))
			printf("Child stopped, signal #%d\n", WSTOPSIG(status));
		count++;
	}
	return 0;
}
```



Задание 5:

В программу с программным каналом включить собственный обработчик сигнала. Использовать сигнал для изменения хода выполнения программы.

```
#include
<stdio.h>
            #include <stdlib.h>
            #include <unistd.h>
            #include <sys/wait.h>
            #include <signal.h>


            #define BUFFER_SIZE 64
            #define TIME 5


            static int signal_is_catched = 0;
            static int fd[2];
            static char buffer[BUFFER_SIZE];


            void catch_signal(int sig_num)
            {
                signal_is_catched = 1;
            }


            int main()
            {
                  int childPIDS[2];
                  int status, wpid;
                char msg[] = "Important message from child #";


                signal(SIGINT, catch_signal);


                if (pipe(fd) == -1)
                {
                    exit(1);
                }


                  for (int i = 0; i < 2; i++)
                  {
                        if ((childPIDS[i] = fork()) == -1)
                        {
                              perror("Can't fork.\n");
                        exit(1);
                        }
```

```
            if (childPIDS[i] == 0)
            {
                  sleep(TIME);


            if (signal_is_catched)
            {
                msg[30] = '0' + i + 1;
                close(fd[0]);
                write(fd[1], msg, BUFFER_SIZE);
            }
            exit(0);
            }
      }


    sleep(TIME);
    close(fd[1]);
    printf("\n");


    for (int i = 0; i < 2 && signal_is_catched; i++)
    {
        read(fd[0], buffer, BUFFER_SIZE);
        printf("Message from child: %s\n", buffer);
    }


      for (int i = 0; i < 2; i++)
      {
        wpid = wait(&status);
        if (wpid == -1)
        {
                  perror("waitpid");
                  exit(EXIT_FAILURE);
            }

            if (WIFEXITED(status))
            {
                  printf("\tChild exited with status = %d\n",
WEXITSTATUS(status));
            }
            else if (WIFSIGNALED(status))
            {
                  printf("\tChild with (signal %d) killed\n",
WTERMSIG(status));
            }
            else if (WIFSTOPPED(status))
            {
                  printf("\tChild with (signal %d) stopped\n",
WSTOPSIG(status));
            }
            else
                  printf("\tUnexpected status for Child (0x%x)\n",
status);
      }
```

```
        printf("OK!\n");
          return 0;
    }
```

```
parallels@parallels-Parallels-Virtual-Platform:~/os$ gcc lab4.c
parallels@parallels-Parallels-Virtual-Platform:~/os$ ./a.out

        Child exited with status = 0
        Child exited with status = 0
OK!
parallels@parallels-Parallels-Virtual-Platform:~/os$ ./a.out
^C
Message from child: Important message from child #1
Message from child: Important message from child #2
        Child exited with status = 0
        Child exited with status = 0
OK!
parallels@parallels-Parallels-Virtual-Platform:~/os$
```