##### 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.

task1.c

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

int main()

{

int childpid[2];

for (int i = 0; i < 2; i++)

{

childpid[i] = fork();

if (childpid[i] == -1)

{

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

exit(1);

}

else if (childpid[i] == 0)

{

printf("\nChild: id = %d \tparent\_id = %d \tgroup\_id = %d\n", getpid(), getppid(), getpgrp());

sleep(1);

printf("\nChild: id = %d \tparent\_id = %d \tgroup\_id = %d\n", getpid(), getppid(), getpgrp());

return 0;

}

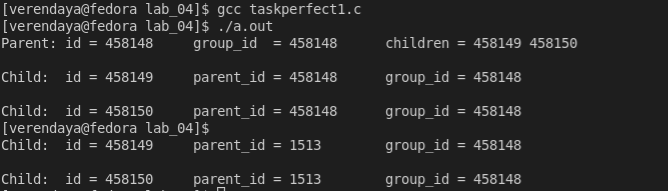
}

printf("Parent: id = %d group\_id = %d \tchildren = %d %d\n", getpid(), getpgrp(), childpid[0], childpid[1]);

return 0;

}

Листинг 1.



Вывод программы 1.

##### 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.

task2.c

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <errno.h>

#include <sys/types.h>

#include <sys/wait.h>

int main()

{

int childpid[2];

int status;

//pid\_t childpidw;

for (int i = 0; i < 2; i++)

{

childpid[i] = fork();

if (childpid[i] == -1)

{

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

exit(1);

}

else if (childpid[i] == 0)

{

printf("\nChild: id = %d \tparent\_id = %d \tgroup\_id = %d\n", getpid(), getppid(), getpgrp());

return 0;

}

}

printf("Parent: id = %d group\_id = %d \tchildren = %d %d\n", getpid(), getpgrp(), childpid[0], childpid[1]);

for (int i = 0; i < 2; i++)

{

printf("\n--- Parent is waiting ---");

pid\_t childpid = wait(&status);

if (childpid == -1)

{

if (errno == ECHILD)

printf("Process does not have any unwaited for children\n");

else if (errno == EINTR)

printf("Call interrupted by signal\n");

else if (errno == EINVAL)

printf("Wrong argument\n");

exit(1);

}

printf("\nChild finished: pid = %d\n", childpid);

if (WIFEXITED(status))

printf("Child exited normally with code %d\n", WEXITSTATUS(status));

else printf("Child terminated abnormally\n");

if (WIFSIGNALED(status))

printf("Child exited due to uncaught signal # %d\n", WTERMSIG(status));

if (WIFSTOPPED(status))

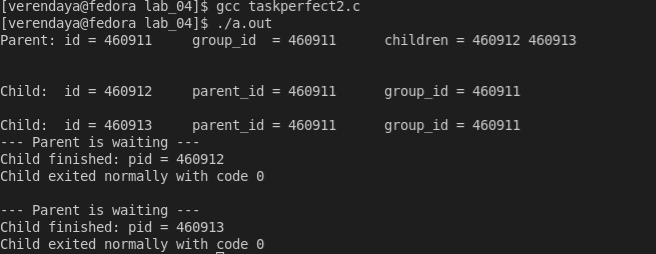
printf("Child stopped, signal # %d\n", WSTOPSIG(status));

}

return 0;

}

Листинг 2.



Вывод программы 2.

##### 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.

task3.c

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <errno.h>

#include <sys/types.h>

#include <sys/wait.h>

int main()

{

int childpid[2];

int status;

//pid\_t childpidw;

for (int i = 0; i < 2; i++)

{

sleep(5);

childpid[i] = fork();

if (childpid[i] == -1)

{

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

exit(1);

}

else if (childpid[i] == 0)

{

printf("\nChild: id = %d \tparent\_id = %d \tgroup\_id = %d\n", getpid(), getppid(), getpgrp());

//sleep(5);

if (i)

{

status = execl("child1.o", NULL);

if (status == -1)

{

printf("Error execl.\n");

exit(1);

}

}

else

{

status = execl("child2.o", NULL);

if (status == -1)

{

printf("Error execl.\n");

exit(1);

}

}

return 0;

}

}

printf("Parent: id = %d group\_id = %d \tchildren = %d %d\n", getpid(), getpgrp(), childpid[0], childpid[1]);

for (int i = 0; i < 2; i++)

{

printf("\n--- Parent is waiting ---");

pid\_t childpid = wait(&status);

if (childpid == -1)

{

if (errno == ECHILD)

printf("Process does not have any unwaited for children\n");

else if (errno == EINTR)

printf("Call interrupted by signal\n");

else if (errno == EINVAL)

printf("Wrong argument\n");

exit(1);

}

printf("\nChild finished: pid = %d\n", childpid);

if (WIFEXITED(status))

printf("Child exited normally with code %d\n", WEXITSTATUS(status));

else printf("Child terminated abnormally\n");

if (WIFSIGNALED(status))

printf("Child exited due to uncaught signal # %d\n", WTERMSIG(status));

if (WIFSTOPPED(status))

printf("Child stopped, signal # %d\n", WSTOPSIG(status));

}

return 0;

}

Листинг 3.

child1.cpp

#include <iostream>

using namespace std;

long double fact(int N)

{

if(N < 0)

return 0;

if (N == 0)

return 1;

else

return N \* fact(N - 1);

}

int main()

{

int N;

cout << "Input number to calculate the factorial: ";

cin >> N;

cout << "Factorial " << N << " = " << fact(N) << endl << endl;

return 0;

}

Листинг 3.2

child2.cpp

#include <iostream>

using namespace std;

bool checkPrimeNumber(int n)

{

bool isPrime = true;

if (n == 0 || n == 1)

{

isPrime = false;

}

else

{

for (int i = 2; i <= n / 2; ++i)

{

if (n % i == 0)

{

isPrime = false; break;

}

}

}

return isPrime;

}

int main()

{

int n;

cout << "Enter a positive integer: ";

cin >> n;

if (checkPrimeNumber(n))

cout << n << " is a prime number.\n";

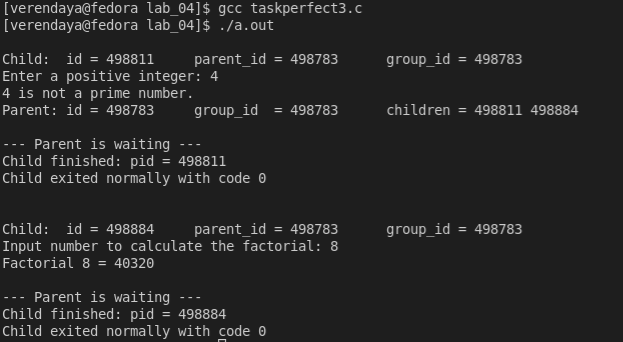
else

cout << n << " is not a prime number.\n";

return 0;

}

Листинг 3.3



Вывод программы 3.

##### Задание 4

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

task4.c

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <errno.h>

#include <sys/types.h>

#include <sys/wait.h>

int main()

{

int childpid[2];

int status;

char getMes[30];

char sendMes[2][24];

sprintf(sendMes[0], "\nHahahahahahahahahahahahaha");

sprintf(sendMes[1], "\nMeh");

int fd[2];

status = pipe(fd);

if (status == -1)

{

printf("Can't pipe\n");

exit(1);

}

for (int i = 0; i < 2; i++)

{

childpid[i] = fork();

if (childpid[i] == -1)

{

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

exit(1);

}

else if (childpid[i] == 0)

{

printf("\nChild: id = %d \tparent\_id = %d \tgroup\_id = %d\n", getpid(), getppid(), getpgrp());

status = close(fd[0]);

if (status == -1)

{

printf("Close error.\n");

exit(i);

}

status = write(fd[1], sendMes[i], sizeof(sendMes[i]));

if (status == -1)

{

printf("Error %d write", i);

}

return 0;

}

}

for (int i = 0; i < 2; i++)

{

printf("\n--- Parent is waiting ---");

pid\_t childpid = wait(&status);

if (childpid == -1)

{

if (errno == ECHILD)

printf("Process does not have any unwaited for children\n");

else if (errno == EINTR)

printf("Call interrupted by signal\n");

else if (errno == EINVAL)

printf("Wrong argument\n");

exit(1);

}

printf("\nChild finished: pid = %d\n", childpid);

if (WIFEXITED(status))

printf("Child exited normally with code %d\n", WEXITSTATUS(status));

else printf("Child terminated abnormally\n");

if (WIFSIGNALED(status))

printf("Child exited due to uncaught signal # %d\n", WTERMSIG(status));

if (WIFSTOPPED(status))

printf("Child stopped, signal # %d\n", WSTOPSIG(status));

}

status = close(fd[1]);

if (status == -1)

{

printf("Close error.\n");

exit(1);

}

status = read(fd[0], getMes, sizeof(getMes));

if (status == -1)

{

printf("Error %d read\n");

exit(1);

}

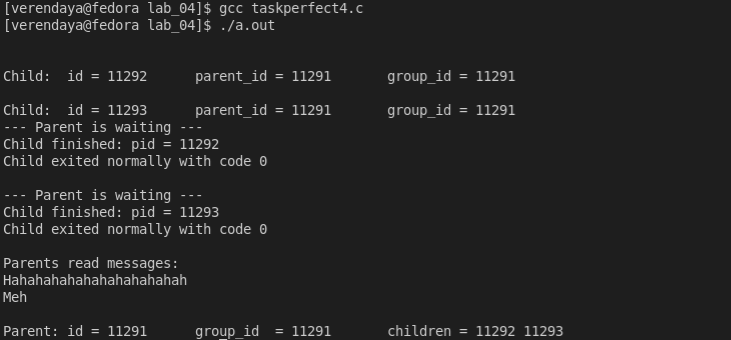
printf("\nParents read messages: %s\n", getMes);

printf("\nParent: id = %d group\_id = %d \tchildren = %d %d\n", getpid(), getpgrp(), childpid[0], childpid[1]);

return 0;

}

Листинг 4.



Вывод программы 4.

##### Задание 5

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

task5.c

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <errno.h>

#include <signal.h>

#include <sys/types.h>

#include <sys/wait.h>

static int sigFlag = 0;

void catch\_sig(int sig\_numb)

{

signal(sig\_numb, catch\_sig);

sigFlag = 1;

printf("catch\_sig %d\n", sig\_numb);

}

int main()

{

int childpid[2];

int status;

char getMes[30];

char sendMes[2][24];

sprintf(sendMes[0], "\nHahahahahahahahahahahahaha");

sprintf(sendMes[1], "\nMeh");

int fd[2];

status = pipe(fd);

if (status == -1)

{

printf("Can't pipe\n");

exit(1);

}

signal(SIGINT, catch\_sig);

sleep(5);

for (int i = 0; i < 2; i++)

{

childpid[i] = fork();

if (childpid[i] == -1)

{

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

exit(1);

}

else if (childpid[i] == 0)

{

printf("\nChild: id = %d \tparent\_id = %d \tgroup\_id = %d\n", getpid(), getppid(), getpgrp());

if (sigFlag){

status = close(fd[0]);

if (status == -1)

{

printf("Close error.\n");

exit(i);

}

status = write(fd[1], sendMes[i], sizeof(sendMes[i]));

if (status == -1)

{

printf("Error %d write", i);

}

}

return 0;

}

}

for (int i = 0; i < 2; i++)

{

printf("\n--- Parent is waiting ---");

pid\_t childpid = wait(&status);

if (childpid == -1)

{

if (errno == ECHILD)

printf("Process does not have any unwaited for children\n");

else if (errno == EINTR)

printf("Call interrupted by signal\n");

else if (errno == EINVAL)

printf("Wrong argument\n");

exit(1);

}

printf("\nChild finished: pid = %d\n", childpid);

if (WIFEXITED(status))

printf("Child exited normally with code %d\n", WEXITSTATUS(status));

else printf("Child terminated abnormally\n");

if (WIFSIGNALED(status))

printf("Child exited due to uncaught signal # %d\n", WTERMSIG(status));

if (WIFSTOPPED(status))

printf("Child stopped, signal # %d\n", WSTOPSIG(status));

}

status = close(fd[1]);

if (status == -1)

{

printf("Close error.\n");

exit(1);

}

status = read(fd[0], getMes, sizeof(getMes));

if (status == -1)

{

printf("Error %d read\n");

exit(1);

}

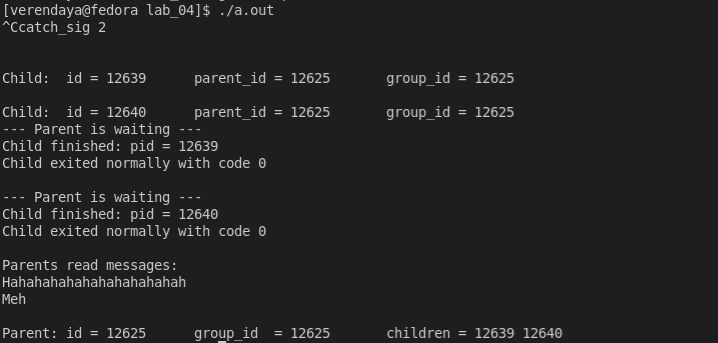
printf("\nParents read messages: %s\n", getMes);

printf("\nParent: id = %d group\_id = %d \tchildren = %d %d\n", getpid(), getpgrp(), childpid[0], childpid[1]);

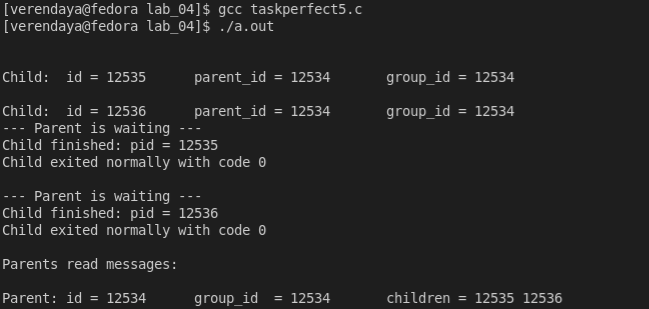
return 0;

}

Листинг 5.



Вывод программы 5(с сигналом ^C)



Вывод программы 5(без сигнала)