**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.

**Исходный код:**

#include <stdio.h>

#include <unistd.h>

int main() {

pid\_t child\_pid[2] = {0, 0};

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

child\_pid[i] = fork();

if (child\_pid[i] == -1) {

printf("Cant fork!\n");

return 1;

} else if (child\_pid[i] == 0) {

printf ("This is child #%d.\n"

"Its pid is %d.\n"

"Its parents pid is %d.\n"

"Its group id is %d.\n\n",

i + 1, getpid(), getppid(), getpgrp());

sleep(2);

printf ("This is child #%d.\n"

"Its pid is %d.\n"

"Its parents pid is %d.\n"

"Its group id is %d.\n\n",

i + 1, getpid(), getppid(), getpgrp());

return 0;

}

}

sleep(1);

printf ("This is the main process.\n"

"Its pid is %d.\n"

"Its group id is %d.\n "

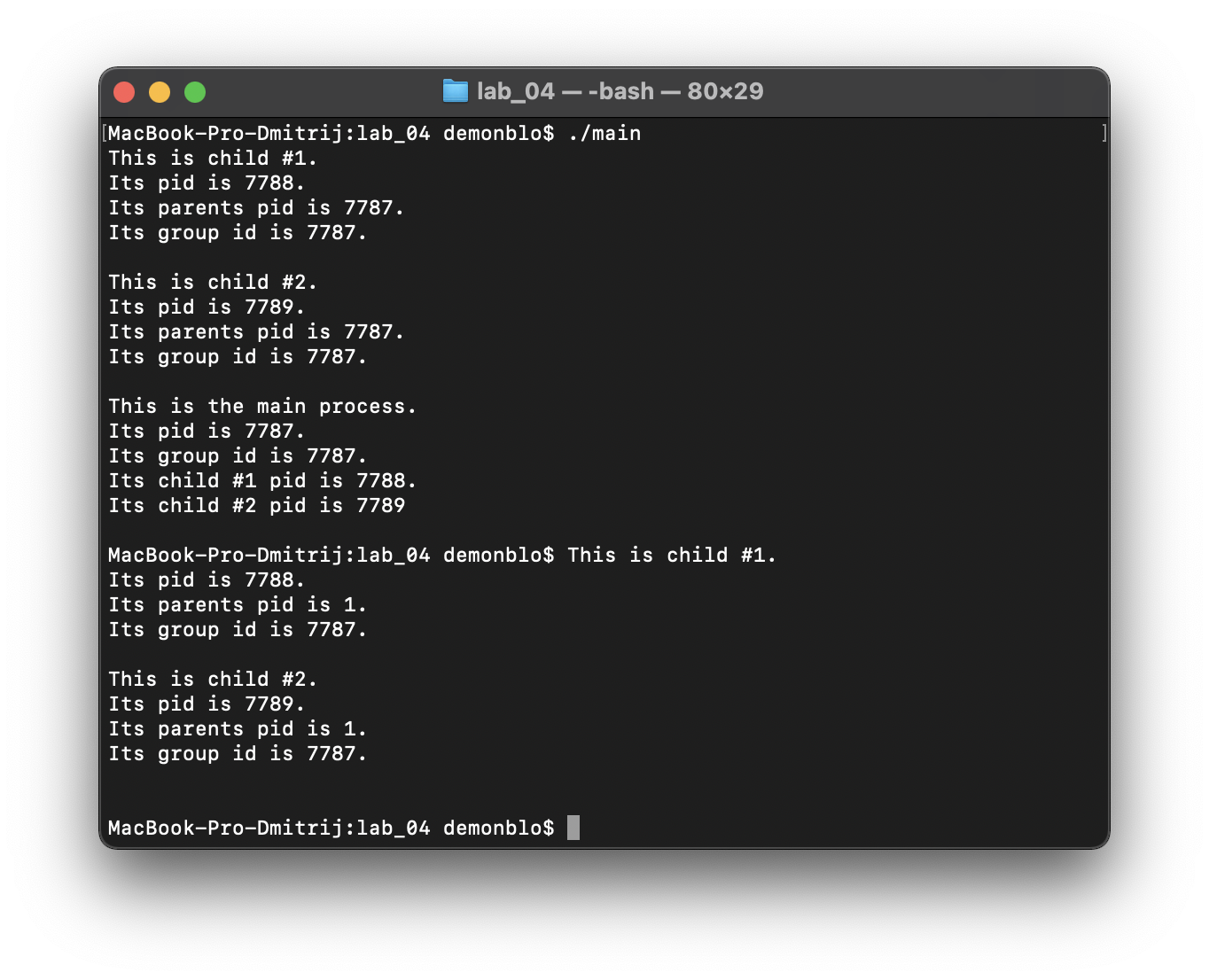
"Its child #1 pid is %d.\n"

"Its child #2 pid is %d\n\n",

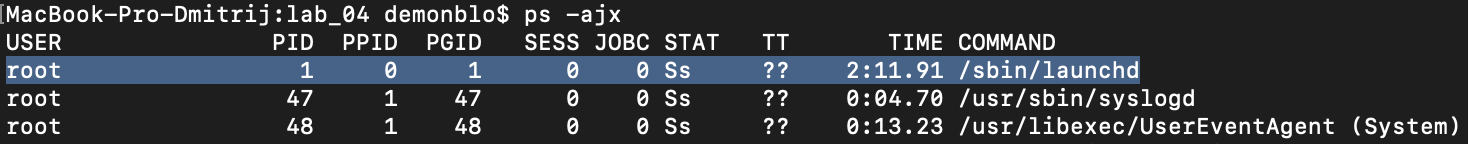
getpid(), getpgrp(), child\_pid[0], child\_pid[1]);

return 0;

}

**Результат работы программы:**

Как видно на скриншоте, оба процесса-потомка после завершения работы процесса-предка были усыновлены процессом-посредником c pid = 1

С помощью команды **ps -ajx** можно увидеть, что это процесс **launchd**

**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.

**Исходный код:**

#include <stdio.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#define FORKS\_AMOUNT 2

int main() {

pid\_t child\_pids[2] = {0, 0};

for (int i = 0; i < FORKS\_AMOUNT; i++) {

child\_pids[i] = fork();

if (child\_pids[i] == -1) {

printf("Cant fork!\n");

return 1;

} else if (child\_pids[i] == 0) {

printf ("This is child #%d.\n"

"Its pid is %d.\n"

"Its parents pid is %d.\n"

"Its group id is %d.\n\n",

i + 1, getpid(), getppid(), getpgrp());

return 0;

}

}

printf ("This is the main process.\n"

"Its pid is %d.\n"

"Its group id is %d.\n"

"Its child #1 pid is %d.\n"

"Its child #2 pid is %d\n\n",

getpid(), getpgrp(), child\_pids[0], child\_pids[1]);

int status = 0;

pid\_t child\_pid = 0;

for (int i = 0; i < FORKS\_AMOUNT; i++) {

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 because of signal number %d\n", WTERMSIG(status));

} else if (WIFSTOPPED(status)) {

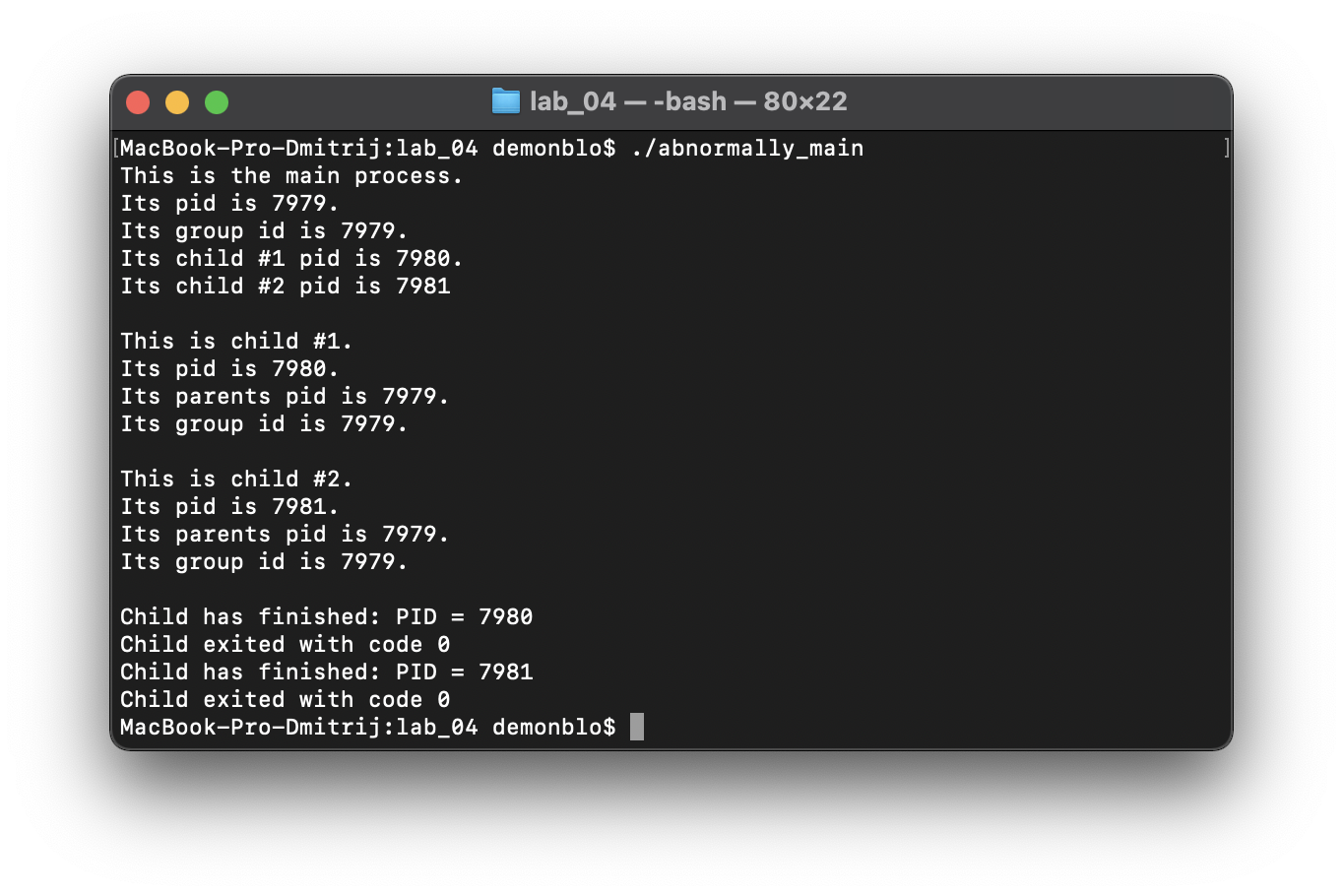
printf("Child stopped with signal number %d\n", WSTOPSIG(status));

}

}

return 0;

}

**Результат работы программы**: 

На скриншоте выше можно заметить, что идентификатор первого процесса-потомка действительно на 1 больше идентификатора процесса-предка

**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.

Исходный код программы:

#include <stdio.h>

#include<stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

int main() {

pid\_t child\_pids[2] = {0, 0};

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

child\_pids[i] = vfork();

if (child\_pids[i] == -1) {

printf("Cant fork!\n");

return 1;

} else if (child\_pids[i] == 0 && i == 0) {

printf ("This is child #%d.\n"

"My pid is %d.\n"

"My parents pid is %d.\n"

"My group id is %d.\n\n",

i + 1, getpid(), getppid(), getpgrp());

execl("/bin/ls", "ls", "-lah", NULL);

exit (0);

} else if (child\_pids[i] == 0 && i == 1) {

printf ("This is child #%d.\n"

"My pid is %d.\n"

"My parents pid is %d.\n"

"My group id is %d.\n\n",

i + 1, getpid(), getppid(), getpgrp());

execl("/bin/ping", "ping", "www.yandex.ru", "-c 5", NULL);

exit (0);

}

}

printf ("This is main process.\n"

"My pid is %d.\n"

"My group id is %d.\n "

"My child #1 pid is %d.\n"

"My child #2 pid is %d\n\n",

getpid(), getpgrp(), child\_pids[0], child\_pids[1]);

int status = 0;

pid\_t child\_pid = 0;

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

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 because of signal number %d\n", WTERMSIG(status));

} else if (WIFSTOPPED(status)) {

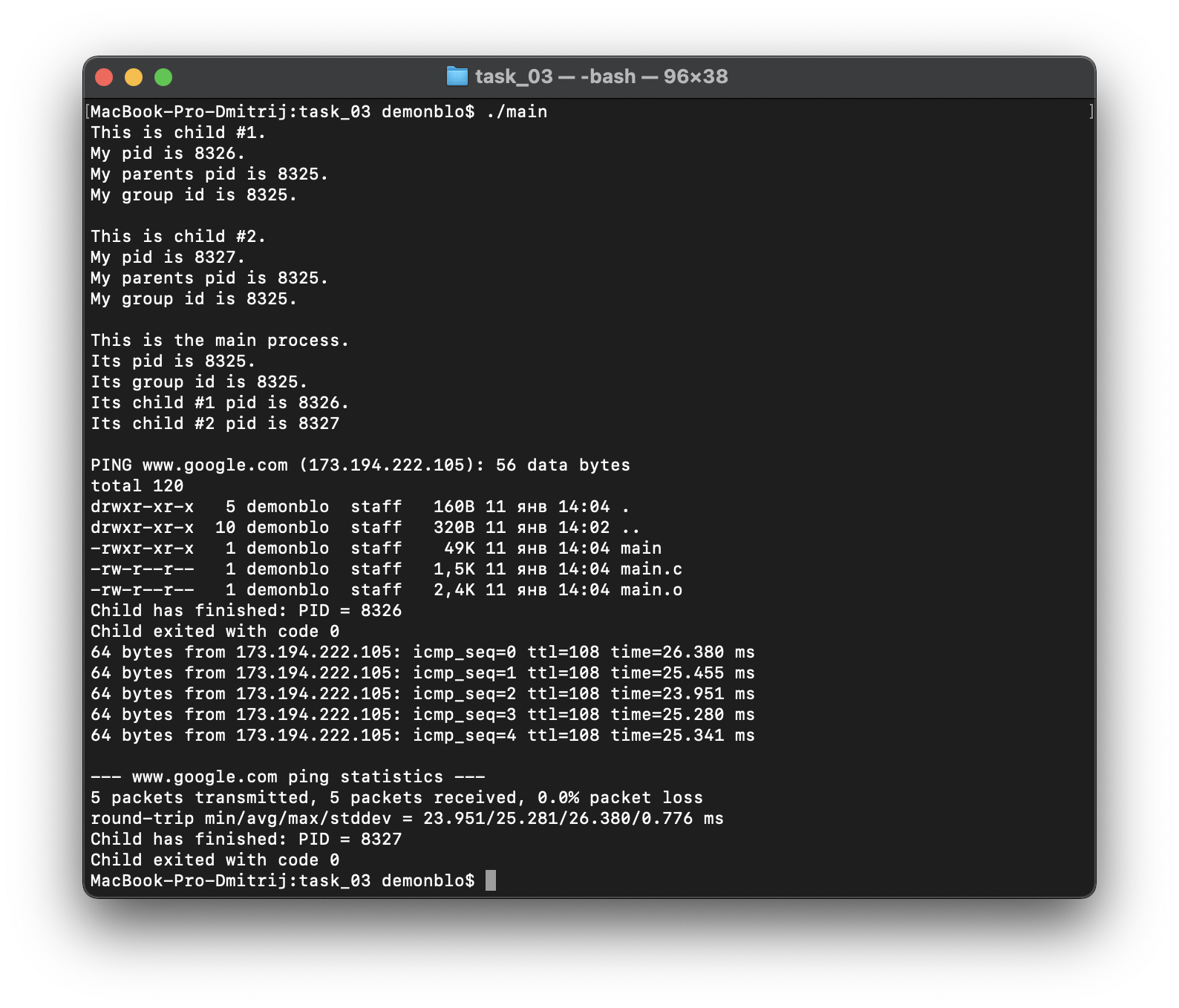
printf("Child stopped with signal number %d\n", WSTOPSIG(status));

}

}

return 0;

}

**Пример работы программы:**

**Задание 4:**

Предок и потомки обмениваются сообщениями через неименованный программный канал. Предок ждет завершения своих потомков. Вывод соответствующих сообщений на экран.

Исходный код:

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <string.h>

#include <unistd.h>

#define BUFFER\_SIZE 50

int main() {

pid\_t child\_pid1 = 0;

pid\_t child\_pid2 = 0;

char\* messages[2] = {"aaaaaaaaaaaauuuuuuuuuuuuuuuu6666666",

"33333333eeeeeeeeennnnnnnnnnnnnnnnnnnn"};

int fd[2] = {0, 0};

if (pipe(fd) == -1) {

printf("Cant make pipe!\n");

return -1;

}

if ((child\_pid1 = fork()) == 0) {

close(fd[0]);

if (!write(fd[1], messages[0], BUFFER\_SIZE)) {

printf("Cant write in pipe!\n");

}

return 0;

}

int status = 0;

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 because of signal nubmer %d\n", WTERMSIG(status));

} else if (WIFSTOPPED(status)) {

printf("Child is stopped with signal nubmer %d\n", WSTOPSIG(status));

}

if ((child\_pid2 = fork()) == 0) {

close(fd[0]);

if (!write(fd[1], messages[1], BUFFER\_SIZE)) {

printf("Cant write in pipe!\n");

}

return 0;

}

status = 0;

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 because of signal nubmer %d\n", WTERMSIG(status));

} else if (WIFSTOPPED(status)) {

printf("Child is stopped with signal nubmer %d\n", WSTOPSIG(status));

}

char buffer1[BUFFER\_SIZE];

char buffer2[BUFFER\_SIZE];

close(fd[1]);

read(fd[0], buffer1, BUFFER\_SIZE);

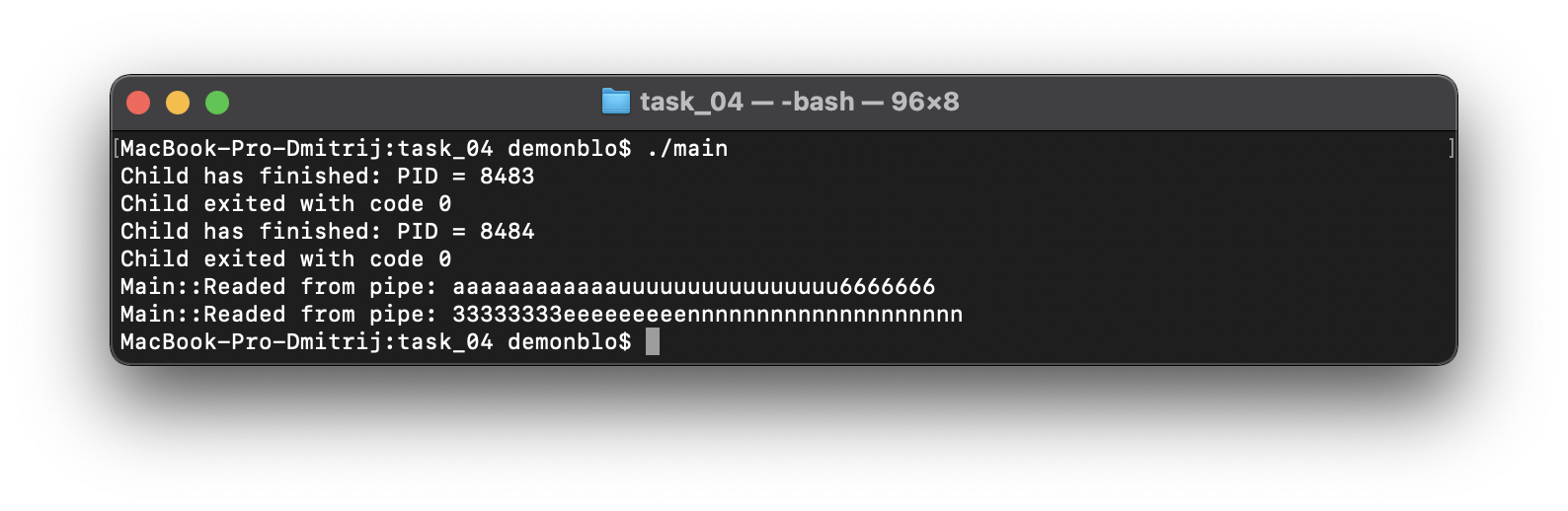
read(fd[0], buffer2, BUFFER\_SIZE);

printf("Main::Readed from pipe: %s\n", buffer1);

printf("Main::Readed from pipe: %s\n", buffer2);

return 0;

}

Пример работы:

**Задание 5:**

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

В написанной программе добавлен обработчик сигнала Quit с терминала (Ctr + \). Предок и потомки обмениваются сообщениями через программный канал. Сообщения разной длины и содержания. У пользователя есть 10 секунд, чтобы нажать комбинацию клавиш “Ctr + \”, тогда в канал будут записаны сообщения от потомков.

**Исходный код**:

#include <sys/types.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#include <string.h>

#define BUFFER\_SIZE 100

typedef enum flag {

UNCATCHED = 0,

CATCHED,

} flag\_t;

flag\_t flag = UNCATCHED;

pid\_t child\_pid1 = 0;

pid\_t child\_pid2 = 0;

void my\_SIGQUIT\_handler(int signal) {

flag = CATCHED;

}

int main() {

int fd[2] = {0, 0};

char\* messages[2] = {"aaaaaaaaaaaauuuuuuuuuuuuuuuu6666666",

"33333333eeeeeeeeennnnnnnnnnnnnnnnnnnn"};

if (pipe(fd) == -1) {

printf("Cant make pipe!\n");

return -1;

}

void (\*old\_handler)(int) = signal(SIGQUIT, my\_SIGQUIT\_handler);

if ((child\_pid1 = fork()) == 0) {

sleep(10);

if (flag == CATCHED) {

close(fd[0]);

if (!write(fd[1], messages[0], BUFFER\_SIZE)) {

printf("Cant write in pipe!\n");

}

}

return 0;

}

if ((child\_pid2 = fork()) == 0) {

sleep(10);

if (flag == CATCHED) {

close(fd[0]);

if (!write(fd[1], messages[1], BUFFER\_SIZE)) {

printf("Cant write in pipe!\n");

}

}

return 0;

}

printf("You have 10 second for inputing Ctrl+\\ combination to write messages in channel:\n");

int status = 0;

pid\_t child\_pid = wait(&status);

printf("\nChild 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 because of signal nubmer %d\n", WTERMSIG(status));

} else if (WIFSTOPPED(status)) {

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

}

status = 0;

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 because of signal nubmer %d\n", WTERMSIG(status));

} else if (WIFSTOPPED(status)) {

printf("Child is stopped with signal nubmer %d\n", WSTOPSIG(status));

}

char buffer1[BUFFER\_SIZE]={“”};

char buffer2[BUFFER\_SIZE]={“”};

close(fd[1]);

read(fd[0], buffer1, BUFFER\_SIZE);

read(fd[0], buffer2, BUFFER\_SIZE);

printf("Main::Readed from pipe: %s\n", buffer1);

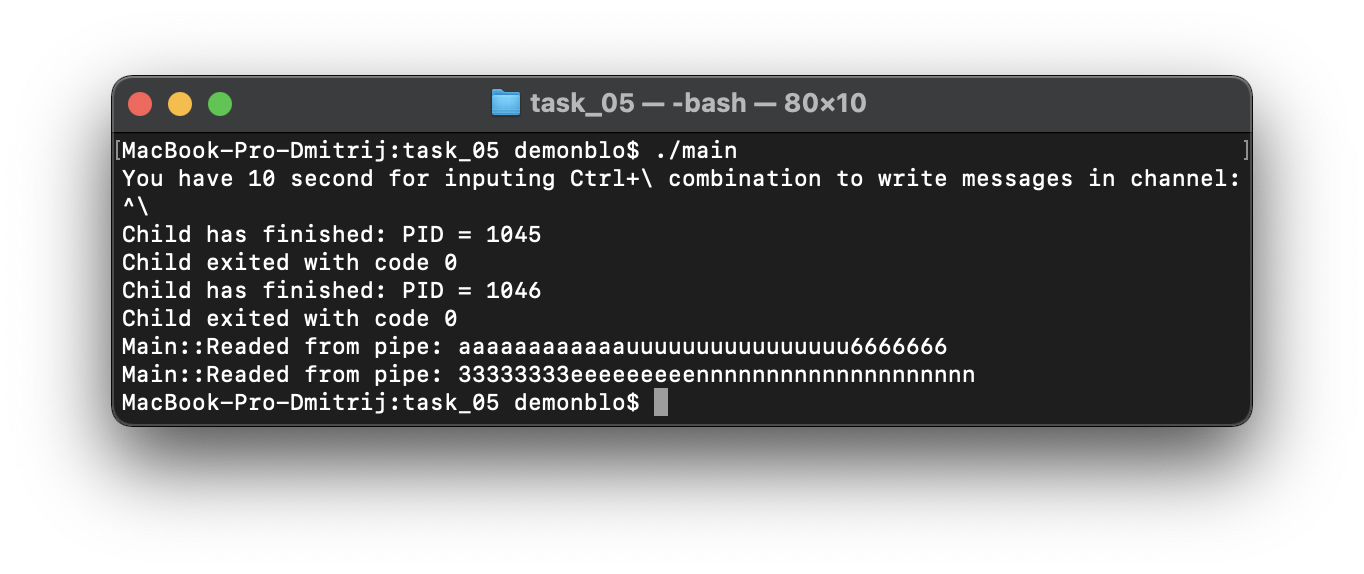
printf("Main::Readed from pipe: %s\n", buffer2);

return 0;

}

Примеры работы:

1. Сигнал SIGQUIT



1. Комбинация не была нажата

