|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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»

**Отчёт по лабораторной работе №4 по курсу**

**«Операционные системы»**

**Тема**  Процессы. Системные вызовы fork() и exec()

**Студент**  Якуба Д.В.

**Группа**  ИУ7-53Б

**Оценка (баллы)**

**Преподаватель**  Рязанова Н.Ю.

*Москва, 2020 г*

# Задание 1

Процессы-сироты. В программе создаются не менее двух потомков. В потомках вызывается sleep(). Чтобы предок гарантированно завершился раньше своих потомков. Продемонстрировать с помощью соответствующего вывода информацию об идентификаторах процессов и их группе.

Листинг кода 1

|  |
| --- |
| #include <sys/types.h>  #include <unistd.h>  #include <stdio.h>  #include <stdlib.h>  #define CANT\_FORK\_ERROR 1  #define SUCCESS 0   int main(void)   {      printf("From Parent. Parent identifiers: parentProcID is %d, groupID is %d\n", getpid(), getpgrp());      pid\_t childpid;      int children[2];        for (int i = 0; i < 2; i++)      {          if ((childpid = fork()) == -1)          {              perror("Can't fork");              exit(CANT\_FORK\_ERROR);          }          else if (childpid == 0)          {              sleep(1);              printf("From child. Child identifiers: childProcID is %d, groupID is %d, parentID is %d\n", getpid(), getpgrp(), getppid());              exit(SUCCESS);          }          else              children[i] = childpid;        }      printf("Children IDs from parent proccess: %d and %d\nEnd of parent existence\n\n", children[0], children[1]);      return SUCCESS;   } |

# Задание 2

Рисунок 1 Демонстрация работа написанной программы

Предок ждет завершения своих потомков, используя системный вызов wait(). Вывод соответствующих сообщений на экран.

Листинг кода 2

|  |
| --- |
| #include <sys/types.h>  #include <sys/wait.h>  #include <unistd.h>  #include <stdio.h>  #include <stdlib.h>  #define CANT\_FORK\_ERROR 1  #define SUCCESS 0   int main(void)   {      printf("From Parent. Parent identifiers: parentProcID is %d, groupID is %d\n", getpid(), getpgrp());      pid\_t childpid;      int children[2];        for (int i = 0; i < 2; i++)      {          if ((childpid = fork()) == -1)          {              perror("Can't fork");              return CANT\_FORK\_ERROR;          }          else if (childpid == 0)          {              sleep(1);              printf("From child. Child identifiers: childProcID is %d, groupID is %d, parentID is %d\n", getpid(), getpgrp(), getppid());              return SUCCESS;          }          else              children[i] = childpid;        }      int childStatus;      for (int i = 0; i < 2; i++)      {          childpid = wait(&childStatus);          printf("Child has finished: PID = %d with status: %d\n", childpid, childStatus);          if (WIFEXITED(childStatus))              printf("Child exited with code %d\n", WEXITSTATUS(childStatus));          else              printf("Child terminated abnormally\n");      }      printf("Children IDs from parent proccess: %d and %d\nEnd of parent existence\n\n", children[0], children[1]);      return SUCCESS;   } |

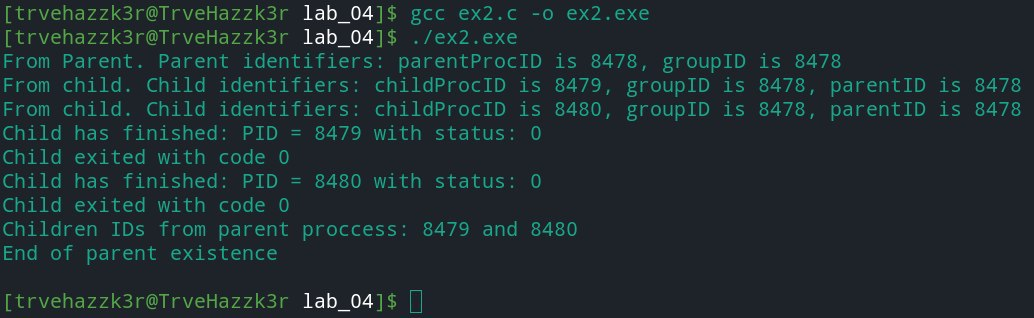


Рисунок 2 Демонстрация работы написанной программы

# Задание 3

Потомки переходят на выполнение других программ. Предок ждет завершения своих потомков. Вывод соответствующих сообщений на экран.

Листинг кода 3

|  |
| --- |
| #include <sys/types.h>  #include <sys/wait.h>  #include <unistd.h>  #include <stdio.h>  #include <stdlib.h>  #define CANT\_FORK\_ERROR 1  #define CANT\_EXECLP\_ERROR 2  #define SUCCESS 0   int main(void)   {      printf("From Parent. Parent identifiers: parentProcID is %d, groupID is %d\n", getpid(), getpgrp());      pid\_t childpid;      int children[2];      char \*commands[2] = {"gcc", "ls"};      char \*arguments[2] = {"--version", "-a"};        for (int i = 0; i < 2; i++)      {          if ((childpid = fork()) == -1)          {              perror("Can't fork");              exit(CANT\_FORK\_ERROR);          }          else if (childpid == 0)          {              printf("From child. Child identifiers: childProcID is %d, groupID is %d, parentID is %d\n", getpid(), getpgrp(), getppid());                if (execlp(commands[i], commands[i], arguments[i], NULL) < 0)                  {                      perror("Can't execlp");                      exit(CANT\_EXECLP\_ERROR);                  }              exit(SUCCESS);          }          else              children[i] = childpid;        }      printf("\n");      int childStatus;      for (int i = 0; i < 2; i++)      {          childpid = wait(&childStatus);          printf("Child has finished: PID = %d; with status: %d\n", childpid, childStatus);            if (WIFEXITED(childStatus))              printf("Child exited with code %d\n", WEXITSTATUS(childStatus));          else              printf("Child terminated abnormally\n");      }      printf("Children IDs from parent proccess: %d and %d\nEnd of parent existence\n\n", children[0], children[1]);      return SUCCESS;   } |

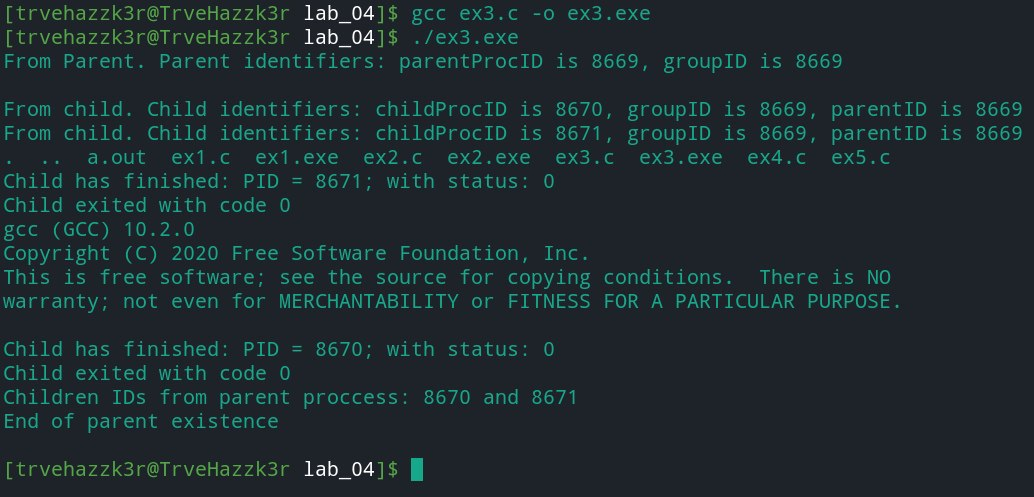


Рисунок 3 Демонстрация работы написанной программы

# Задание 4

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

Листинг кода 4

|  |
| --- |
| #include <sys/types.h>  #include <sys/wait.h>  #include <unistd.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #define CANT\_FORK\_ERROR 1  #define CANT\_EXECLP\_ERROR 2  #define CANT\_CREATE\_PIPE\_ERROR 3  #define SUCCESS 0   int main(void)   {      printf("From Parent. Parent identifiers: parentProcID is %d, groupID is %d\n", getpid(), getpgrp());      int fd[2];      pid\_t childpid;      int children[2];      char \*pipeMessages[2] = {"Message №1\n", "Message №2\n"};      if (pipe(fd) == -1)      {          perror("Can't create pipe");          exit(CANT\_CREATE\_PIPE\_ERROR);      }        for (int i = 0; i < 2; i++)      {          if ((childpid = fork()) == -1)          {              perror("Can't fork");              exit(CANT\_FORK\_ERROR);          }          else if (childpid == 0)          {              close(fd[0]);              write(fd[1], pipeMessages[i], strlen(pipeMessages[i]));              printf("Message №%d was sent\n", i + 1);              exit(SUCCESS);          }          else              children[i] = childpid;        }      printf("\n");      int childStatus;      for (int i = 0; i < 2; i++)      {          childpid = wait(&childStatus);          printf("Child has finished: PID = %d; with status: %d\n", childpid, childStatus);            if (WIFEXITED(childStatus))              printf("Child exited with code %d\n", WEXITSTATUS(childStatus));          else              printf("Child terminated abnormally\n");      }      char gotMessages[32] = { 0 };      close(fd[1]);      if (read(fd[0], gotMessages, 32) > 0)          printf("Received from children:\n%s\n", gotMessages);      else          printf("No messages from children.\n");      printf("Children IDs from parent proccess: %d and %d\nEnd of parent existence\n\n", children[0], children[1]);      return SUCCESS;   } |

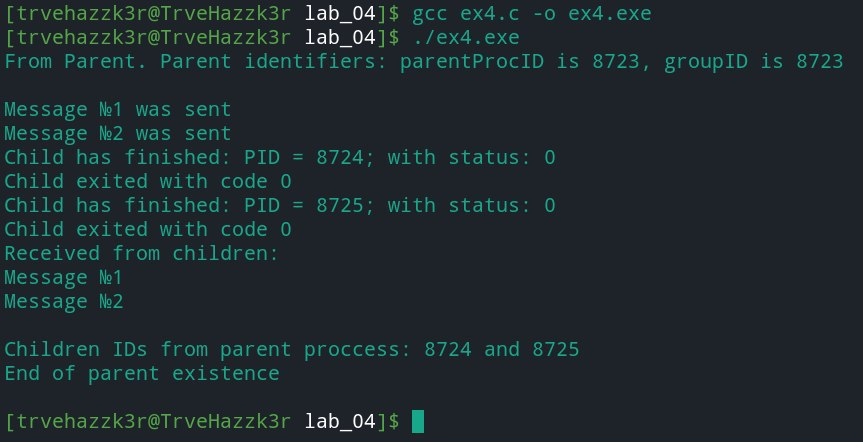


Рисунок 4 Демонстрация работы написанной программы

# Задание 5

Предок и потомки обмениваются сообщениями через неименованный программный канал. С помощью сигнала меняется ход выполнения программы. Предок ждет завершения своих потомков. Вывод соответствующих сообщений на экран.

Листинг кода 5

|  |
| --- |
| #include <sys/types.h>  #include <sys/wait.h>  #include <unistd.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #define CANT\_FORK\_ERROR 1  #define CANT\_EXECLP\_ERROR 2  #define CANT\_CREATE\_PIPE\_ERROR 3  #define SUCCESS 0  short canWrite = 0;  void noSIGTSTP()  {  }  void makeCanWriteTrue()  {      canWrite = 1;  }   int main(void)   {      printf("From Parent. Parent identifiers: parentProcID is %d, groupID is %d\n", getpid(), getpgrp());      int fd[2];      pid\_t childpid;      int children[2];      char \*pipeMessages[2] = {"Message №1\n", "Message №2\n"};      if (pipe(fd) == -1)      {          perror("Can't create pipe");          exit(CANT\_CREATE\_PIPE\_ERROR);      }      signal(SIGTSTP, noSIGTSTP);        for (int i = 0; i < 2; i++)      {          if ((childpid = fork()) == -1)          {              perror("Can't fork");              exit(CANT\_FORK\_ERROR);          }          else if (childpid == 0)          {              signal(SIGTSTP, makeCanWriteTrue);              sleep(4);              if (canWrite)              {                  close(fd[0]);                  write(fd[1], pipeMessages[i], strlen(pipeMessages[i]));                  printf("Message №%d was sent\n", i + 1);              }              else                  printf("Message №%d was NOT sent\n", i + 1);              exit(SUCCESS);          }          else              children[i] = childpid;        }      printf("\n\n");      int childStatus;      for (int i = 0; i < 2; i++)      {          childpid = wait(&childStatus);          printf("Child has finished: PID = %d; with status: %d\n", childpid, childStatus);            if (WIFEXITED(childStatus))              printf("Child exited with code %d\n", WEXITSTATUS(childStatus));          else              printf("Child terminated abnormally\n");      }      char gotMessages[32] = { 0 };      close(fd[1]);      if (read(fd[0], gotMessages, 32) > 0)          printf("Received from children:\n%s\n", gotMessages);      else          printf("No messages from children.\n");      printf("Children IDs from parent proccess: %d and %d\nEnd of parent existence\n\n", children[0], children[1]);      return SUCCESS;   } |

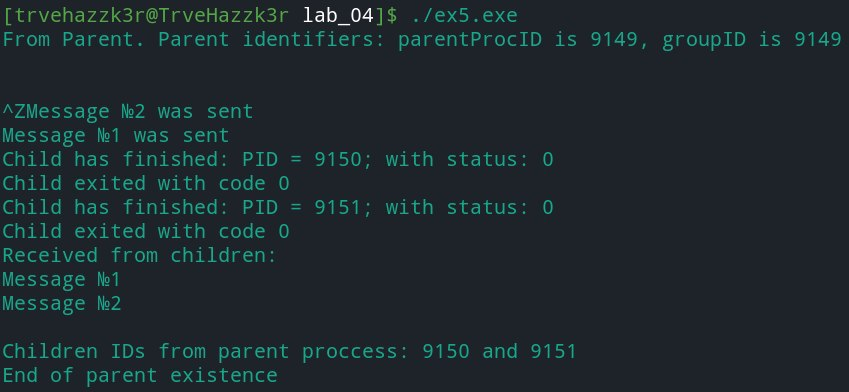


Рисунок 5 Демонстрация работы написанной программы, сигнал вызывается

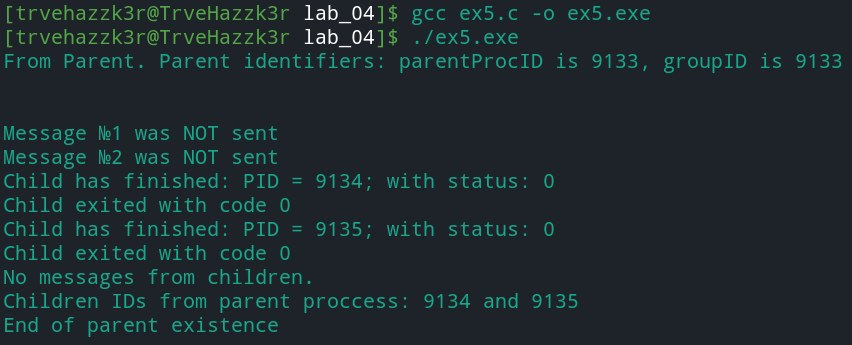


Рисунок 6 Демонстрация работы написанной программы, сигнал не вызывается