Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

Лабораторная работа №3

По дисциплине «Операционные системы»

На тему «Управление процессами»

Выполнила:

Студентка 3 курса 7 группы

Гибадулина Алина Фархадовна

Преподаватель: асс. Бернацкий П.В.

2025, Минск

# 1 **Введение**

**Цель работы:** Получение практических навыков в работе с процессами в OC Windows и Linux.

**Используемые инструменты:**

* Утилита Sysinternals Process Explorer ([ссылка](https://learn.microsoft.com/en-us/sysinternals/downloads/process-explorer));
* Работа с процессами в Windows (лекция и [документация](https://learn.microsoft.com/en-us/windows/win32/procthread/processes-and-threads)) и POSIX (лекция и [документация](https://pubs.opengroup.org/onlinepubs/9799919799/functions/fork.html));
* Работа с IPC (лекция, [Linux](https://support.tools/linux-pipes-ipc-mastery/), [Windows](https://learn.microsoft.com/en-us/windows/win32/ipc/interprocess-communications))

# 2 Основная часть

## 2.1 Выполнение задания для Windows

### 2.1.1 Приложение Lab-03х:

Создать программу, выполняющую цикл с задержкой 500 мс и выводящую PID на каждой итерации.

Количество итераций передаётся:

* через аргумент командной строки,
* если аргумента нет — через переменную окружения ITER\_NUM,
* если нет и переменной — программа завершает работу с ошибкой.

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include <windows.h>  #include <iostream>  #include <string>  #include <stdlib.h>  int main(int argc, char\* argv[]) {  int iterations = 0;  if (argc > 1) {  iterations = atoi(argv[1]);  }  else {  char\* env\_var = getenv("ITER\_NUM");  if (env\_var != NULL) {  iterations = atoi(env\_var);  }  }  if (iterations <= 0) {  std::cerr << "Error: number of iterations not specified!" << std::endl;  ExitProcess(1);  }  std::cout << "Number of iterations: " << iterations << std::endl;  std::cout << "PID: " << GetCurrentProcessId() << std::endl;  for (int i = 1; i <= iterations; i++) {  std::cout << "Iteration " << i << ", PID: " << GetCurrentProcessId() << std::endl;  Sleep(500);  }  std::cout << "Process " << GetCurrentProcessId() << " completed" << std::endl;  return 0;  } |

Листинг 2.1.1.1 – Исходный код программы Lab-03x

Теперь выполним программу без параметров командной строки:

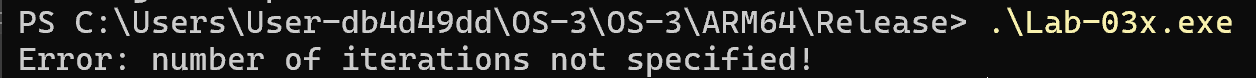


Рисунок 2.1.1.1 – Работа программы Lab-03x при отсутсвии параметров

Теперь выполним программу с параметром командной строки:

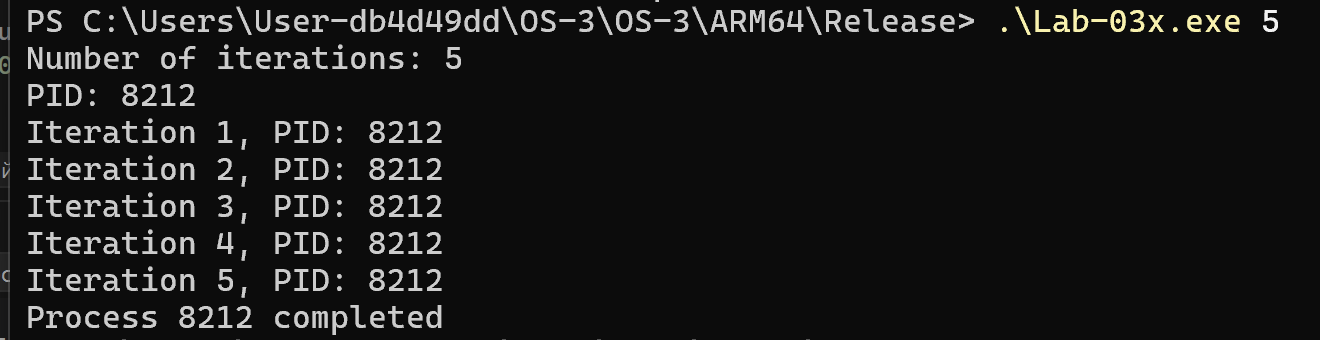


Рисунок 2.1.1.2 – Работа программы Lab-03x при передаче количества итераций через аргумент

Теперь выполним её без аргументов, но задав переменную окружения:

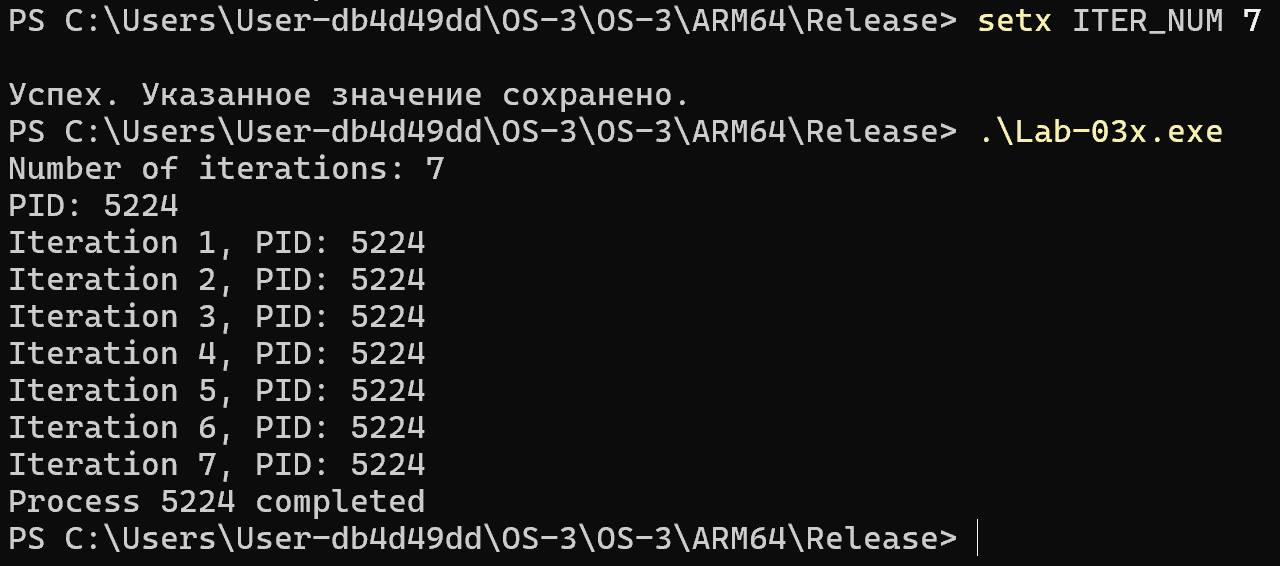


Рисунок 2.1.1.3 – Получение количества итераций из переменной окружения ITER\_NUM

Далее проведем тест, который демонстрирует корректную работу с Proccess Explorer.

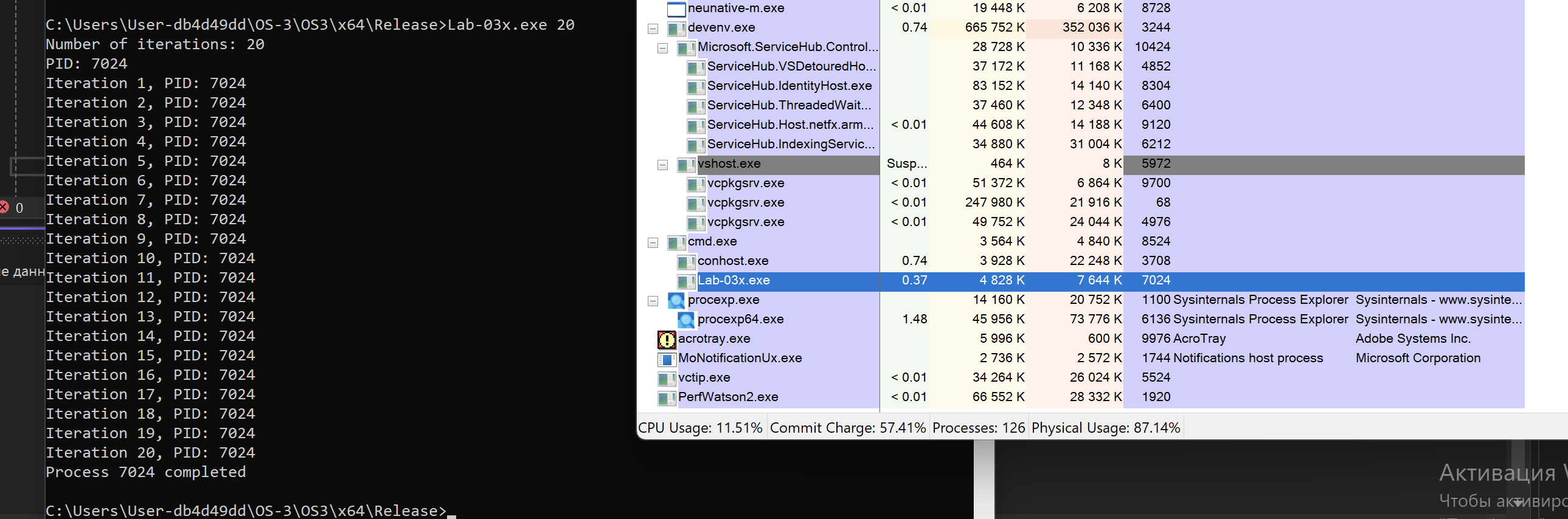


Рисунок 2.1.1.4 – Работа программы Lab-03x с Proccess Explorer

Результат: PID в программе = 7024. PID в Process Explorer: 7024. Совпадение подтверждено.

### 2.1.2 Приложение Lab-03a

Создать программу, которая запускает три дочерних процесса для Lab-03x.exe с разными вариантами передачи аргументов функции CreateProcess.

1. При создании первого процесса используется только первый параметр функции **CreateProcess**. Количество итераций также требуется указать в первом параметре, так задумано (прим. «**Lab-03х.exe 50**»).
2. При создании второго процесса используется только второй параметр функции **CreateProcess**. Надо взять строку из предыдущего пункта и передать во второй параметр.
3. При создании третьего процесса для передачи имени программы используется – первый параметр, для количества итераций – второй. (при этом во втором параметре вам может понадобиться пробел в начале строки)

Родительский процесс должен дождаться завершения всех дочерних процессов (выполняются одновременно), перед тем как завершиться самому.

|  |
| --- |
| #include <windows.h>  #include <iostream>  int main() {  STARTUPINFOA si[3];  PROCESS\_INFORMATION pi[3];  BOOL success[3];  // Инициализация структур  for (int i = 0; i < 3; i++) {  ZeroMemory(&si[i], sizeof(si[i]));  si[i].cb = sizeof(si[i]);  ZeroMemory(&pi[i], sizeof(pi[i]));  }  std::cout << "Parent process PID: " << GetCurrentProcessId() << std::endl;  // ============================  // 1. Первый процесс  // Используем только lpApplicationName с указанием аргумента  std::cout << "\n1. Creating process (lpApplicationName only)..." << std::endl;  success[0] = CreateProcessA(  (LPSTR)"Lab-03x.exe 4", // lpApplicationName + аргумент  NULL, // lpCommandLine  NULL, NULL, FALSE, 0, NULL, NULL,  &si[0], &pi[0]  );  // ============================  // 2. Второй процесс  // Используем только lpCommandLine  std::cout << "2. Creating process (lpCommandLine only)..." << std::endl;  success[1] = CreateProcessA(  NULL, // lpApplicationName  (LPSTR)"Lab-03x.exe 6", // lpCommandLine  NULL, NULL, FALSE, 0, NULL, NULL,  &si[1], &pi[1]  );  // ============================  // 3. Третий процесс  // Используем оба параметра  std::cout << "3. Creating process (both parameters)..." << std::endl;  success[2] = CreateProcessA(  (LPSTR)"Lab-03x.exe", // lpApplicationName  (LPSTR)" 8", // lpCommandLine (с пробелом!)  NULL, NULL, FALSE, 0, NULL, NULL,  &si[2], &pi[2]  );  // ============================  // Вывод результатов  for (int i = 0; i < 3; i++) {  std::cout << "Process " << (i + 1) << ": "  << (success[i] ? "SUCCESS" : "FAILED")  << ", PID: " << (success[i] ? pi[i].dwProcessId : 0)  << std::endl;  }  // ============================  // Ожидание завершения всех дочерних процессов  std::cout << "\nWaiting for child processes to complete..." << std::endl;  for (int i = 0; i < 3; i++) {  if (success[i]) {  WaitForSingleObject(pi[i].hProcess, INFINITE);  CloseHandle(pi[i].hProcess);  CloseHandle(pi[i].hThread);}  }  std::cout << "All child processes completed!" << std::endl;  return 0;  } |

Листинг 2.1.2.1 – Исходный код программы Lab-03a, создающей три дочерних процесса

Результат выполнения представлен на рисунке 2.1.2.1.

Все ли процессы созданы успешно? Нет, не все процессы созданы успешно. Первый процесс завершился с ошибкой из-за того, что lpApplicationName не может содержать аргументы.

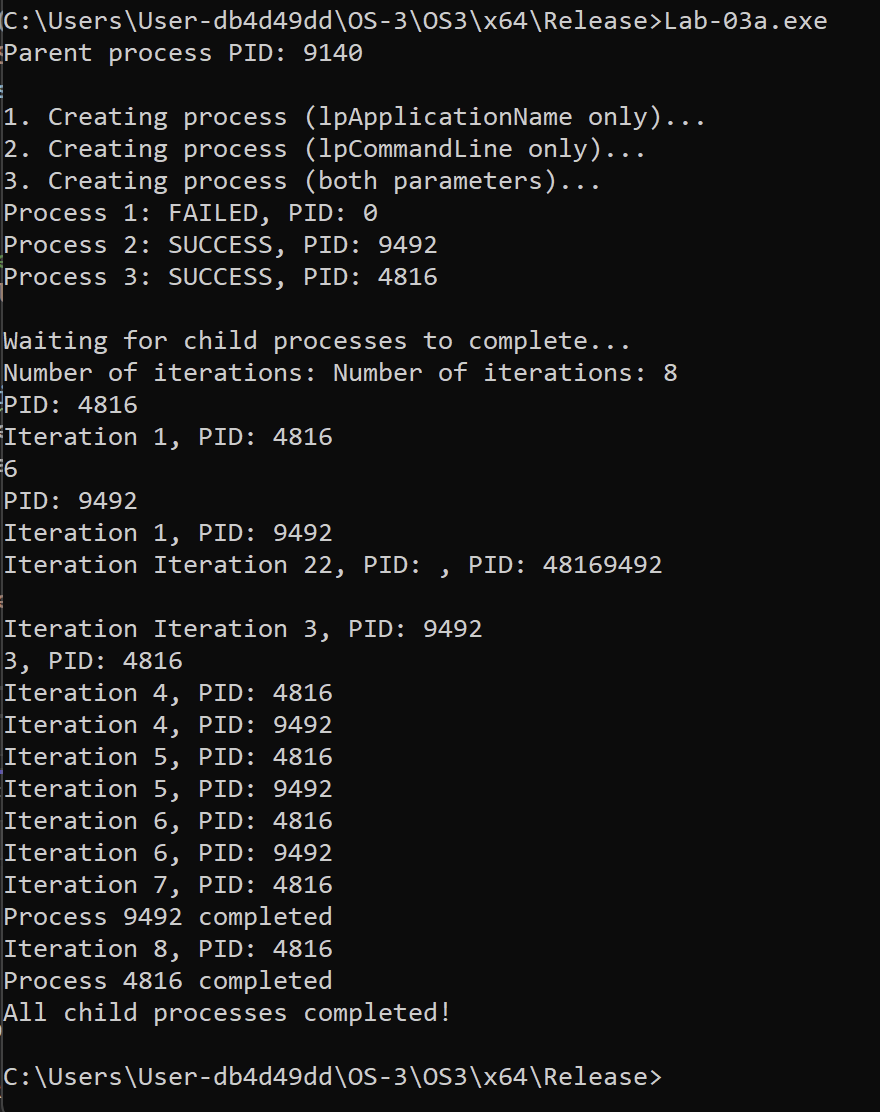


Рисунок 2.1.2.1 – Одновременное выполнение дочерних процессов Lab-03x.exe, запущенных из Lab-03a.exe

Теперь выполним программу и проведем тест, который демонстрирует корректную работу с Proccess Explorer:

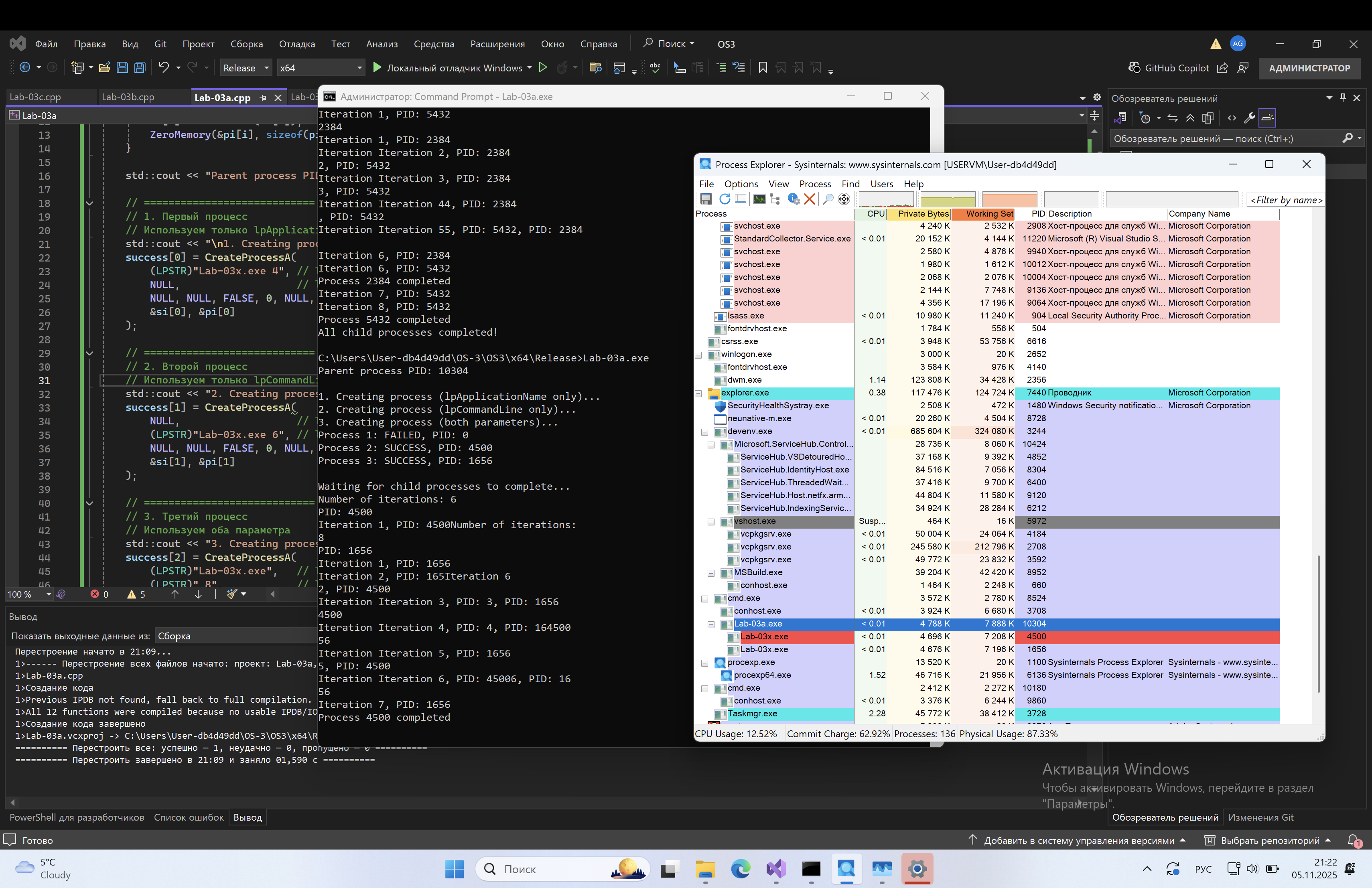


Рисунок 2.1.2.2 – Работа программы Lab-03а с Proccess Explorer

Просмотр информации о созданных процессах при помощи cmd:

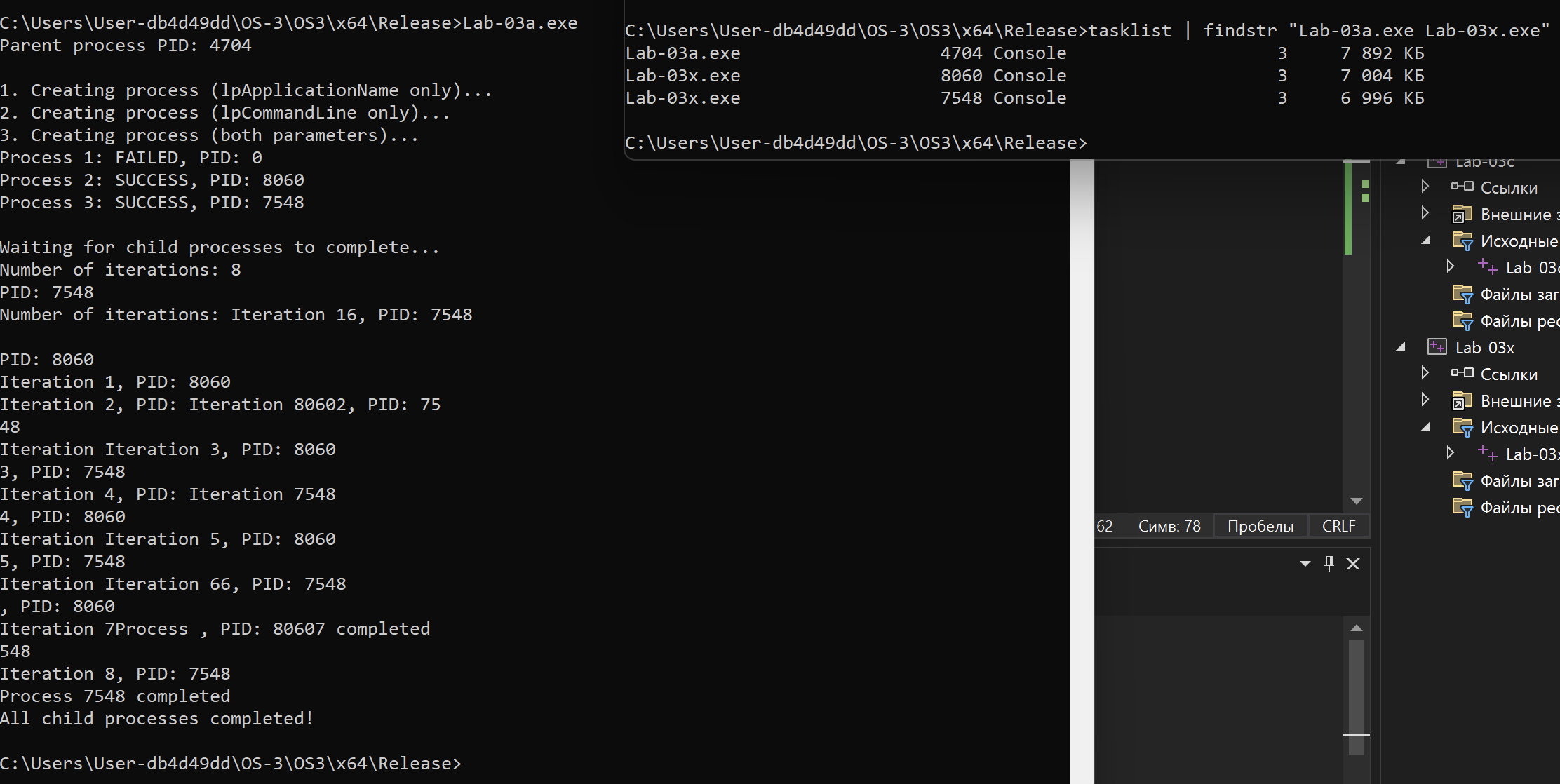


Рисунок 2.1.2.3 – Информация о процессах через cmd

Просмотр информации о созданных процессах при помощи Диспетчера задач:

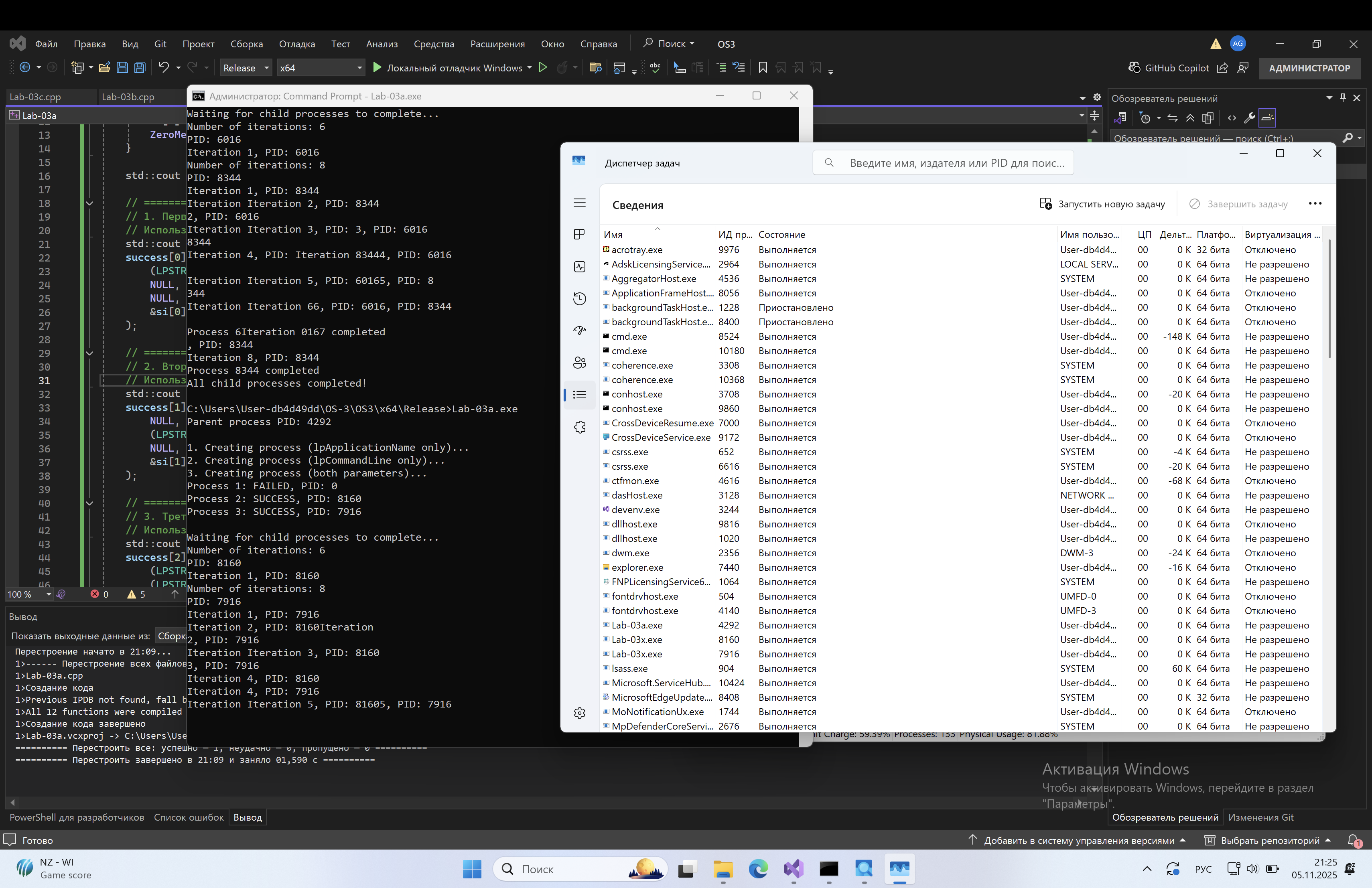


Рисунок 2.1.2.4 – Информация о процессах через Диспетчер задач

### 2.1.3 Приложение Lab-03b.exe

Аналогично Lab-03a.exe, но изменён порядок передачи аргументов:

* первый процесс запускается без указания количества итераций;
* третий процесс запускается с NULL во втором параметре,
* а значение переменной окружения устанавливается локально.

|  |
| --- |
| #include <windows.h>  #include <iostream>  int main() {  STARTUPINFOA si[3];  PROCESS\_INFORMATION pi[3];  BOOL success[3];  // Инициализация структур  for (int i = 0; i < 3; i++) {  ZeroMemory(&si[i], sizeof(si[i]));  si[i].cb = sizeof(si[i]);  ZeroMemory(&pi[i], sizeof(pi[i]));  }  std::cout << "Parent PID: " << GetCurrentProcessId() << std::endl;  // --- Установим глобальную переменную окружения ITER\_NUM ---  SetEnvironmentVariableA("ITER\_NUM", "4");  std::string exe = "Lab-03x.exe";  // Первый процесс — только имя файла (без аргументов)  std::cout << "\n1. Creating process (only filename)..." << std::endl;  success[0] = CreateProcessA(  exe.c\_str(), // lpApplicationName  NULL, // lpCommandLine — отсутствует  NULL, NULL, FALSE, 0,  NULL, NULL, &si[0], &pi[0]  );  // Второй процесс — как обычно, с аргументом  std::cout << "2. Creating process (with command line argument)..." << std::endl;  success[1] = CreateProcessA(  NULL,  (LPSTR)"Lab-03x.exe 6",  NULL, NULL, FALSE, 0,  NULL, NULL, &si[1], &pi[1]  );  // Третий процесс — NULL во втором параметре, локальное окружение  std::cout << "3. Creating process (NULL command line, local env var)..." << std::endl;  // локальная переменная окружения ITER\_NUM = 8  SetEnvironmentVariableA("ITER\_NUM", "8");  success[2] = CreateProcessA(  exe.c\_str(), // lpApplicationName  NULL, // lpCommandLine = NULL  NULL, NULL, FALSE, 0,  NULL, NULL, &si[2], &pi[2]  );  // --- Выводим результаты ---  for (int i = 0; i < 3; i++) {  std::cout << "Process " << (i + 1) << ": "  << (success[i] ? "SUCCESS" : "FAILED")  << ", PID: " << (success[i] ? pi[i].dwProcessId : 0)  << std::endl;  }  // --- Ждём завершения ---  std::cout << "\nWaiting for all processes..." << std::endl;  for (int i = 0; i < 3; i++) {  if (success[i])  WaitForSingleObject(pi[i].hProcess, INFINITE);  }  for (int i = 0; i < 3; i++) {  if (success[i]) {  CloseHandle(pi[i].hProcess);  CloseHandle(pi[i].hThread);  }  }  // Восстановим переменную окружения  SetEnvironmentVariableA("ITER\_NUM", NULL);  std::cout << "All processes completed!" << std::endl;  return 0;  } |

Листинг 2.1.3.1 – Исходный код программы Lab-03b.exe, использующей глобальную и локальную переменные окружения

Программа запустит три процесса Lab-03x.exe:

* без аргумента (использует глобальную ITER\_NUM),
* с аргументом 6,
* с локальной переменной окружения ITER\_NUM=8.

Все процессы завершены успешно, так как первый процесс теперь не содержит аргумента.

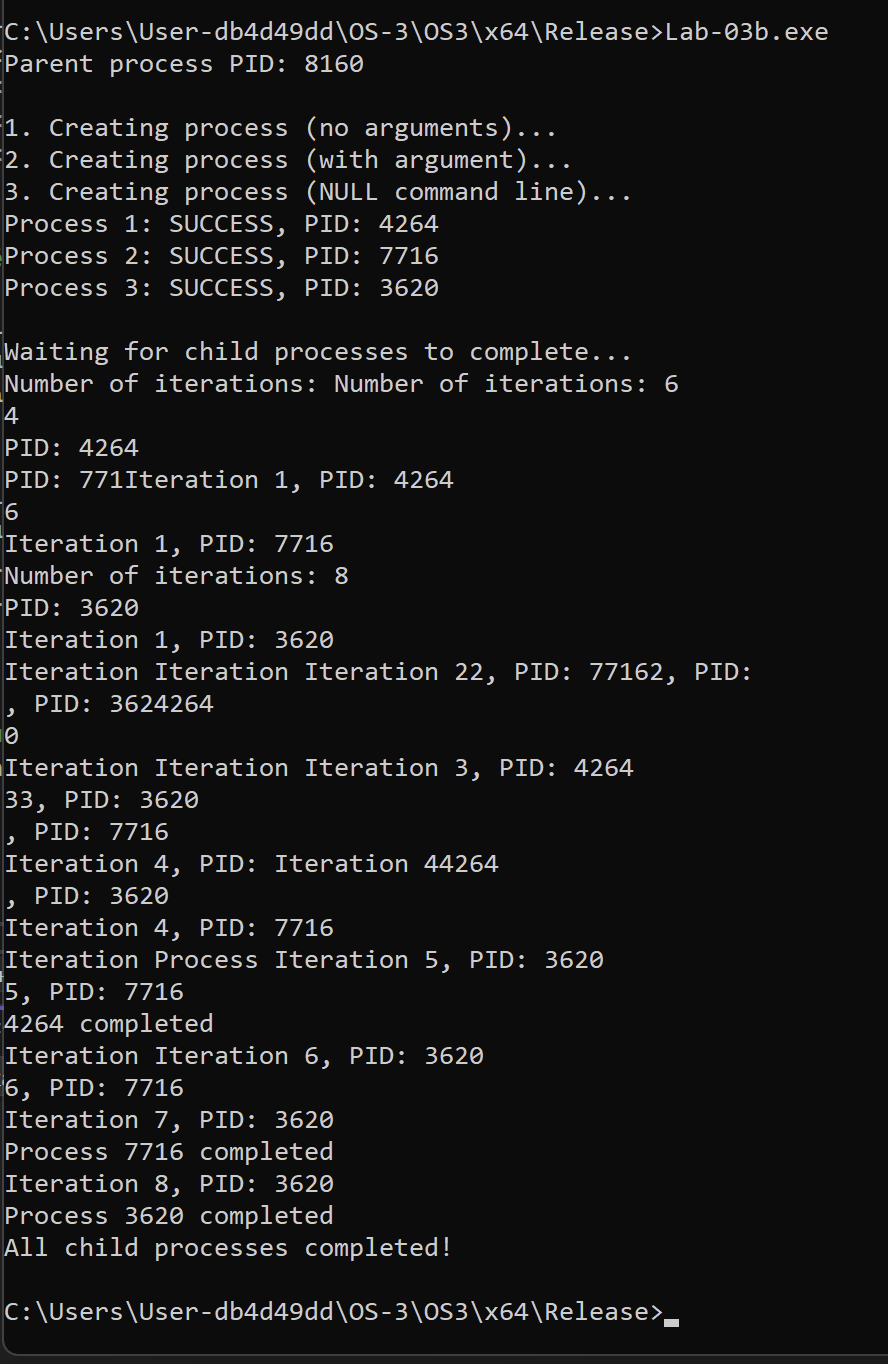


Рисунок 2.1.3.1 – Работа программы Lab-03b

### 2.4 Приложение Lab-03c

Программа выводит список всех запущенных в системе процессов,

включая их PID и родительский PID.

|  |
| --- |
| #include <windows.h>  #include <tlhelp32.h>  #include <iostream>  #include <iomanip>  int main() {  HANDLE hSnapshot = CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, 0);  if (hSnapshot == INVALID\_HANDLE\_VALUE) {  std::cerr << "Error creating process snapshot!" << std::endl;  return 1;  }  PROCESSENTRY32 pe;  pe.dwSize = sizeof(PROCESSENTRY32);  std::cout << "List of running processes:" << std::endl;  std::cout << std::left  << std::setw(30) << "Process Name"  << std::setw(10) << "PID"  << std::setw(10) << "Parent PID"  << std::endl;  std::cout << std::string(50, '-') << std::endl;  // Get first process  if (Process32First(hSnapshot, &pe)) {  do {  std::cout << std::left  << std::setw(30) << pe.szExeFile  << std::setw(10) << pe.th32ProcessID  << std::setw(10) << pe.th32ParentProcessID  << std::endl;  } while (Process32Next(hSnapshot, &pe));  }  CloseHandle(hSnapshot);  return 0;  } |

Листинг 2.1.3.1 – Программа Lab-03c.exe, выводящая список запущенных в системе процессов

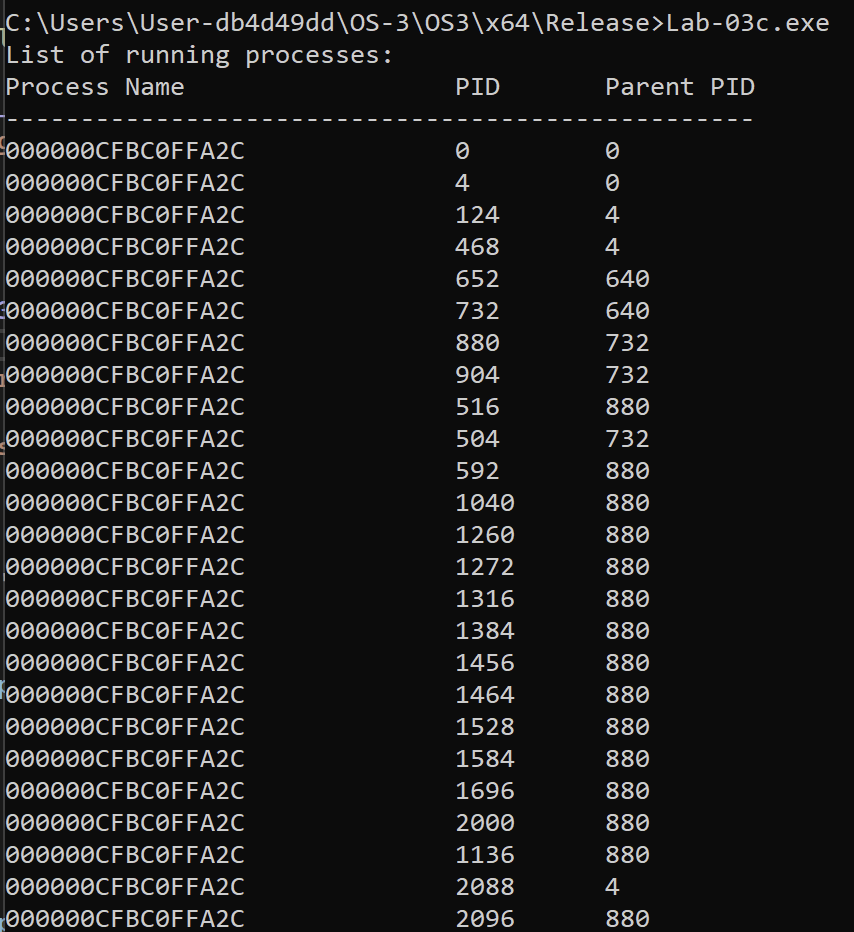


Рисунок 2.1.3.1 – Фрагмент списка запущенных процессов, полученного через Lab-03c.exe

## 2.2 Выполнение заданий для Linux

### 2.2.1 Приложение Lab-03x.exe

Программа является полным аналогом Windows-варианта. Она выполняет заданное количество итераций с задержкой 0.5 с, выводит PID и номер итерации. Количество итераций берётся из аргумента командной строки, а при его отсутствии — из переменной окружения ITER\_NUM. Если оба отсутствуют, программа завершает работу с сообщением об ошибке.

|  |
| --- |
| #include <iostream>  #include <unistd.h> // getpid(), usleep()  #include <cstdlib> // atoi(), getenv(), exit()  int main(int argc, char\* argv[]) {  int iterations = 0;  if (argc > 1) {  iterations = std::atoi(argv[1]);  } else {  char\* env\_var = std::getenv("ITER\_NUM");  if (env\_var != nullptr) {  iterations = std::atoi(env\_var);  }  }  if (iterations <= 0) {  std::cerr << "Error: number of iterations not specified!" << std::endl;  return 1;  }  std::cout << "Number of iterations: " << iterations << std::endl;  std::cout << "PID: " << getpid() << std::endl;  for (int i = 1; i <= iterations; i++) {  std::cout << "Iteration " << i << ", PID: " << getpid() << std::endl;  usleep(500000); // 500 мс = 0.5 секунды  }  std::cout << "Process " << getpid() << " completed" << std::endl;  return 0;  } |

Листинг 2.2.1.1 – Программа Lab-03x.exe

Теперь выполним программу без параметров командной строки:

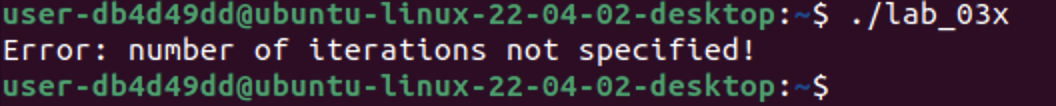


Рисунок 2.2.1.1 – Работа программы Lab-03x при отсутсвии параметров

Теперь выполним программу с параметром командной строки:

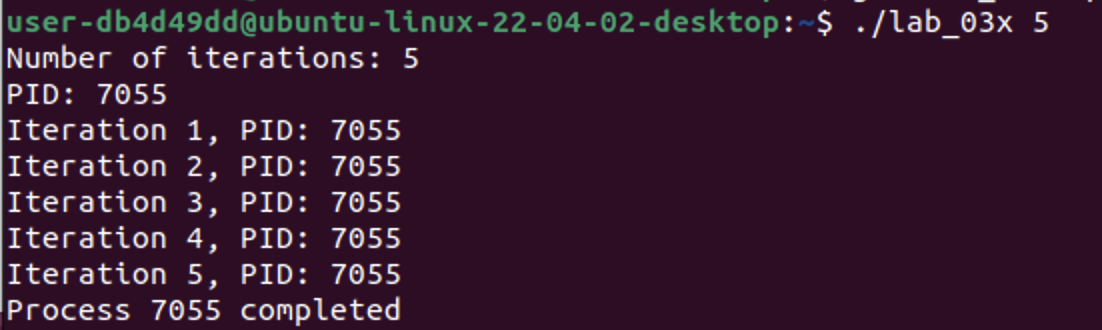


Рисунок 2.2.1.2 – Работа программы Lab-03x при передаче количества итераций через аргумент

Теперь выполним её без аргументов, но задав переменную окружения:

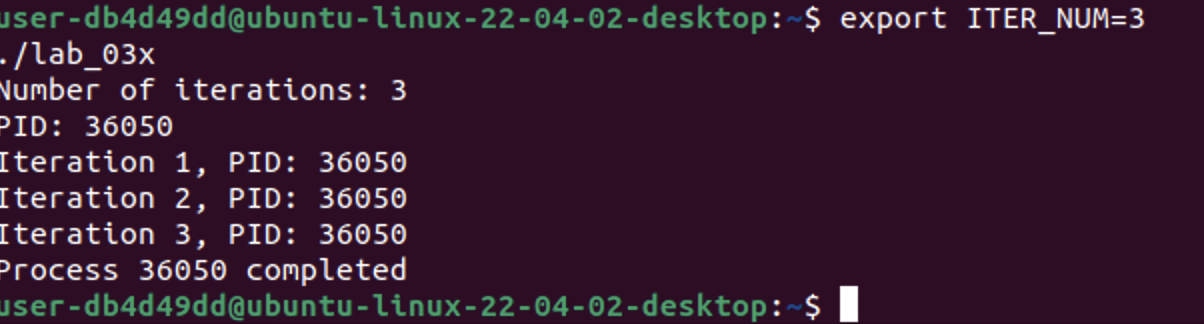


Рисунок 2.2.1.3 – Получение количества итераций из переменной окружения ITER\_NUM

### 2.2.2 Приложение Lab-03а.exe

Программа создаёт два дочерних процесса, загружая в них Lab-03x.exe.

* Первый процесс получает количество итераций параметром.
* Второй процесс получает количество итераций из переменной окружения ITER\_NUM, создаваемой непосредственно в родительском процессе перед запуском второго дочернего.
* Родительский процесс ждет завершения обоих дочерних процессов.

|  |
| --- |
| #include <iostream>  #include <unistd.h> // fork(), exec(), getpid()  #include <sys/wait.h> // waitpid()  #include <cstdlib> // setenv(), exit()  #include <cstring> // strerror()  int main() {  pid\_t pids[2];  std::cout << "Parent PID: " << getpid() << std::endl;  // ============================  // 1. Первый дочерний процесс  // Аргумент через командную строку  // ============================  pids[0] = fork();  if (pids[0] == -1) {  std::cerr << "Error creating first process: " << strerror(errno) << std::endl;  return 1;  }  if (pids[0] == 0) {  // Дочерний процесс 1  std::cout << "Child 1 started, PID: " << getpid() << std::endl;  char \*args[] = { (char\*)"./lab\_03x", (char\*)"4", nullptr };  execvp(args[0], args);  std::cerr << "Failed to start child 1: " << strerror(errno) << std::endl;  exit(1);  }  // ============================  // 2. Второй дочерний процесс  // Параметр через переменную окружения ITER\_NUM  // ============================  pids[1] = fork();  if (pids[1] == -1) {  std::cerr << "Error creating second process: " << strerror(errno) << std::endl;  return 1;  }  if (pids[1] == 0) {  std::cout << "Child 2 started, PID: " << getpid() << std::endl;  // Локальная установка переменной окружения (только для этого процесса)  setenv("ITER\_NUM", "6", 1);  char \*args[] = { (char\*)"./lab\_03x", nullptr };  execvp(args[0], args);  std::cerr << "Failed to start child 2: " << strerror(errno) << std::endl;  exit(1);  }  // ============================  // Родительский процесс  // ============================  std::cout << "Waiting for child processes to finish..." << std::endl;  for (int i = 0; i < 2; i++) {  int status;  pid\_t finished = waitpid(pids[i], &status, 0);  if (finished == -1) {  std::cerr << "Error waiting for child " << i+1 << ": " << strerror(errno) << std::endl;  } else if (WIFEXITED(status)) {  std::cout << "Child " << i+1 << " (PID " << finished  << ") exited with code " << WEXITSTATUS(status) << std::endl;  } else {  std::cout << "Child " << i+1 << " (PID " << finished  << ") terminated abnormally" << std::endl;  }  }  std::cout << "All child processes completed!" << std::endl;  return 0;  } |

Листинг 2.2.2.1 – Исходный код программы Lab-03a.exe, создающей два дочерних процесса и передающей параметры разными способами

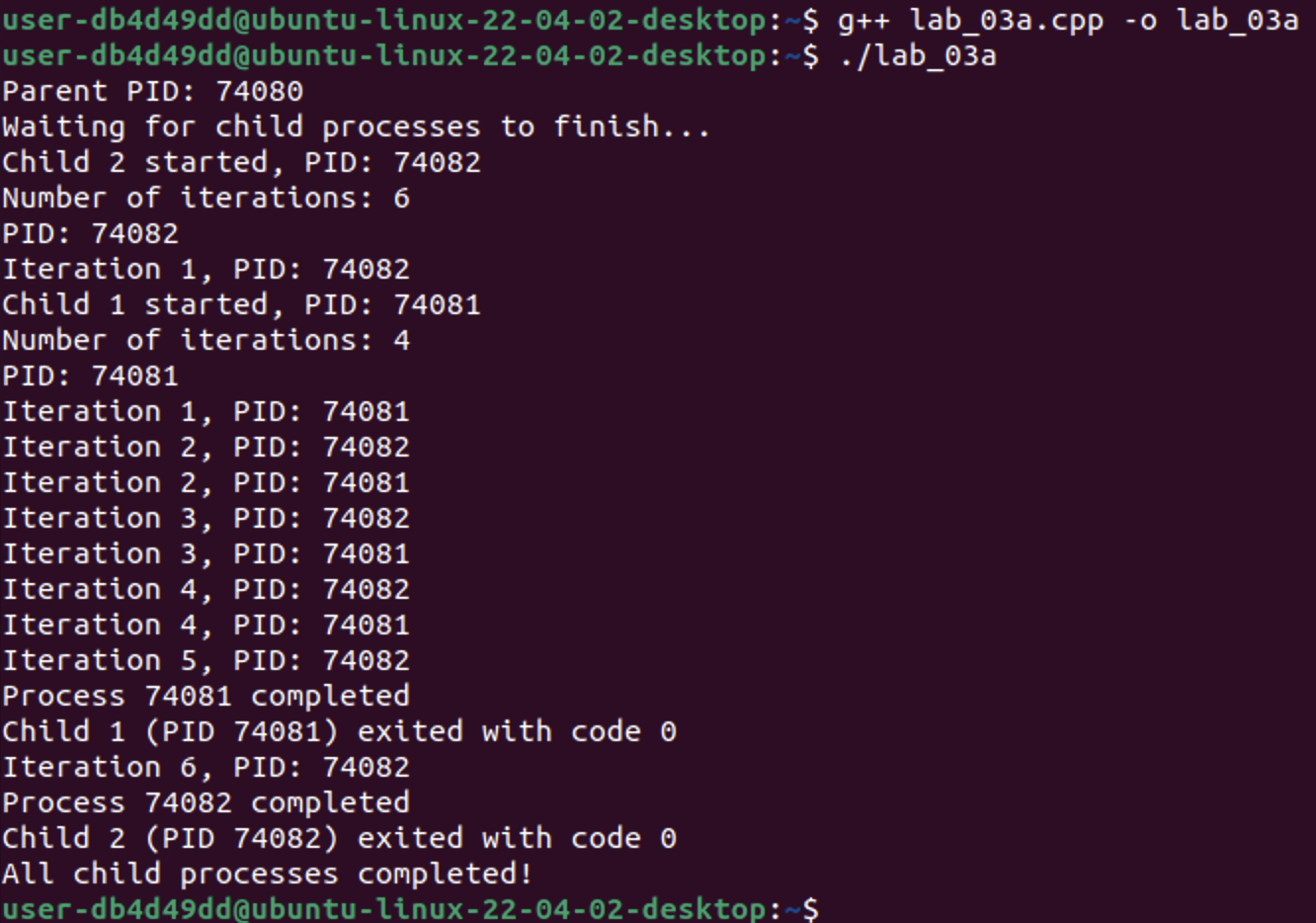


Рисунок 2.2.2.1 – дновременная работа двух дочерних процессов Lab-03a.exe

### 2.2.3 Проверка процессов через утилиты Linux

После запуска можно проверить активные процессы и их PID следующими командами:

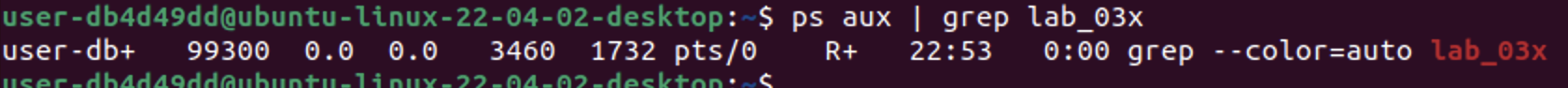


Рисунок 2.2.3.1 – Подтверждение запуска процессов через ps

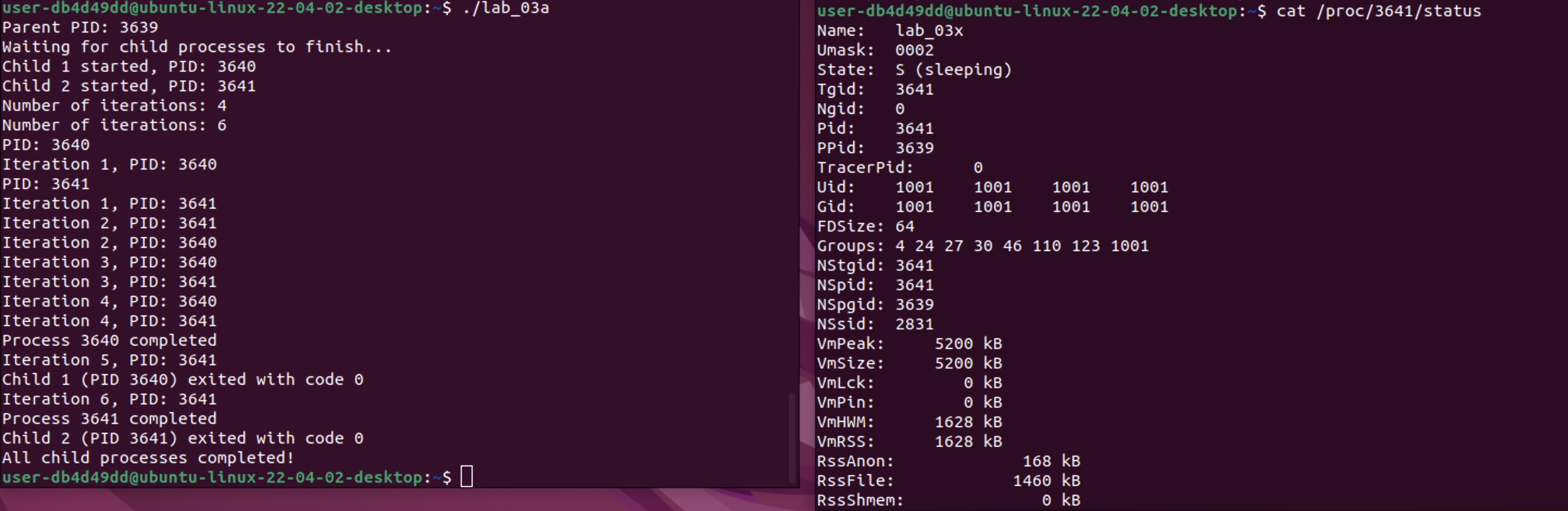


Рисунок 2.2.3.2 – Подтверждение запуска процессов через /proc

Все процессы созданы успешно. Оба дочерних процесса выполняются параллельно и завершаются корректно. Первый получает значение итераций через аргумент, второй — из переменной окружения ITER\_NUM. Ошибок при создании и завершении процессов не возникло, что подтверждается выводом программы и анализом через ps и /proc.

3 Заключение

В ходе работы освоены процессы в Windows и Linux. Обе ОС используют схожие концепции, но разный API. Windows: CreateProcess, ExitProcess, WaitForSingleObject. Linux: fork/exec, /proc. Изучены создание процессов, передача параметров (аргументы, переменные окружения), ожидание завершения и анализ процессов. Все программы работают корректно, процессы создаются успешно. Получен практический опыт работы с процессами в разных ОС.