Белорусский государственный технологический университет
Факультет информационных технологий
Кафедра программной инженерии

Лабораторная работа 4

По дисциплине «Операционные системы»

На тему «Управление потоками

Выполнил:
Студент 3 курса 9 группы
Павлович Ян Андреевич
Преподаватель: Савельева М.Г.

Минск, 2025

Введение

В рамках данной лабораторной работы предстоит освоить практические аспекты параллельного программирования на уровне потоков (нитей) выполнения в операционных системах Windows и Linux.

Будет сделан на фундаментальных операциях управления потоками. В ходе работы предстоит:

1. Разработать серию приложений для Windows, которые будут:
   - Создавать и контролировать выполнение дочерних потоков.
   - Реализовывать механизмы управления их жизненным циклом: целенаправленную приостановку и возобновление выполнения, а также принудительное завершение.
   - Наглядно демонстрировать работу потоков с помощью вывода их идентификаторов и циклического отображения символов из имени пользователя.
2. Изучить состояния созданных потоков и их поведение в реальном времени с помощью системных инструментов Process Explorer и PowerShell.
3. Реализовать аналогичную базовую логику работы с потоками в среде Linux, используя стандарт POSIX Threads (pthreads).
4. Освоить анализ информации о потоках в Linux через файловую систему /proc и утилиту ps.

Ключевой задачей является сравнение архитектурных подходов и API двух операционных систем для решения одной и той же задачи параллельного выполнения кода.

Инструментальная база:

- Среда разработки: Visual Studio, MinGW-w64
- Системное ПО: Parallels Desktop
- Командные оболочки: cmd, PowerShell
- Утилиты мониторинга: Sysinternals Process Explorer, ps

1 Windows

1.1 Приложение Lab-04x

Для работы необходимо реализовать функцию, которая в цикле выполняет заданное число повторов. Каждый шаг цикла должен сопровождаться выводом в консоль четырех элементов: идентификатора текущего процесса (PID), идентификатора текущего потока (TID), порядкового номера текущей итерации, а также одного символа из имени пользователя. Количество итераций передается в функцию в качестве входного параметра. После каждой итерации выполняется пауза длительностью от 300 до 400 миллисекунд. В коде необходимо обеспечить обработку возможных ошибок и освобождение всех используемых ресурсов.

Для вывода информации следует использовать функции семейства printf. Формат строки вывода должен быть следующим: сначала выводится PID, затем TID, далее номер итерации, и в конце — очередная буква из имени пользователя. Символы из имени пользователя выводятся по кругу: после последней буквы снова выводится первая и так далее.

```
C:\lab4os\x64\Debug>C:\lab4os\x64\Debug\Lab_04x.exe 14
PID: 12092 - TID: 5040 - Iteration: 1 - Letter: U
PID: 12092 - TID: 5040 - Iteration: 2 - Letter: s
PID: 12092 - TID: 5040 - Iteration: 3 - Letter: e
PID: 12092 - TID: 5040 - Iteration: 4 - Letter: r
PID: 12092 - TID: 5040 - Iteration: 5 - Letter: -
PID: 12092 - TID: 5040 - Iteration: 6 - Letter: 9
PID: 12092 - TID: 5040 - Iteration: 7 - Letter: e
PID: 12092 - TID: 5040 - Iteration: 8 - Letter: 5
PID: 12092 - TID: 5040 - Iteration: 9 - Letter: b
PID: 12092 - TID: 5040 - Iteration: 10 - Letter: b
PID: 12092 - TID: 5040 - Iteration: 11 - Letter: 2
PID: 12092 - TID: 5040 - Iteration: 12 - Letter: d
PID: 12092 - TID: 5040 - Iteration: 13 - Letter: 1
PID: 12092 - TID: 5040 - Iteration: 14 - Letter: 9
```

Рисунок 1.1 – Работа Lab_04x

Был создан текстовый файл с расширением .cpp, в который помещен исходный код функции (см. листинг А.1). После компиляции и запуска полученной программы ее работа была проверена. Результат выполнения (рисунок 1.1) подтверждает, что функция работает в соответствии с поставленными требованиями.

1.2 Приложение Lab-04a:

Требуется разработать приложение, которое создает два дополнительных потока. В каждом из этих потоков должна выполняться ранее реализованная функция Lab_04x с разным количеством итераций. Параллельно с этим в основном (главном) потоке программы также вызывается функция Lab_04x. Главный поток обязан дождаться полного завершения работы обоих созданных им дочерних потоков, прежде чем завершить свою собственную работу.

```
C:\lab4os\x64\Debug>C:\lab4os\x64\Debug\Lab-04a.exe
Thread TID: 1908 started
PID: 10684 - TID: 1908 - Iteration: 1 - Letter: U
Thread TID: 4360 started
PID: 10684 - TID: 4360 - Iteration: 1 - Letter: U
Thread TID: 4440 started
PID: 10684 - TID: 4440 - Iteration: 1 - Letter: U
PID: 10684 - TID: 1908 - Iteration: 2 - Letter: s
PID: 10684 - TID: 4440 - Iteration: 2 - Letter: s
PID: 10684 - TID: 4360 - Iteration: 2 - Letter: s
PID: 10684 - TID: 1908 - Iteration: 3 - Letter: e
PID: 10684 - TID: 4440 - Iteration: 3 - Letter: e
PID: 10684 - TID: 4360 - Iteration: 3 - Letter: e
PID: 10684 - TID: 1908 - Iteration: 4 - Letter: r
PID: 10684 - TID: 4440 - Iteration: 4 - Letter: r
PID: 10684 - TID: 4360 - Iteration: 4 - Letter: r
PID: 10684 - TID: 4440 - Iteration: 5 - Letter: -
PID: 10684 - TID: 1908 - Iteration: 5 - Letter: -
PID: 10684 - TID: 4360 - Iteration: 5 - Letter: -
PID: 10684 - TID: 1908 - Iteration: 6 - Letter: 9
PID: 10684 - TID: 4440 - Iteration: 6 - Letter: 9
PID: 10684 - TID: 4360 - Iteration: 6 - Letter: 9
PID: 10684 - TID: 1908 - Iteration: 7 - Letter: e
PID: 10684 - TID: 4360 - Iteration: 7 - Letter: e
PID: 10684 - TID: 4440 - Iteration: 7 - Letter: e
PID: 10684 - TID: 1908 - Iteration: 8 - Letter: 5
PID: 10684 - TID: 4360 - Iteration: 8 - Letter: 5
PID: 10684 - TID: 4440 - Iteration: 8 - Letter: 5
PID: 10684 - TID: 1908 - Iteration: 9 - Letter: b
PID: 10684 - TID: 4360 - Iteration: 9 - Letter: b
PID: 10684 - TID: 4440 - Iteration: 9 - Letter: b
```

Рисунок 1.2 – Работа Lab_04x

Исходный код приложения был написан на C++ и сохранен в файл с расширением .cpp (листинг Б.1). После компиляции программа была запущена, что иллюстрирует рисунок 1.2.

```
PID: 10684 - TID: 1908 - Iteration: 50 - Letter: 5
PID: 10684 - TID: 4360 - Iteration: 51 - Letter: b
Thread TID: 1908 finished
PID: 10684 - TID: 4440 - Iteration: 51 - Letter: b
PID: 10684 - TID: 4360 - Iteration: 52 - Letter: b
```

Рисунок 1.3 – Завершение первого дочернего процесса

В момент работы приложения одновременно выполняются три потока. Каждый из них независимо выполняет цикл с назначенным ему количеством итераций, что видно по выводу в консоли. На рисунке 1.3 показан процесс завершения дочернего процесса.

```
PID: 10684 - TID: 4440 - Iteration: 100 - Letter: s
PID: 10684 - TID: 4360 - Iteration: 100 - Letter: s
Main thread TID: 4440 finished iterations, waiting for child threads...
PID: 10684 - TID: 4360 - Iteration: 101 - Letter: e
PID: 10684 - TID: 4360 - Iteration: 102 - Letter: r
```

Рисунок 1.4 – Завершение главного процесса с ожиданием

На рисунке 1.4 показан момент, когда цикл в главном потоке завершил свои итерации. Однако, как и требуется, главный поток не завершается сразу, а переходит в состояние ожидания, останавливаясь на функции синхронизации (например, WaitForMultipleObjects) до окончания работы дочерних потоков.

```
PID: 10684 - TID: 4360 - Iteration: 122 - Letter: b
PID: 10684 - TID: 4360 - Iteration: 123 - Letter: 2
PID: 10684 - TID: 4360 - Iteration: 124 - Letter: d
PID: 10684 - TID: 4360 - Iteration: 125 - Letter: 1
Thread TID: 4360 finished
Thread TID: 4440 finished
All threads finished. Program completed.
```

Рисунок 1.5 – Завершение второго дочернего процесса и главного

Рисунок 1.5 демонстрирует завершение второго дочернего потока после отработки его цикла, вместе с этим, когда все дочерние потоки завершают свои задачи, главный поток также завершает ожидание и работу, после чего приложение закрывается.

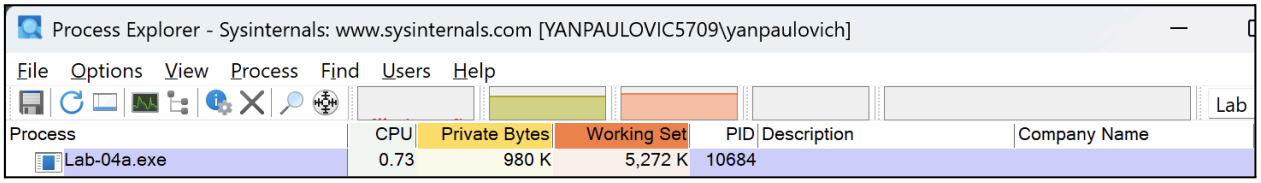

Рисунок 1.6 – Процесс в Process Explorer

Для наблюдения за потоками в реальном времени использовался Process Explorer (PE). На рисунке 1.6 виден поток в рамках исследуемого процесса. Вкладка "Threads" в свойствах процесса (рисунок 1.7) позволяет детально увидеть все созданные потоки, их идентификаторы (TID) и другую служебную информацию.

```
PS C:\Users\yanpaulovich> Get-Process -Name Lab-04a | Select-Object -ExpandProperty Threads


BasePriority            : 8
CurrentPriority         : 9
Id                      : 1692
IdealProcessor          :
PriorityBoostEnabled    : True
PriorityLevel           : Normal
PrivilegedProcessorTime : 00:00:00.0156250
StartAddress            : 140720624026896
StartTime               : 12/2/2025 4:33:35 PM
ThreadState             : Wait
TotalProcessorTime      : 00:00:00.0156250
UserProcessorTime       : 00:00:00
WaitReason              : UserRequest
ProcessorAffinity       :
Site                    :
Container               :

BasePriority            : 8
CurrentPriority         : 9
Id                      : 12000
IdealProcessor          :
PriorityBoostEnabled    : True
PriorityLevel           : Normal
PrivilegedProcessorTime : 00:00:00
StartAddress            : 140720624026896
StartTime               : 12/2/2025 4:33:35 PM
ThreadState             : Wait
TotalProcessorTime      : 00:00:00
UserProcessorTime       : 00:00:00
WaitReason              : EventPairLow
ProcessorAffinity       :
Site                    :
Container               :

BasePriority            : 8
CurrentPriority         : 9
Id                      : 13248
IdealProcessor          :
PriorityBoostEnabled    : True
```

Рисунок 1.7 – Процесс через Threads

Аналогичную проверку можно выполнить с помощью PowerShell, используя соответствующие командлеты для получения информации о потоках. В выводе PowerShell отображается полный список потоков процесса, включая их идентификаторы, время запуска, текущее состояние (например,

Running или Wait) и установленный приоритет. Это подтверждает, что в процессе действительно было создано и работало три потока, включая основной.

1.3 Приложение Lab-04b

Далее разрабатывается приложение Lab-04b. Его основа аналогична предыдущей программе Lab-04a, но с важными изменениями в логике управления потоками. Во-первых, прямой вызов функции Lab_04x в главном потоке заменен на аналогичный цикл, реализованный непосредственно в его коде. Во-вторых, добавлено активное управление дочерними потоками: работа первого потока должна быть приостановлена (suspend) на 20-й итерации цикла главного потока, а затем возобновлена (resume) на 60-й итерации. Работа второго потока приостанавливается на 40-й итерации главного цикла, а его возобновление происходит только после того, как сам главный цикл полностью завершит все свои итерации. Как и прежде, главный поток обязан ожидать окончательного завершения всех дочерних потоков.

```
C:\lab4os\x64\Debug>C:\lab4os\x64\Debug\Lab-04b.exe
Main thread TID: 4416 started
PID: 4436 - TID: 4416 - Iteration: 1 - Letter: U
Thread TID: 10568 started
PID: 4436 - TID: 10568 - Iteration: 1 - Letter: U
Thread TID: 10680 started
PID: 4436 - TID: 10680 - Iteration: 1 - Letter: U
PID: 4436 - TID: 4416 - Iteration: 2 - Letter: s
PID: 4436 - TID: 10568 - Iteration: 2 - Letter: s
PID: 4436 - TID: 10680 - Iteration: 2 - Letter: s
```

Рисунок 1.8 – Работа Lab-04b

Исходный код был написан и сохранен (листинг В.1), после чего скомпилирован и запущен. Результат запуска показан на рисунке 1.8.

```
PID: 4436 - TID: 10568 - Iteration: 20 - Letter: 9
PID: 4436 - TID: 10680 - Iteration: 20 - Letter: 9
Main thread suspended thread 1 at iteration 21
PID: 4436 - TID: 4416 - Iteration: 21 - Letter: e
PID: 4436 - TID: 10568 - Iteration: 21 - Letter: e
```

Рисунок 1.9 – Приостановка на 20 итерации

После старта программы начинают работу три потока. Каждый выполняет свой цикл. В соответствии с алгоритмом, после 20-й итерации главного потока первый дочерний поток успешно приостанавливается, что видно по его последнему выводу на 21-й итерации (рисунок 1.9).

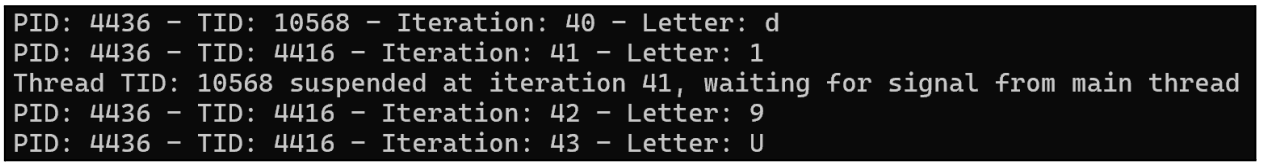

Рисунок 1.10 – Приостановка на 40 итерации

Далее, когда главный поток достигает 40-й итерации, приостанавливается выполнение второго дочернего потока (рисунок 1.10). По достижении главным потоком 60-й итерации он возобновляет работу первого потока (рисунок 1.11). Первый поток, получив управление, дорабатывает свой цикл до конца и завершается.

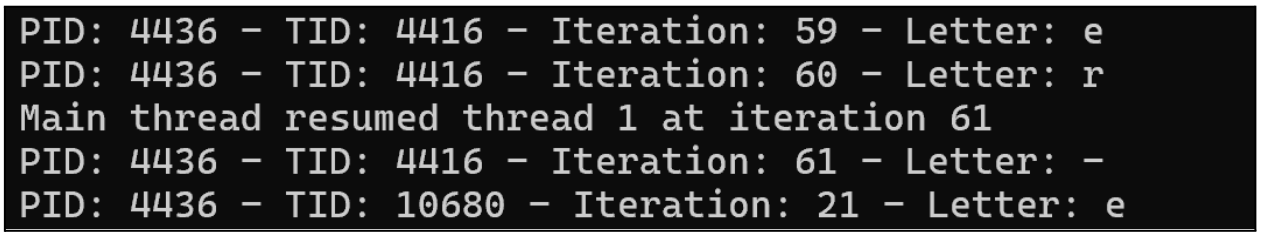

Рисунок 1.11 – Возобновление первого потока

Главный поток продолжает выполнение до своей 100-й итерации. На этом шаге он посылает сигнал на возобновление второму потоку, после чего завершает свой цикл и переходит в режим ожидания завершения оставшихся потоков.

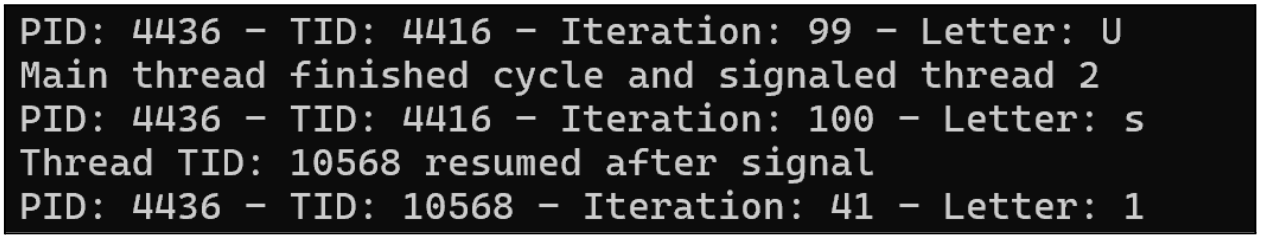

Рисунок 1.12 – Возобновление второго потока

Второй поток возобновляет выполнение с того места, где был остановлен (рисунок 1.12), дорабатывает свои итерации и завершается, после чего завершается и вся программа (рисунок 1.13).

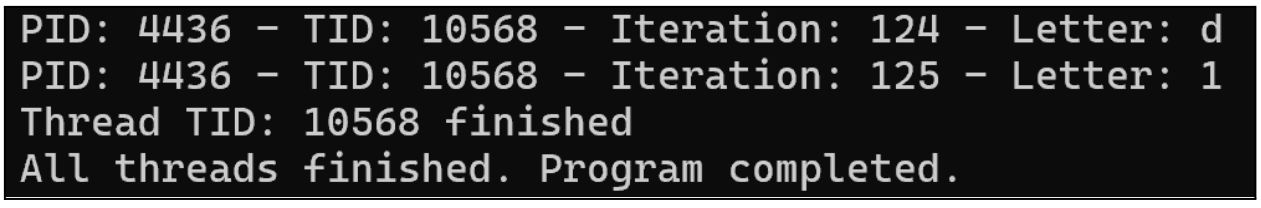

Рисунок 1.13 – Завершение работы программы

Для наблюдения за состояниями потоков использовался Process Explorer (рисунки 1.14) и Threads (рисунок 1.15).

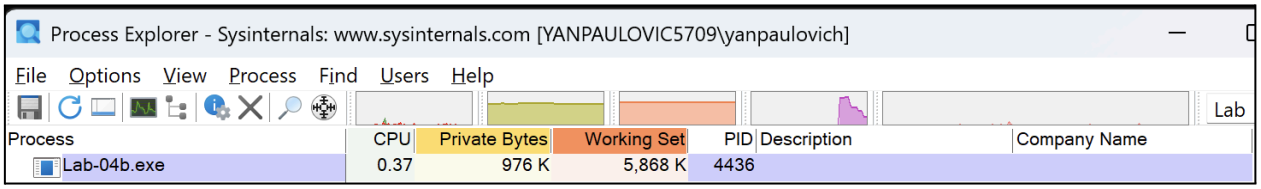

Рисунок 1.14 – Процесс через Process Explorer

В ходе наблюдений было подтверждено, что приостановленный поток, ожидающий сигнала от другого (например, с помощью события или мьютекса), в PE отображается с состоянием "Wait: UserRequest".

```
PS C:\Users\yanpaulovich> Get-Process -Name Lab-04b | Select-Object -ExpandProperty Threads


BasePriority             : 8
CurrentPriority          : 8
Id                       : 4416
IdealProcessor           :
PriorityBoostEnabled     : True
PriorityLevel            : Normal
PrivilegedProcessorTime  : 00:00:00.0468750
StartAddress             : 140720624026896
StartTime                : 12/2/2025 5:01:32 PM
ThreadState              : Wait
TotalProcessorTime       : 00:00:00.0625000
UserProcessorTime        : 00:00:00.0156250
WaitReason               : ExecutionDelay
ProcessorAffinity        :
Site                     :
Container                :

BasePriority             : 8
CurrentPriority          : 9
Id                       : 12436
IdealProcessor           :
PriorityBoostEnabled     : True
PriorityLevel            : Normal
PrivilegedProcessorTime  : 00:00:00
StartAddress             : 140720624026896
StartTime                : 12/2/2025 5:01:32 PM
ThreadState              : Wait
TotalProcessorTime       : 00:00:00
UserProcessorTime        : 00:00:00
WaitReason               : EventPairLow
ProcessorAffinity        :
Site                     :
Container                :

BasePriority             : 8
CurrentPriority          : 9
Id                       : 6344
IdealProcessor           :
PriorityBoostEnabled     : True
```

Рисунок 1.15 – Процесс через Threads

Если же поток находится в ожидании из-за вызова функции задержки, такой как Sleep. Это позволяет наглядно изучать изменения состояния потоков на разных этапах выполнения программы.

1.4 Приложение Lab-04c

Следующим этапом является разработка приложения Lab-04c. Его структура базируется на Lab-04a, но содержит ключевое изменение в логике. Вызов функции Lab_04x в главном потоке снова заменен на аналогичный встроенный цикл. Внутри этого цикла, по достижении 40-й итерации, программа инициирует принудительное завершение второго дочернего потока. Главный поток, как и в предыдущих заданиях, должен дождаться завершения всех созданных им дочерних потоков (включая тот, который был завершен досрочно) перед своим окончанием.

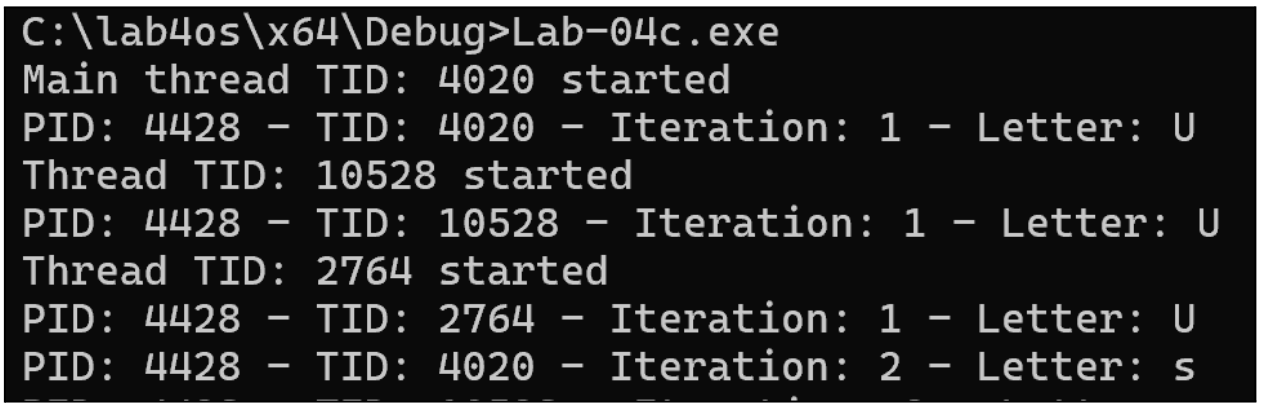

Рисунок 1.16 – Работа Lab-04c

Исходный код был написан, сохранен в файл .cpp (листинг Г.1), скомпилирован и запущен. При запуске приложения, как и ожидалось, создаются и начинают работу три потока: главный и два дочерних (рисунок 1.16).

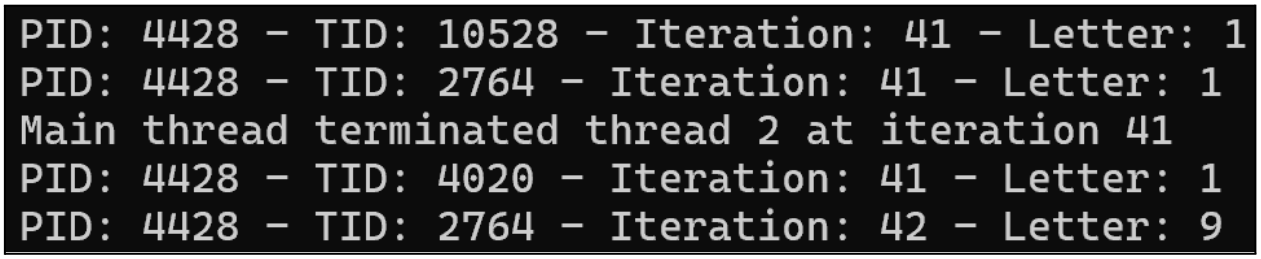

Рисунок 1.17 – Вызываем остановку потока

После выполнения 40-й итерации главный поток вызывает функцию для завершения второго потока (рисунок 1.17).

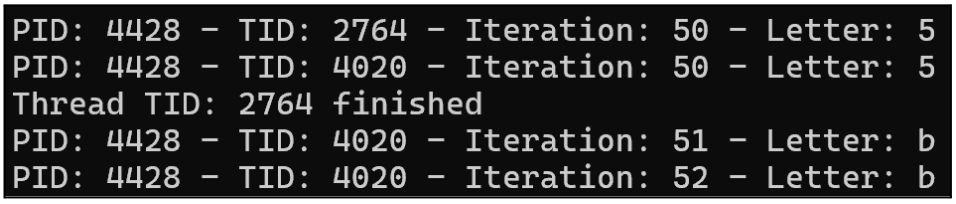

Рисунок 1.18 – Конец работы потока

Далее программа продолжает работу: первый дочерний поток и главный поток выполняют свои циклы до конца (рисунки 1.18 и 1.19).

```
PID: 4428 - TID: 4020 - Iteration: 98 - Letter: 9
PID: 4428 - TID: 4020 - Iteration: 99 - Letter: U
PID: 4428 - TID: 4020 - Iteration: 100 - Letter: s
All threads finished. Program completed.
```

Рисунок 1.19 – Завершение программы

После полного завершения работы приложения, при анализе через Process Explorer (PE) в списке потоков процесса отображается только два потока (рисунок 1.20). Это связано с тем, что один из потоков был завершен в середине работы.

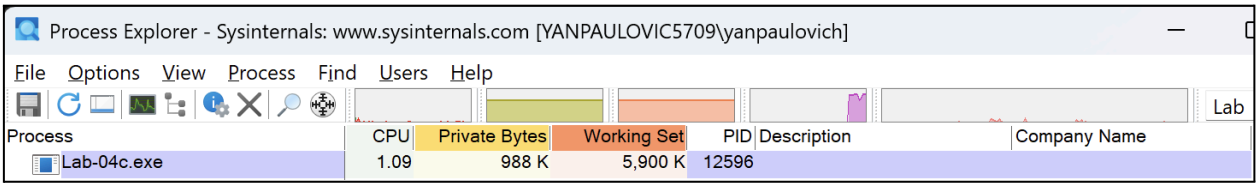

Рисунок 1.20 – Процесс через Process Explorer

Для более детального анализа использовался PowerShell. История или снимок системы, сделанный во время работы приложения, показывает, что изначально потоков действительно было три (рисунок 1.17). Однако после завершения второго потока его дескриптор в системе был очищен, и ядро ОС освободило связанные с ним ресурсы.

```
PS C:\Users\yanpaulovich> Get-Process -Name Lab-04c | Select-Object -ExpandProperty Threads

BasePriority            : 8
CurrentPriority         : 8
Id                      : 12320
IdealProcessor          :
PriorityBoostEnabled    : True
PriorityLevel           : Normal
PrivilegedProcessorTime : 00:00:00.0312500
StartAddress            : 140720624026896
StartTime               : 12/2/2025 5:41:32 PM
ThreadState             : Wait
TotalProcessorTime      : 00:00:00.0468750
UserProcessorTime       : 00:00:00.0156250
WaitReason              : ExecutionDelay
ProcessorAffinity       :
Site                    :
Container               :

BasePriority            : 8
CurrentPriority         : 9
Id                      : 980
IdealProcessor          :
PriorityBoostEnabled    : True
PriorityLevel           : Normal
PrivilegedProcessorTime : 00:00:00
StartAddress            : 140720624026896
StartTime               : 12/2/2025 5:41:32 PM
ThreadState             : Wait
TotalProcessorTime      : 00:00:00
UserProcessorTime       : 00:00:00
WaitReason              : EventPairLow
ProcessorAffinity       :
Site                    :
Container               :

BasePriority            : 8
CurrentPriority         : 9
Id                      : 11912
IdealProcessor          :
PriorityBoostEnabled    : True
```

Рисунок 1.21 – Процесс через Threads

Именно поэтому в Process Explorer, который отображает текущее состояние системы после остановки программы, завершенный поток больше не виден. Это подтверждает, что дескриптор потока после его корректного (или принудительного) завершения удаляется из системы, и память освобождается.

1.5 Приложение Lab-04d

Создание утилиты Lab-04d. Ее цель — получение и вывод списка всех потоков, выполняющихся в контексте текущего процесса. Для каждого такого потока программа должна отобразить его уникальный идентификатор (TID - Thread Identifier). Реализация данной задачи выполняется с использованием Tool Help Library — специального набора функций WinAPI, предназначенных для создания снимков состояния системы (snapshot), включая информацию о процессах, потоках и загруженных модулях.

Исходный код программы был написан на C++ и сохранен (листинг Д.1). После компиляции и запуска приложение выводит в консоль список идентификаторов потоков. Пример результата работы представлен на рисунке 1.22.

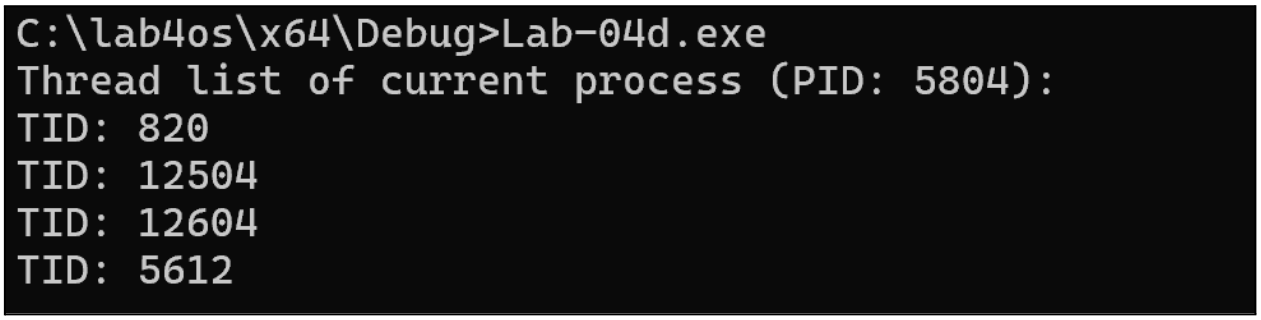

Рисунок 1.22 – Работа Lab-04d

В выводе программы первой строкой, как правило, отображается идентификатор самого процесса (PID — Process Identifier), в контексте которого работает утилита. Далее следует перечисление идентификаторов всех потоков (TID), принадлежащих этому процессу. Данная утилита наглядно демонстрирует механизм перечисления системных объектов с помощью Tool Help Library и позволяет убедиться в том, что даже сама программа, будучи однопоточной на уровне логики, в действительности может состоять из нескольких потоков, созданных системой или рантайм-библиотеками для служебных целей.

2 Linux

2.1 Приложение Lab-04x

Для работы необходимо реализовать функцию, которая в цикле выполняет заданное число повторов. Каждый шаг цикла должен сопровождаться выводом в консоль четырех элементов: идентификатора текущего процесса (PID), идентификатора текущего потока (TID), порядкового номера текущей итерации, а также одного символа из имени пользователя. Количество итераций передается в функцию в качестве входного параметра.

```
parallels@ubuntu-linux-22-04-02-desktop:~$ nano lab-04x.cpp
parallels@ubuntu-linux-22-04-02-desktop:~$ g++ lab-04x.cpp -o lab-04x
parallels@ubuntu-linux-22-04-02-desktop:~$ ./lab-04x 14
PID: 17943 - TID: 17944 - Iteration: 1 - Letter: U
PID: 17943 - TID: 17944 - Iteration: 2 - Letter: s
PID: 17943 - TID: 17944 - Iteration: 3 - Letter: e
PID: 17943 - TID: 17944 - Iteration: 4 - Letter: r
PID: 17943 - TID: 17944 - Iteration: 5 - Letter: -
PID: 17943 - TID: 17944 - Iteration: 6 - Letter: 9
PID: 17943 - TID: 17944 - Iteration: 7 - Letter: e
PID: 17943 - TID: 17944 - Iteration: 8 - Letter: 5
PID: 17943 - TID: 17944 - Iteration: 9 - Letter: b
PID: 17943 - TID: 17944 - Iteration: 10 - Letter: b
PID: 17943 - TID: 17944 - Iteration: 11 - Letter: 2
PID: 17943 - TID: 17944 - Iteration: 12 - Letter: d
PID: 17943 - TID: 17944 - Iteration: 13 - Letter: 1
PID: 17943 - TID: 17944 - Iteration: 14 - Letter: 9
parallels@ubuntu-linux-22-04-02-desktop:~$
```

Рисунок 2.1 – Работа Lab_04x

Был создан текстовый файл с расширением .cpp, в который помещен исходный код функции (см. листинг Е.1). После компиляции и запуска полученной программы ее работа была проверена. Результат выполнения (рисунок 2.1) подтверждает, что функция работает в соответствии с поставленными требованиями.

2.2 Приложение Lab-04px

Разработано приложение Lab-04px для Linux. Его задача аналогична Windows-версии: создать два дочерних потока с помощью pthreads, в которых выполняется функция Lab_04x на 50 и 125 итераций, и одновременно запустить эту же функцию в главном потоке на 100 итераций. Главный поток обязан ожидать завершения дочерних потоков с помощью pthread_join.

```
parallels@ubuntu-linux-22-04-02-desktop:~$ nano lab-04px.cpp
parallels@ubuntu-linux-22-04-02-desktop:~$ g++ lab-04px.cpp -0 lab-04px
g++: error: unrecognized command-line option '-0'
parallels@ubuntu-linux-22-04-02-desktop:~$ g++ lab-04px.cpp -o lab-04px
parallels@ubuntu-linux-22-04-02-desktop:~$ ./lab-04px
Главный поток TID: 140273471476672 начал работу
PID: 18879 - TID: 140273471476672 - №1 - U
Поток TID: 140273449432640 начал работу
PID: 18879 - TID: 140273449432640 - №1 - U
Поток TID: 140273441039936 начал работу
PID: 18879 - TID: 140273441039936 - №1 - U
PID: 18879 - TID: 140273471476672 - №2 - s
```

Рисунок 2.2 – Работа Lab_04x

Основная логика приложения с созданием и ожиданием потоков находится в файле Lab_04px.cpp (листинг Ж.1). После компиляции программа была запущена. На рисунке 2.2 показано начало ее работы с параллельным выводом от трех потоков. На рисунке 2.3 показано завершение одного из потоков.

```
PID: 18879 - TID: 140273449432640 - №50 - 1
PID: 18879 - TID: 140273471476672 - №50 - 1
PID: 18879 - TID: 140273441039936 - №50 - 1
Поток TID: 140273449432640 завершил работу
PID: 18879 - TID: 140273471476672 - №51 - a
```

Рисунок 2.3 – Завершение дочернего потока

На рисунках 2.4 показано завершение главного потока и ожидание выполнения дочерних.

```
PID: 18879 - TID: 140273471476672 - №100 - 8
PID: 18879 - TID: 140273441039936 - №100 - 8
Главный поток завершил итерации, ждёт дочерние потоки...
PID: 18879 - TID: 140273441039936 - №101 - 7
PID: 18879 - TID: 140273441039936 - №102 - 1
```

Рисунок 2.4 – Завершение главного потока

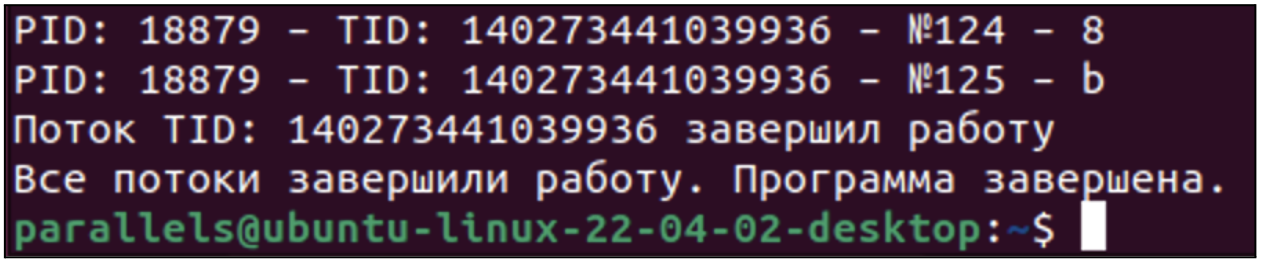

Рисунок 2.5 – Завершение программы Lab_04px

Рисунок 2.5 демонстрирует корректное завершение всех потоков и самой программы.

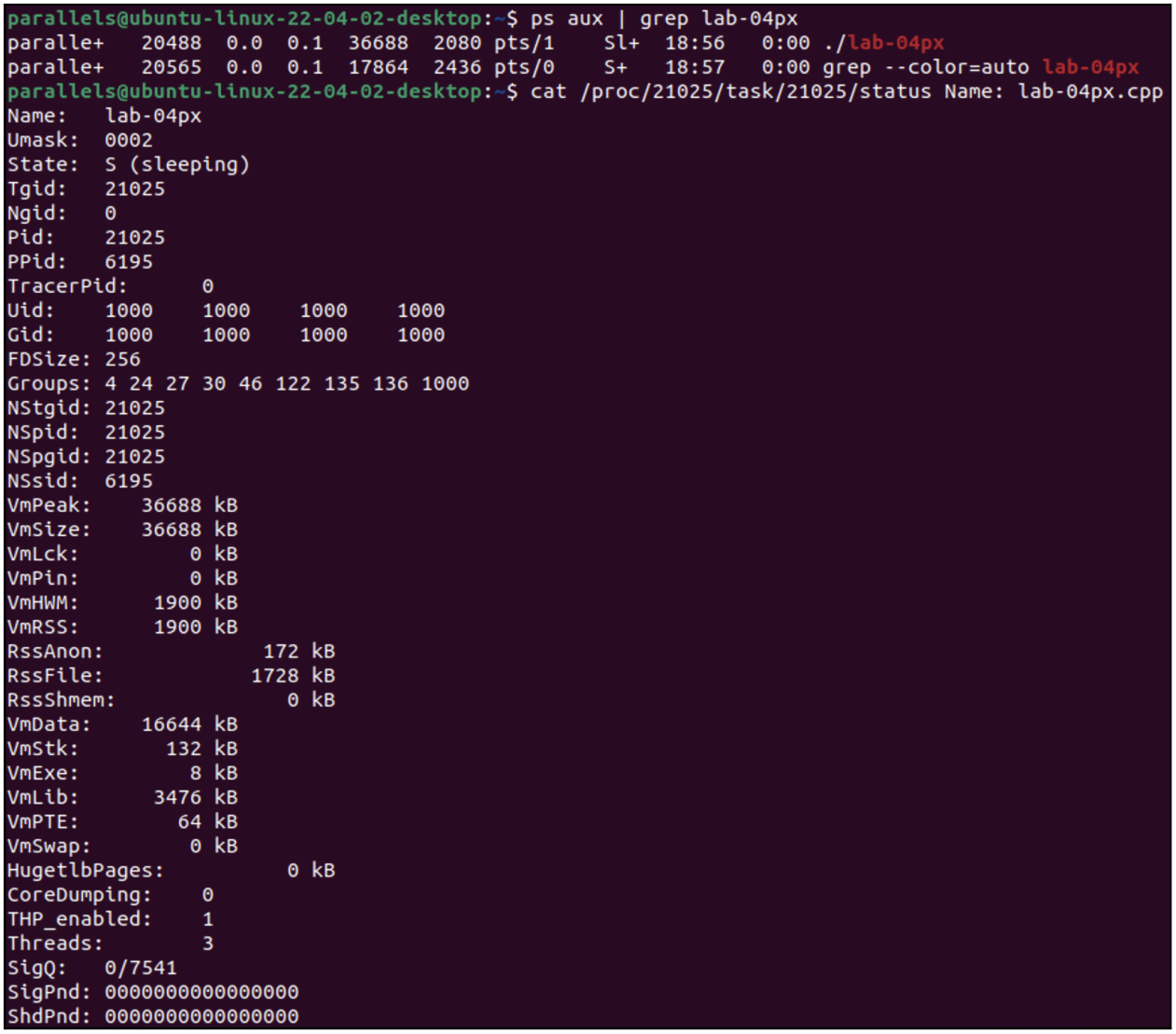

Рисунок 2.6 – Проверка потоков

Для анализа потоков использовались системные инструменты Linux. Сначала командой `ps -ef | grep Lab_04px` был найден PID запущенного процесса. Затем, исследуя виртуальную файловую систему `/proc`, в каталоге `/proc/<PID>/task/` были обнаружены поддиректории, соответствующие

идентификаторам всех потоков (TID) процесса. Детальная информация о состоянии и атрибутах каждого потока была получена с помощью команды `cat /proc/<PID>/task/<TID>/status`.

Заключение

В ходе лабораторной работы были успешно освоены практические аспекты работы с потоками в операционных системах Windows и Linux. Для Windows разработаны и протестированы приложения, демонстрирующие создание потоков через WinAPI, их синхронизацию, управление жизненным циклом (приостановку, возобновление, принудительное завершение), а также использование Tool Help Library для анализа системных потоков. Для Linux реализовано аналогичное приложение на основе POSIX Threads (pthreads), что позволило изучить кросс-платформенный подход и освоить методы мониторинга потоков через файловую систему `/proc` и утилиту `ps`.

Проведенное сравнение показало, что, несмотря на различия в API (WinAPI и pthreads) и инструментах мониторинга (Process Explorer в Windows, `/proc` и `ps` в Linux), фундаментальные принципы управления потоками остаются общими. Приобретённые навыки написания, отладки и анализа многопоточных приложений позволяют создавать эффективные параллельные программы, адаптированные к различным операционным средам, что соответствует современным требованиям разработки программного обеспечения.

Приложение А – Листинг А.1

```
#include <windows.h>
#include <iostream>
#include <string>
#include <thread>
#include <chrono>
const std::string USERNAME = "User-9e5bb2d19";
// Структура для передачи параметров потоку
struct ThreadArgs {
    int iterations;
};
// Функция потока
DWORD WINAPI Lab_04x(LPVOID lpParam) {
    ThreadArgs* args = static_cast<ThreadArgs*>(lpParam);
    int iterations = args->iterations;
    DWORD pid = GetCurrentProcessId();
    DWORD tid = GetCurrentThreadId();
    size_t nameLen = USERNAME.length();
    for (int i = 0; i < iterations; ++i) {
        char letter = USERNAME[i % nameLen];
        printf("PID: %lu - TID: %lu - Iteration: %d - Letter: %c\n", pid, tid,
i + 1, letter);
        std::this_thread::sleep_for(std::chrono::milliseconds(350));
    }
    return 0;
}
int main(int argc, char* argv[]) {
    setlocale(LC_ALL, "ru");
    if (argc != 2) {
        std::cerr << "Usage: " << argv[0] << " <iterations_count>" <<
std::endl;
        return 1;
    }
    int iterations = std::stoi(argv[1]);
    if (iterations <= 0) {
        std::cerr << "Error: iterations count must be positive." << std::endl;
        return 1;
    }
    ThreadArgs args{ iterations };
    // Создание потока
    HANDLE hThread = CreateThread(
        nullptr,            // Дескриптор безопасности
        0,                  // Размер стека
        Lab_04x,            // Функция потока
        &args,              // Аргументы
        0,                  // Флаги
        nullptr             // ID потока
    );
    if (!hThread) {
        std::cerr << "Error creating thread: " << GetLastError() << std::endl;
        return 1;
    }
    // Ожидание завершения потока
    WaitForSingleObject(hThread, INFINITE);
```

```
    // Очистка ресурсов
    CloseHandle(hThread);
    return 0;
}
```

Приложение Б – Листинг Б.1

```
#include <windows.h>
#include <iostream>
#include <string>
#include <thread>
#include <chrono>
const std::string USERNAME = "User-9e5bb2d19";
// Структура для передачи параметров потока
struct ThreadArgs {
    int iterations;
    HANDLE waitFor1;
    HANDLE waitFor2;
    bool isMain;
};
// Функция потока
DWORD WINAPI Lab_04x(LPVOID lpParam) {
    ThreadArgs* args = static_cast<ThreadArgs*>(lpParam);
    int iterations = args->iterations;
    DWORD pid = GetCurrentProcessId();
    DWORD tid = GetCurrentThreadId();
    size_t nameLen = USERNAME.length();
    printf("Thread TID: %lu started\n", tid);
    for (int i = 0; i < iterations; ++i) {
        char letter = USERNAME[i % nameLen];
        printf("PID: %lu - TID: %lu - Iteration: %d - Letter: %c\n", pid, tid,
i + 1, letter);
        std::this_thread::sleep_for(std::chrono::milliseconds(350));
    }
    // Если это главный поток — ждём завершения дочерних
    if (args->isMain) {
        printf("Main thread TID: %lu finished iterations, waiting for child
threads...\n", tid);
        WaitForSingleObject(args->waitFor1, INFINITE);
        WaitForSingleObject(args->waitFor2, INFINITE);
    }
    printf("Thread TID: %lu finished\n", tid);
    return 0;
}
int main() {
    setlocale(LC_ALL, "ru");
    // Аргументы для дочерних потоков
    ThreadArgs args1{ 50, nullptr, nullptr, false };
    ThreadArgs args2{ 125, nullptr, nullptr, false };
    // Создание дочерних потоков
    HANDLE hThread1 = CreateThread(nullptr, 0, Lab_04x, &args1, 0, nullptr);
    HANDLE hThread2 = CreateThread(nullptr, 0, Lab_04x, &args2, 0, nullptr);
    if (!hThread1 || !hThread2) {
        std::cerr << "Error creating child threads\n";
        return 1;
    }
    // Аргументы для главного потока
    ThreadArgs argsMain{ 100, hThread1, hThread2, true };
    // Создание главного потока
    HANDLE hMainThread = CreateThread(nullptr, 0, Lab_04x, &argsMain, 0,
```

```
nullptr);
    if (!hMainThread) {
        std::cerr << "Error creating main thread\n";
        CloseHandle(hThread1);
        CloseHandle(hThread2);
        return 1;
    }
    // Ожидание завершения главного потока
    WaitForSingleObject(hMainThread, INFINITE);
    // Очистка ресурсов
    CloseHandle(hThread1);
    CloseHandle(hThread2);
    CloseHandle(hMainThread);
    printf("All threads finished. Program completed.\n");
    return 0;
}
```

Приложение В – Листинг В.1

```
#include <windows.h>
#include <iostream>
#include <string>
#include <thread>
#include <chrono>
const std::string USERNAME = "User-9e5bb2d19";
// Структура для передачи параметров потока
struct ThreadArgs {
    int iterations;
    HANDLE selfHandle;
    HANDLE resumeEvent;
    bool waitForMain;
};
// Функция потока
DWORD WINAPI Lab_04x(LPVOID lpParam) {
    ThreadArgs* args = static_cast<ThreadArgs*>(lpParam);
    DWORD pid = GetCurrentProcessId();
    DWORD tid = GetCurrentThreadId();
    size_t nameLen = USERNAME.length();
    printf("Thread TID: %lu started\n", tid);
    for (int i = 0; i < args->iterations; ++i) {
        // Поток 2 ждёт сигнала от главного потока
        if (args->waitForMain && i == 40) {
            printf("Thread TID: %lu suspended at iteration %d, waiting for
signal from main thread\n", tid, i + 1);
            WaitForSingleObject(args->resumeEvent, INFINITE);
            printf("Thread TID: %lu resumed after signal\n", tid);
        }
        char letter = USERNAME[i % nameLen];
        printf("PID: %lu - TID: %lu - Iteration: %d - Letter: %c\n", pid, tid,
i + 1, letter);
        std::this_thread::sleep_for(std::chrono::milliseconds(350));
    }
    printf("Thread TID: %lu finished\n", tid);
    return 0;
}
int main() {
    setlocale(LC_ALL, "ru");
    const int mainIterations = 100;
    size_t nameLen = USERNAME.length();
    DWORD pid = GetCurrentProcessId();
    DWORD mainTid = GetCurrentThreadId();
    // Событие для возобновления второго потока
    HANDLE resumeEvent = CreateEvent(nullptr, TRUE, FALSE, nullptr);
    // Аргументы и создание первого потока
    ThreadArgs args1{ 50, nullptr, nullptr, false };
    HANDLE hThread1 = CreateThread(nullptr, 0, Lab_04x, &args1,
CREATE_SUSPENDED, nullptr);
    args1.selfHandle = hThread1;
    ResumeThread(hThread1);
    // Аргументы и создание второго потока
    ThreadArgs args2{ 125, nullptr, resumeEvent, true };
    HANDLE hThread2 = CreateThread(nullptr, 0, Lab_04x, &args2, 0, nullptr);
```

```
    args2.selfHandle = hThread2;
    printf("Main thread TID: %lu started\n", mainTid);
    for (int i = 0; i < mainIterations; ++i) {
        // Приостановка первого потока на 20-й итерации
        if (i == 20) {
            SuspendThread(hThread1);
            printf("Main thread suspended thread 1 at iteration %d\n", i + 1);
        }
        // Возобновление первого потока на 60-й итерации
        if (i == 60) {
            ResumeThread(hThread1);
            printf("Main thread resumed thread 1 at iteration %d\n", i + 1);
        }
        // Завершение цикла — сигнал второму потоку
        if (i == mainIterations - 1) {
            SetEvent(resumeEvent);
            printf("Main thread finished cycle and signaled thread 2\n");
        }
        char letter = USERNAME[i % nameLen];
        printf("PID: %lu - TID: %lu - Iteration: %d - Letter: %c\n", pid,
mainTid, i + 1, letter);
        std::this_thread::sleep_for(std::chrono::milliseconds(350));
    }
    // Ожидание завершения дочерних потоков
    WaitForSingleObject(hThread1, INFINITE);
    WaitForSingleObject(hThread2, INFINITE);
    // Очистка ресурсов
    CloseHandle(hThread1);
    CloseHandle(hThread2);
    CloseHandle(resumeEvent);
    printf("All threads finished. Program completed.\n");
    return 0;
}
```

Приложение Г – Листинг Г.1

```
#include <windows.h>
#include <iostream>
#include <string>
#include <thread>
#include <chrono>
const std::string USERNAME = "User-9e5bb2d19";
// Структура для передачи параметров потока
struct ThreadArgs {
    int iterations;
};
// Функция потока
DWORD WINAPI Lab_04x(LPVOID lpParam) {
    ThreadArgs* args = static_cast<ThreadArgs*>(lpParam);
    DWORD pid = GetCurrentProcessId();
    DWORD tid = GetCurrentThreadId();
    size_t nameLen = USERNAME.length();
    printf("Thread TID: %lu started\n", tid);
    for (int i = 0; i < args->iterations; ++i) {
        char letter = USERNAME[i % nameLen];
        printf("PID: %lu - TID: %lu - Iteration: %d - Letter: %c\n", pid, tid,
i + 1, letter);
        std::this_thread::sleep_for(std::chrono::milliseconds(350));
    }
    printf("Thread TID: %lu finished\n", tid);
    return 0;
}
int main() {
    setlocale(LC_ALL, "ru");
    const int mainIterations = 100;
    size_t nameLen = USERNAME.length();
    DWORD pid = GetCurrentProcessId();
    DWORD mainTid = GetCurrentThreadId();
    // Аргументы для дочерних потоков
    ThreadArgs args1{ 50 };
    ThreadArgs args2{ 125 };
    // Создание дочерних потоков
    HANDLE hThread1 = CreateThread(nullptr, 0, Lab_04x, &args1, 0, nullptr);
    HANDLE hThread2 = CreateThread(nullptr, 0, Lab_04x, &args2, 0, nullptr);
    if (!hThread1 || !hThread2) {
        std::cerr << "Error creating threads\n";
        return 1;
    }
    printf("Main thread TID: %lu started\n", mainTid);
    for (int i = 0; i < mainIterations; ++i) {
        // Завершение второго потока на 40-й итерации
        if (i == 40) {
            TerminateThread(hThread2, 0);
            printf("Main thread terminated thread 2 at iteration %d\n", i +
1);
        }
        char letter = USERNAME[i % nameLen];
        printf("PID: %lu - TID: %lu - Iteration: %d - Letter: %c\n", pid,
mainTid, i + 1, letter);
```

```
        std::this_thread::sleep_for(std::chrono::milliseconds(350));
    }
    // Ожидание завершения первого потока
    HANDLE handles[2] = { hThread1, hThread2 };
    WaitForMultipleObjects(2, handles, TRUE, INFINITE);
    // Очистка ресурсов
    CloseHandle(hThread1);
    CloseHandle(hThread2);
    printf("All threads finished. Program completed.\n");
    return 0;
}
```

Приложение Д – Листинг Д.1

```
#include <windows.h>
#include <tlhelp32.h>
#include <iostream>
int main() {
    setlocale(LC_ALL, "ru");
    DWORD currentPID = GetCurrentProcessId();
    HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPTHREAD, 0);
    if (snapshot == INVALID_HANDLE_VALUE) {
        std::cerr << "Не удалось получить снимок потоков." << std::endl;
        return 1;
    }
    THREADENTRY32 te32;
    te32.dwSize = sizeof(THREADENTRY32);
    if (!Thread32First(snapshot, &te32)) {
        std::cerr << "Ошибка при получении первого потока." << std::endl;
        CloseHandle(snapshot);
        return 1;
    }
    std::cout << "Thread list of current process (PID: " << currentPID << "):"
<< std::endl;
    do {
        if (te32.th32OwnerProcessID == currentPID) {
            std::cout << "TID: " << te32.th32ThreadID << std::endl;
        }
    } while (Thread32Next(snapshot, &te32));
    CloseHandle(snapshot);
    return 0;
}
```

Приложение Е – Листинг Е.1

```
#include <iostream>
#include <string>
#include <thread>
#include <chrono>
#include <pthread.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/syscall.h>

const std::string USERNAME = "User-9e5bb2d19";

// Структура для передачи параметров потоку
struct ThreadArgs {
    int iterations;
};

// Функция потока для Linux
void* Lab_04x(void* lpParam) {
    ThreadArgs* args = static_cast<ThreadArgs*>(lpParam);
    int iterations = args->iterations;

    // В Linux: getpid() возвращает PID процесса, syscall(SYS_gettid)
возвращает TID потока
    pid_t pid = getpid();
    pid_t tid = syscall(SYS_gettid);

    size_t nameLen = USERNAME.length();

    for (int i = 0; i < iterations; ++i) {
        char letter = USERNAME[i % nameLen];
        printf("PID: %d - TID: %ld - Iteration: %d - Letter: %c\n",
               pid, static_cast<long>(tid), i + 1, letter);
        std::this_thread::sleep_for(std::chrono::milliseconds(350));
    }

    return nullptr;
}

int main(int argc, char* argv[]) {
    setlocale(LC_ALL, "ru");

    if (argc != 2) {
        std::cerr << "Usage: " << argv[0] << " <iterations_count>" <<
std::endl;
        return 1;
    }

    int iterations = std::stoi(argv[1]);
    if (iterations <= 0) {
        std::cerr << "Error: iterations count must be positive." << std::endl;
        return 1;
    }

    ThreadArgs args{ iterations };

    // Создание потока в Linux
    pthread_t thread;
    int result = pthread_create(&thread, nullptr, Lab_04x, &args);

    if (result != 0) {
        std::cerr << "Error creating thread: " << result << std::endl;
```

```
        return 1;
    }

    // Ожидание завершения потока
    pthread_join(thread, nullptr);

    return 0;
}
```

Приложение Ж – Листинг Ж.1

```
#include <windows.h>
#include <tlhelp32.h>
#include <iostream>

int main() {
    setlocale(LC_ALL, "ru");

    DWORD currentPID = GetCurrentProcessId();
    HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPTHREAD, 0);

    if (snapshot == INVALID_HANDLE_VALUE) {
        std::cerr << "Не удалось получить снимок потоков." << std::endl;
        return 1;
    }

    THREADENTRY32 te32;
    te32.dwSize = sizeof(THREADENTRY32);
    if (!Thread32First(snapshot, &te32)) {
        std::cerr << "Ошибка при получении первого потока." << std::endl;
        CloseHandle(snapshot);
        return 1;
    }
    std::cout << "Список потоков текущего процесса (PID: " << currentPID <<
"):" << std::endl;
    do {
        if (te32.th32OwnerProcessID == currentPID) {
            std::cout << "TID: " << te32.th32ThreadID << std::endl;
        }
    } while (Thread32Next(snapshot, &te32));

    CloseHandle(snapshot);
    return 0;
}
```