**Белорусский государственный технологический университет**

**Факультет информационных технологий**

**Кафедра программной инженерии**

**Реферат**

По дисциплине «Основы алгоритмизации и программирования»

На тему «Процессы и потоки»

Выполнил:

Студент 1 курса 10 группы ПИ

Макаревич Кирилл Витальевич

Проверил: Белодед Николай Иванович

2024, Минск

**Оглавление**

**1.**[**Введение**](#_Введение)**3**

**2.** [**Подробно про процесс**](#_Подробно_про_процесс)**4**

3. [Создание процессов](#_Создание_процессов.)5

**4.** [**Завершение процесса**](#_Завершение_процесса)**7**

**5.** [**Подробно про поток**](#_Подробно_про_поток)**9**

**6.** [**Создание потоков**](#_Создание_потоков)**10**

**7.** [**Приостановка и возобновление выполнения потока**](#_Приостановка_и_возобновление)**12**

**8.** [**Отличия процесса от потока**](#_Отличие_процесса_от)**13**

**9.** [**Основные определения, которые я использовал в реферате15**](#_Основные_определения,_которые)

**10.** [**Заключение**](#_Заключение)**16**

# **Введение**

В современных операционных системах процессы и потоки играют ключевую роль, обеспечивая многозадачность и позволяя программам эффективно использовать вычислительные ресурсы. В данном реферате мы рассмотрим, что такое процессы и потоки, чем они отличаются, и как их можно применять в программировании на языке C++.

# **Подробно про процесс**

**Процесс** – это контейнер для набора ресурсов, используемых потоками, которые выполняют экземпляр программы. На самом высоком уровне абстракции процесс включает следующее:

* закрытое виртуальное адресное пространство – диапазон адресов виртуальной памяти, которым может пользоваться процесс;
* исполняемую программу – начальный код и данные, проецируемые на виртуальное адресное пространство процесса;
* список открытых описателей различных системных ресурсов – семафоров, коммуникационных портов, файлов и других объектов, доступных всем потокам в данном процессе (см. Рисунок 1);
* контекст защиты, называемый маркером доступа (Access Token) и идентифицирующий пользователя, группы безопасности и привилегии, сопоставленные с процессом;
* уникальный идентификаторпроцесса (PID) (во внутрисистемной терминологии называемый идентификатором клиента);
* минимум один поток.

# **Создание процесса**

Самый распространенный способ начала процесса — это осуществить запуск приложения в проводнике. Новые процессы обычно создаются:

1. При запуске операционной системы;
2. При появлении запроса на создание процесса — происходит в случае, если работающий процесс создает новый процесс.

*Пример создания нового процесса*

|  |
| --- |
| #include <Windows.h>  int main() {  STARTUPINFO si;  PROCESS\_INFORMATION pi;  ZeroMemory(&si, sizeof(si));  si.cb = sizeof(si);  if (CreateProcess(L"c:\\windows\\system32\\calc.exe", NULL, NULL, NULL, FALSE, NORMAL\_PRIORITY\_CLASS, NULL, NULL, &si, &pi)) {  CloseHandle(pi.hThread);  CloseHandle(pi.hProcess);  }  return 0;  } |

*Описание работы программы*

Эта программа предназначена для создания нового процесса в операционной системе Windows с нормальным уровнем приоритета, запускающего калькулятор (calc.exe). Основные этапы работы программы:

1. ***Инициализация структуры STARTUPINFO (si*)**: Эта структура хранит информацию о создаваемом процессе. Сначала её память очищается с помощью ZeroMemory, а затем в поле cb записывается её размер для корректного создания процесса.
2. ***Создание процесса***: Функция CreateProcess запускает calc.exe. При успешном запуске (если функция возвращает истина) структура PROCESS\_INFORMATION (pi) заполняется сведениями о новом процессе, включая его идентификатор и дескрипторы процесса и основного потока.
3. ***Закрытие дескрипторов***: Если родительскому процессу не нужно взаимодействовать с новым процессом, то для предотвращения утечек ресурсов дескрипторы потока и процесса закрываются с помощью CloseHandle.

**1. ZeroMemory**

* **Описание**: Эта функция инициализирует область памяти нулями. В моём коде она используется для очистки структуры STARTUPINFO, чтобы избежать использования неинициализированных значений.
* **Параметры**:
  1. LPVOID dst: Указатель на область памяти, которую необходимо очистить.
  2. SIZE\_T size: Количество байт, которые нужно очистить.
* **Возвращаемое значение**: Нет.

**2. CreateProcess**

* **Описание**: Эта функция создает новый процесс и его основной поток. В моём коде она используется для запуска приложения calc.exe.
* **Параметры**:
  1. LPCWSTR lpApplicationName: Указатель на строку, содержащую имя исполняемого файла.
  2. LPSTR lpCommandLine: Указатель на строку командной строки.
  3. LPSECURITY\_ATTRIBUTES lpProcessAttributes: Указатель на атрибуты безопасности для нового процесса.
  4. LPSECURITY\_ATTRIBUTES lpThreadAttributes: Указатель на атрибуты безопасности для нового потока.
  5. BOOL bInheritHandles: Указывает, должны ли дескрипторы, открытые для родительского процесса, наследоваться дочерним процессом.
  6. DWORD dwCreationFlags: Указывает параметры создания процесса.
  7. LPVOID lpEnvironment: Указатель на окружение для нового процесса.
  8. LPCTSTR lpCurrentDirectory: Указатель на текущий каталог нового процесса.
  9. LPSTARTUPINFO lpStartupInfo: Указатель на структуру, содержащую информацию о запуске.
  10. LPPROCESS\_INFORMATION lpProcessInformation: Указатель на структуру, в которую будет записана информация о новом процессе.
* **Возвращаемое значение**: Если функция успешна, возвращает ненулевое значение. В противном случае возвращает 0.

**3. CloseHandle**

* **Описание**: Эта функция закрывает открытый дескриптор, освобождая связанные с ним системные ресурсы. В моём коде она используется для закрытия дескрипторов потока и процесса, чтобы избежать утечек памяти и ресурсов.
* **Параметры**:
* HANDLE hObject: Дескриптор, который нужно закрыть.
* **Возвращаемое значение**: Если функция успешна, возвращает ненулевое значение. В противном случае возвращает 0.

# 

# **Завершение процесса**

Рекомендуется проектировать приложение так, чтобы его процесс завершался только после возврата управления функцией первичного потока.

Процесс можно завершить четырьмя способами:

1. ***Возврат управления из функции первичного потока*** (например main):  
   Это естественный способ завершения программы. Когда функция main достигает конца и возвращает значение, операционная система завершает процесс. Это **предпочтительный способ** завершения, так как программа может корректно освободить все ресурсы, выполнить завершающие операции и вывести нужные коды возврата.
2. ***Вызов ExitProcess из любого потока***:  
   Функция ExitProcess немедленно завершает все потоки в процессе, освобождая ресурсы и закрывая дескрипторы. Обычно это используется в случаях, когда программа хочет завершиться по определённому условию. Это также **предпочтительный способ**, так как он завершает процесс, но позволяет правильно освободить ресурсы.

|  |
| --- |
| #include <Windows.h>  #include <iostream>  int main() {  setlocale(LC\_ALL, "ru");  std::cout << "Программа запущена." << std::endl;  // Условие завершения (например, ошибка или достижение конца программы)  bool shouldExit = true;  if (shouldExit) {  std::cout << "Условие выполнено. Завершаем процесс с помощью ExitProcess." << std::endl;  ExitProcess(0); // Завершает процесс с кодом возврата 0  }  std::cout << "Эта строка не будет выполнена, так как процесс завершён." << std::endl;  return 0;  } |

1. **Вызов TerminateProcess из другого процесса**:  
   Функция TerminateProcess завершает процесс принудительно и немедленно, не давая ему шанса освободить ресурсы. Этот способ **нежелателен**, так как процесс прерывается в любой момент, и это может привести к утечкам памяти, повреждению данных или неполным операциям.

|  |
| --- |
| #include <Windows.h>  #include <iostream>  int main() {  setlocale(LC\_ALL, "ru");  // Запуск калькулятора  STARTUPINFO si = { sizeof(si) };  PROCESS\_INFORMATION pi;  if (CreateProcess(L"C:\\Windows\\System32\\calc.exe", NULL, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)) {  std::cout << "Калькулятор запущен. Завершаем процесс принудительно..." << std::endl;  // Завершаем процесс калькулятора  TerminateProcess(pi.hProcess, 0);  // Закрываем дескрипторы  CloseHandle(pi.hProcess);  CloseHandle(pi.hThread);  }  else {  std::cerr << "Не удалось запустить калькулятор." << std::endl;  }  return 0;  } |

1. **Самостоятельное завершение всех потоков**:  
   Если все потоки процесса завершаются сами по себе, процесс также завершится. Однако это **очень редкий случай**, так как большинство программ используют главный поток для управления процессом, и он не завершается до тех пор, пока не завершится работа всей программы.

# 

# **Подробно про поток**

Поток — это исполнительная единица, являющаяся частью процесса. Процесс может иметь несколько потоков, выполняющихся одновременно. Это единица выполнения в параллельном программировании. Поток является легким и может управляться планировщиком независимо. Это поможет вам повысить производительность приложения с помощью параллелизма.

На уровне ОС поток представляется блоком потока с информационными структурами:

* идентификатор потока TID (Thread ID)
* стартовый адрес потока
* данные о стеках потока:

1. пользовательского режима;
2. режима ядра;

* информация о контексте потока
* информация для диспетчеризации
* время создания/завершения
* информация о локальной памяти потока (TLS)

# **Создание потоков**

Для создания нового потока в процессе применяется функция CreateThread. Каждый поток начинает своё выполнение с заданной функции и заканчивает при возврате из подпрограммы или при вызове функции ExitThread либо TerminateThread.

*Пример создания потока:*

|  |
| --- |
| #include <iostream>  #include <windows.h>  // Глобальные переменные для хранения результатов  int sum1 = 0;  int sum2 = 0;  // Функция, исполняемая потоками  DWORD WINAPI CalculateSum(LPVOID param) {  int start = (int)param; // Начальное значение  // Считаем сумму чисел в диапазоне  for (int i = start; i < start + 5; ++i) {  if (start == 1) {  sum1 += i; // Сумма от 1 до 5  }  else {  sum2 += i; // Сумма от 6 до 10  }  }  return 0;  }  int main() {  setlocale(LC\_ALL, "ru");  // Создание потоков  HANDLE hThread1 = CreateThread(NULL, 0, CalculateSum, (LPVOID)1, 0, NULL); // Поток для суммы от 1 до 5  HANDLE hThread2 = CreateThread(NULL, 0, CalculateSum, (LPVOID)6, 0, NULL); // Поток для суммы от 6 до 10  // Ожидание завершения потоков  WaitForSingleObject(hThread1, INFINITE);  WaitForSingleObject(hThread2, INFINITE);  // Закрытие дескрипторов потоков  CloseHandle(hThread1);  CloseHandle(hThread2);  // Вывод результатов  std::cout << "Сумма от 1 до 5: " << sum1 << std::endl;  std::cout << "Сумма от 6 до 10: " << sum2 << std::endl;  std::cout << "Общая сумма: " << sum1 + sum2 << std::endl;  return 0;  } |

*Описание работы программы:*

* Глобальные переменные sum1 и sum2: используются для хранения результатов сумм от 1 до 5 и от 6 до 10 соответственно.
* Функция CalculateSum: принимает начальное значение в качестве параметра. В зависимости от этого значения функция вычисляет сумму чисел от 1 до 5 или от 6 до 10 и добавляет результат в соответствующую глобальную переменную.
* Создание потоков: создаются два потока, каждый из которых обрабатывает свою часть работы.
* Ожидание завершения: программа ждёт завершения обоих потоков.
* Вывод результата: после завершения потоков результаты выводятся на экран.

**1. CalculateSum:**

* **DWORD WINAPI:** Определяет возвращаемый тип функции потока и соглашение о вызове. DWORD - это 32-битное беззнаковое целое число.
* **LPVOID param:** Параметр, передаваемый в функцию потока. Он используется для передачи начального значения (например, 1 или 6).
* Внутри функции происходит вычисление суммы чисел в заданном диапазоне. В зависимости от начального значения (1 или 6) результаты суммируются в соответствующие глобальные переменные (sum1 или sum2).

***2.* CreateThread:**

HANDLE hThread1 = CreateThread(NULL, 0, CalculateSum, (LPVOID)1, 0, NULL);

HANDLE hThread2 = CreateThread(NULL, 0, CalculateSum, (LPVOID)6, 0, NULL);

* Создаёт два потока, которые выполняют функцию CalculateSum с разными начальными значениями.
* **NULL:** Атрибуты безопасности потока (не используются).
* **0:** Размер стека (используется значение по умолчанию).
* **CalculateSum:** Указатель на функцию, которую будет выполнять поток.
* **(LPVOID)1 и (LPVOID)6:** Значения, передаваемые в поток (начальные значения для суммы).
* **0:** Флаги создания (обычно 0 для запуска сразу).
* **NULL:** Указатель на переменную, где будет храниться идентификатор потока (не используется).

**3. WaitForSingleObject:**

Ожидает завершения выполнения потоков. Главный поток приостанавливается до тех пор, пока указанные потоки (hThread1 и hThread2) не завершатся.

**INFINITE:** Ожидание без ограничения по времени.

**4. CloseHandle:**

Закрывает дескрипторы потоков, освобождая ресурсы, которые они занимали. Это важно для предотвращения утечек памяти.

# **Приостановка и возобновление выполнения потока**

Поток может быть приостановлен с помощью функции **SuspendThread**. При этом поток временно прекращает выполнение, и ему не выделяется процессорное время. Важно отметить, что поток может приостановить себя сам, но возобновить работу приостановленного потока может только другой поток, который не находится в состоянии приостановки. Для возобновления работы используется функция **ResumeThread**.

**Примечания:**

* Использование функций **SuspendThread** и **ResumeThread** требует осторожности, поскольку они могут привести к проблемам с синхронизацией и взаимным блокировкам (deadlocks). Это особенно актуально, если потоки зависят друг от друга или используют общие ресурсы.
* Для более безопасного управления состоянием потоков рекомендуется использовать механизмы синхронизации, такие как мьютексы, критические секции или события.

# **Отличия процесса от потока**

Процесс рассматривается ОС, как заявка на все виды ресурсов (память, файлы и пр.), кроме одного — процессорного времени. Поток — это заявка на процессорное время. Процесс — это всего лишь способ сгруппировать взаимосвязанные данные и ресурсы, а потоки — это единицы выполнения (unit of execution), которые выполняются на процессоре.

**Важные свойства процесса:**

* Создание каждого процесса требует отдельных системных вызовов для каждого процесса.
* Это изолированный исполнительный объект, который не обменивается данными и информацией.
* Управление процессом принимает больше системных вызовов.
* У процесса есть стек, куча памяти с памятью и карта данных.

**Важные свойства потока:**

* Один системный вызов может создать более одного потока
* Потоки обмениваются данными и информацией.
* Потоки разделяют инструкции, глобальные регионы и регионы кучи. Однако у него есть свой регистр и стек.
* Управление потоками требует очень мало системных вызовов или не требует их вообще, поскольку связь между потоками может быть достигнута с использованием общей памяти.

**Ключевая разница между процессом и потоком**

* Процесс означает, что программа выполняется, тогда как поток означает сегмент процесса.
* Процесс не является легким, тогда как потоки являются легкими.
* Процессу требуется больше времени для завершения, а потоку требуется меньше времени для завершения.
* Процессу требуется больше времени для создания, тогда как для создания потока требуется меньше времени.
* Процесс, вероятно, требует больше времени для переключения контекста, тогда как потоки требуют меньше времени для переключения контекста.
* Процесс в основном изолирован, тогда как потоки совместно используют память.
* Процесс не обменивается данными, а потоки обмениваются данными друг с другом.

**Таблица различий:**

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Процесс** | **Поток** |
| Определение | Процесс означает, что программа выполняется. | Поток является частью процесса, выполняющей конкретную часть кода. |
| Небольшой вес | Процесс не является легким. | Поток легкий. |
| Время прекращения | Для завершения процесса требуется больше времени. | Для завершения потока требуется меньше времени. |
| Время создания | На создание уходит больше времени. | На создание уходит меньше времени. |
| Связь | Для связи между процессами требуется больше времени по сравнению с потоком. | Связь между потоками требует меньше времени по сравнению с процессами. |
| Время переключения контекста | Переключение контекста занимает больше времени. | Переключение контекста занимает меньше времени. |
| Ресурс | Процесс потребляет больше ресурсов. | Поток потребляет меньше ресурсов. |
| Лечение ОС | Различные процессы выполняются ОС отдельно. | Все одноранговые потоки уровня рассматриваются ОС как одна задача. |
| Память | Процесс преимущественно изолирован. | Потоки разделяют память. |
| Разделение контекста | Он не передает данные | Потоки обмениваются данными друг с другом. |

# 

# **Основные определения, которые я использовал в реферате**

**1. Многозадачность и многопоточность:**

***Многозадачность*** — это возможность операционной системы выполнять несколько задач одновременно или создавать видимость параллельной работы.

***Многопоточность*** — это способность платформы или приложения делить один процесс на несколько потоков, работающих одновременно, что позволяет эффективнее использовать ресурсы системы.

Настоящая параллельность достигается только в многопроцессорных системах с несколькими вычислительными блоками. В однопроцессорных системах многозадачность реализуется псевдопараллельно, чередуя выполнение процессов на малых интервалах времени.

***2.* Процессы и потоки*:***

В различных источниках можно встретить множество определений процессов и потоков, что связано как с развитием операционных систем, так и с разными подходами к изучению этих понятий. В рамках этого реферата я использовал следующие определения:

**С точки зрения пользователя:**

* ***Процесс*** — это выполняющаяся программа. Один или несколько потоков выполняются в контексте процесса.
* ***Потоки*** — то базовая единица, которой операционная система выделяет процессорное время.

**3. *Время переключения контекста* -** время, необходимое операционной системе для сохранения состояния текущего потока или процесса и загрузки состояния другого потока или процесса, чтобы передать ему управление.

**4. *Разделение контекста (или переключение контекста)*** - это процесс, при котором происходит передача управления от одного потока или процесса к другому. Это ключевая концепция многозадачности, позволяющая системе эффективно управлять несколькими задачами одновременно.

**5. *Лечение ОС (операционной системы)***— это процесс поддержки и оптимизации работы системы, включающий:

1. ***Обновление***: Регулярная установка обновлений и патчей для повышения безопасности и производительности.
2. ***Устранение ошибок***: Исправление программных сбоев и проблем, которые могут вызывать зависания или крахи.
3. ***Оптимизация***: Настройка параметров для улучшения быстродействия и управления ресурсами.
4. ***Безопасность***: Установка антивирусного ПО и брандмауэров для защиты от угроз.
5. ***Восстановление***: Создание резервных копий и точек восстановления для возврата к рабочему состоянию в случае неполадок.

# **Заключение**

Я писал описание работы программ, а не алгоритмы, так как раньше не работал с процессами, потоками и разбираюсь с ними прямо сейчас и мне легче писать то, что делает программа. Поскольку программа небольшая, я думаю, что блок-схемы и тд. не нужны и мне кажется, что текстовое описание будет более доступным.

Мы рассмотрели важные аспекты процессов и потоков в современных операционных системах. Процессы и потоки позволяют программам эффективно использовать вычислительные ресурсы и обеспечивают возможность выполнения нескольких задач одновременно. Понимание процессов и потоков помогает программистам разрабатывать более производительные и отзывчивые приложения. Используя потоки, можно ускорить выполнение задач и сделать программы более эффективными. В будущем, знание этих основ позволит лучше разбираться в разработке программного обеспечения и использовании ресурсов операционной системы.