Задача 12. Разбор алгоритма флаговой синхронизации для барьера. Детальное понимание и обоснование известного алгоритма флаговой синхронизации для реализации барьера (см. учебник Эндрюс). Понимание сути алгоритма и как он обеспечивает эффективную синхронизацию.

Исследование провёл студент группы 22207 Гордеев Никита

Дата выполнения работы 25.12.2022 (Вариант 2)

Проблема

- **Во многих параллельных итерационных алгоритмов результат каждой итерации зависит от предыдущей.**
- **Для эффективной реализации воспользуемся барьером, которого должны достигнуть все запущенные процессы до завершения для их синхронизации.**

Задачи

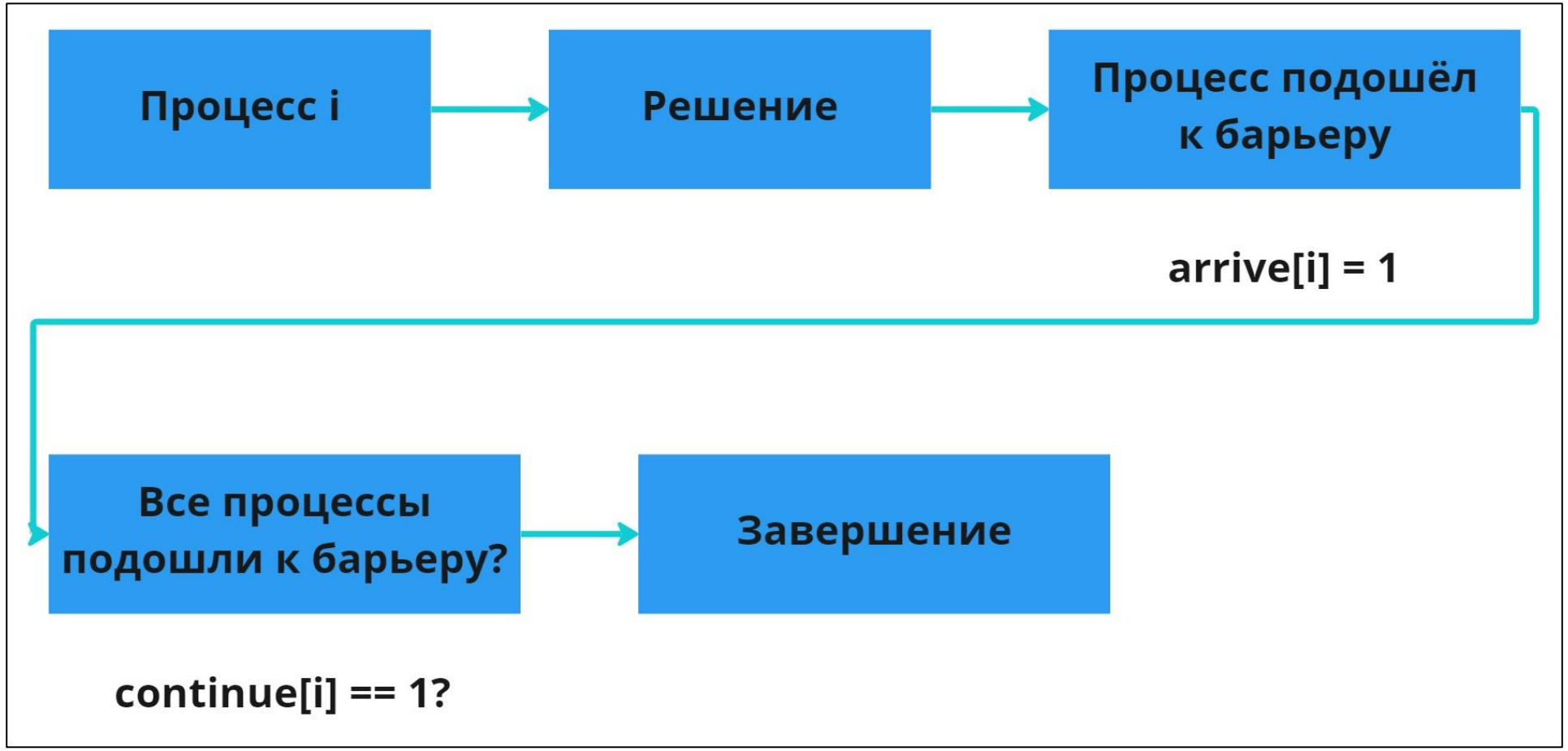
- **Детальное понимание и обоснование известного алгоритма флаговой синхронизации для реализации барьера (см. учебник Эндрюс).**
- **Понимание сути алгоритма и как он обеспечивает эффективную синхронизацию.**

Флаговая синхронизация для реализации барьера

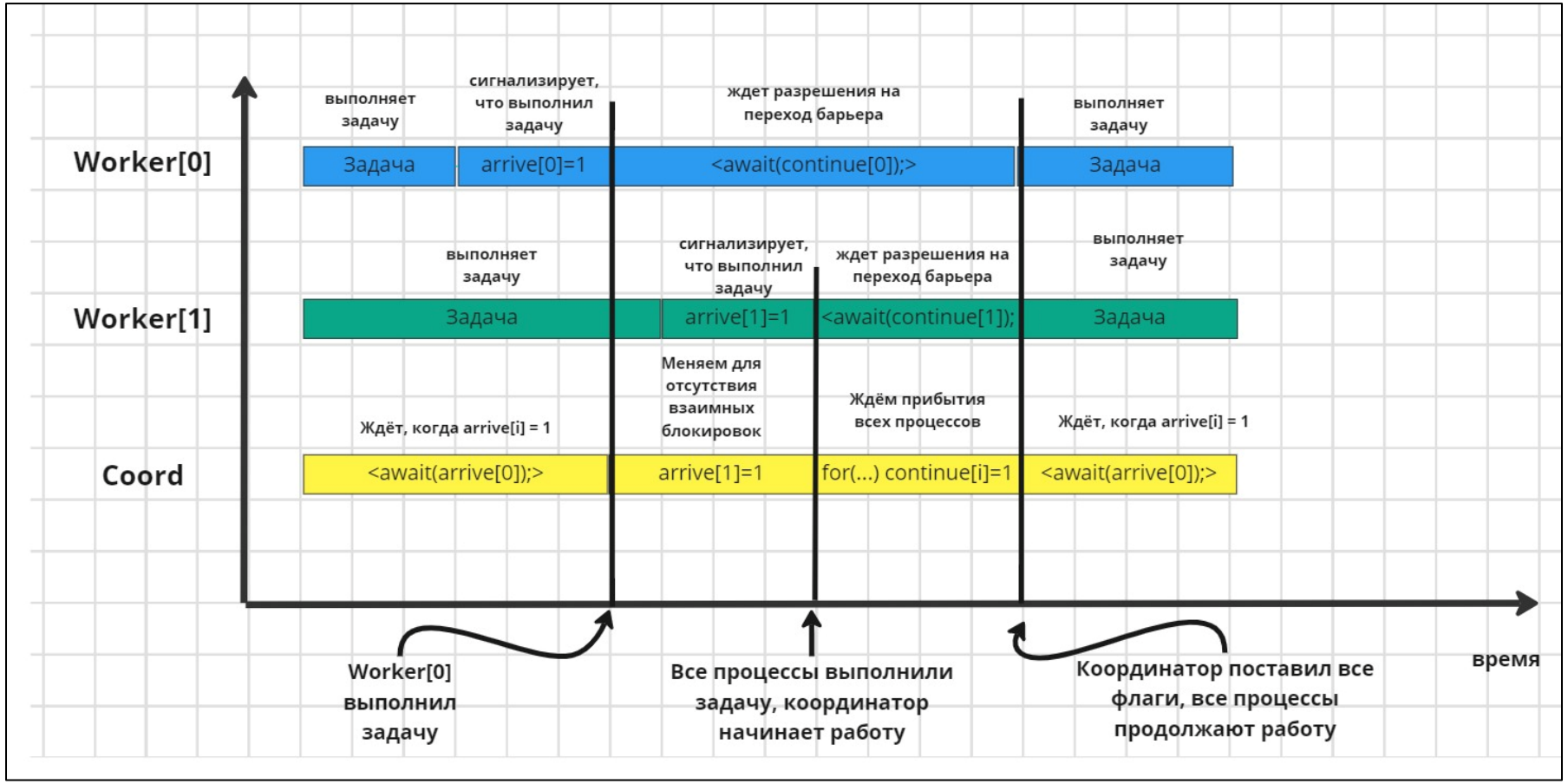
```
// Для каждого процесса заводится два флага: arrive и continue
process Worker(i = 1 to n) {
____while(true) {
// процесс выполнил задачу, он сигнализирует об этом координатору
_________arrive[i] = 1;
// ждет разрешения на переход барьера, то есть когда будет установлен флаг continue
_________<await (continue[i] == 1);>
_________continue[i] = 0
____}
}
```

```
// Управляющий процесс, который будет устанавливать флаги для рабочих процессов
process Coordinator {
____while(true) {
// Идём по процессам от 1 до n
_______for [i = 1 to n] {
// Ждёт, когда arrive[i] = 1
_________<await (arrive[i] == 1);>
// Меняем arrive[i] = 0 для отсутствия взаимных блокировок
_________arrive[i] == 0;
____}
// Ждём прибытия всех процессов и устанавливаем continue[i] = 1
____for [i = 1 to n] {
_________continue[i] = 1;
____}
}
```

Схема флаговой синхронизация для реализации барьера



Временная схема флаговой синхронизация для реализации барьера



Обоснование синхронизации

- **Рабочие процессы могут начать новую итерацию только после того, как будет установлен флаг *continue*. Однако этот флаг может быть установлен только координатором, причем только после того, как завершатся абсолютно все рабочие процессы.**
- **Таким образом, алгоритм обеспечивает синхронизацию барьера.**

Обоснование эффективности

- **В отличие от алгоритма разделяемого счетчика, каждый флаг используется лишь двумя процессами, а не всеми, вследствие чего кэш память используется более эффективно, что ускоряет работу алгоритма.**

Недостатки

- **Использование одного из процессов в качестве управляющего → координатору желательно иметь отдельный процессор для исполнения – нужно для реализации активного ожидания.**
- **Все процессы в итерационных алгоритмах выполняются идентичные действия, поэтому рабочие процессы обычно подходят к барьеру примерно в одно время, однако для установки флагов координатор будет последовательно ожидать завершения каждого из них.**

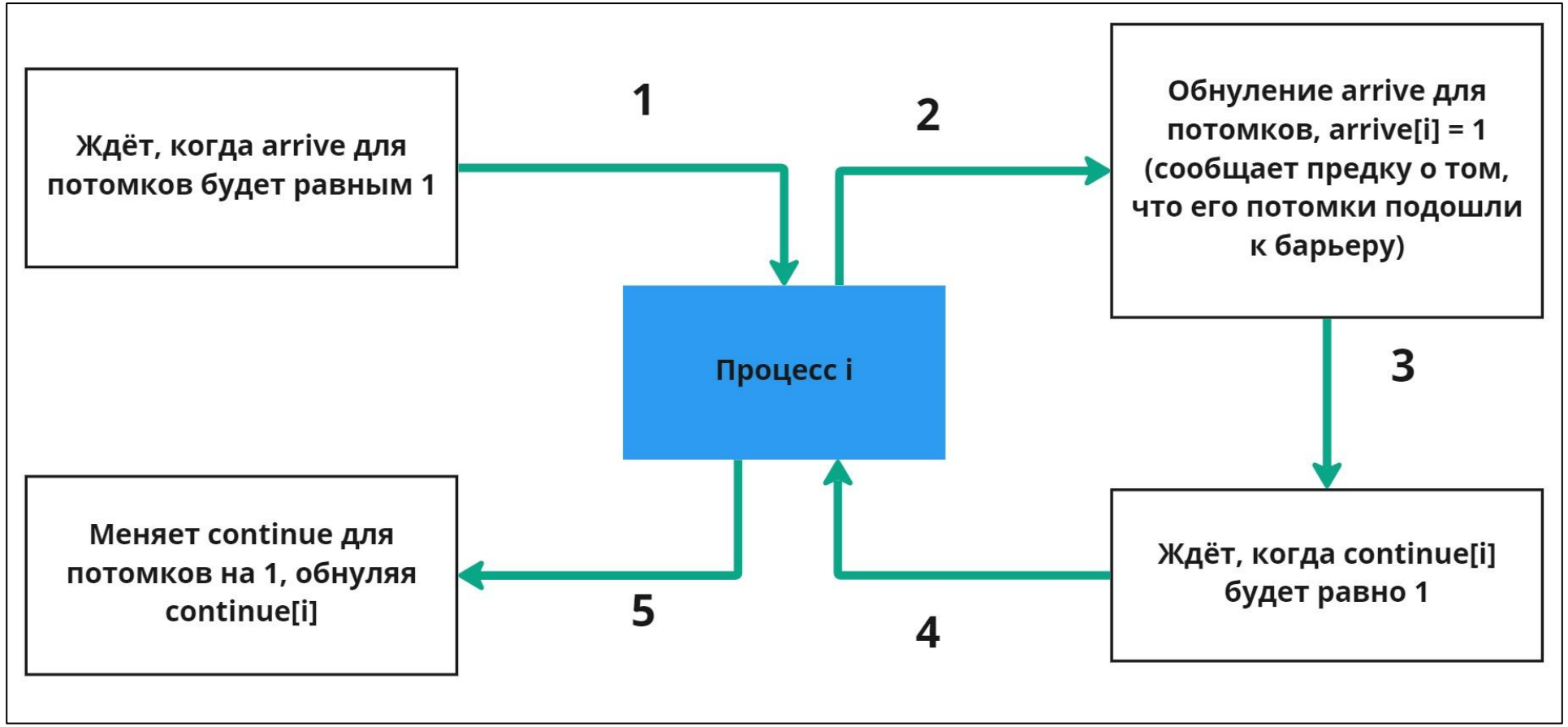
Флаговая синхронизация при помощи дерева

- **Для эффективной синхронизации объединим процессы в дерево, таким образом каждый из них будет выполнять роль рабочего и управляющего.**
- **Будем отсылать сигнал о подходе к барьеру (arrive) вверх по дереву, а сигнал о разрешении продолжения выполнения или завершния (continue) - вниз**

Флаговая синхронизация при помощи дерева

```
узел-лист L: arrive[L] = 1;  # сигнал о том, что процесс подошёл к барьеру
             ⟨await (continue[L] == 1);⟩  # активное ожидание
             continue[L] = 0;  # продолжение программы
промежуточный узел I: ⟨await (arrive[left] == 1);⟩
                      arrive[left] = 0;
                      ⟨await (arrive[right] == 1);⟩
                      arrive[right] = 0;
                      arrive[I] = 1;  # процесс подошёл к барьеру – сигнал вверх по дереву
                      ⟨await (continue[I] == 1);⟩
                      continue[I] = 0;  # сигнал о разрешении продолжения выполнения, вниз по дереву
                      continue[left] = 1; continue[right] = 1;
корневой узел R: ⟨await (arrive[left] == 1);⟩
                 arrive[left] = 0;
                 ⟨await (arrive[right] == 1);⟩
                 arrive[right] = 0;
                 continue[left] = 1; continue[right] = 1;  # сыновья могут продолжить выполнение
```

Схема флаговой синхронизация для реализации барьера



Эффективность решения

- **Нет необходимости использовать дополнительный процесс в качестве координатора**
- **Избегается конфликт обращения к памяти**
- **Высота дерева пропорциональна log2(n) (время значительно меньше при больших n) при использовании такого же размера памяти, какой бы использовался при реализации при помощи координатора**
- **Отсутствие взаимных блокировок (флаг arrive снимается до установления флага continue)**

Материалы:

- **3.4.2 Флаги и управляющие процессы // Грегори Р. Эндрюс - Основы многопоточного, параллельного и распределенного программирования (дата обращения: 18.12.2022).**
- **Листинг 3.12. Барьерная синхронизация с управляющим процессом // Грегори Р. Эндрюс - Основы многопоточного, параллельного и распределенного программирования (дата обращения: 18.12.2022).**
- **3.4.3 Симметричные барьеры // Грегори Р. Эндрюс - Основы многопоточного, параллельного и распределенного программирования (дата обращения: 18.12.2022).**

Изменения

- **Версия 2**
  - **Добавил вариант второй программы**
  - **Провёл анализ второй программы**