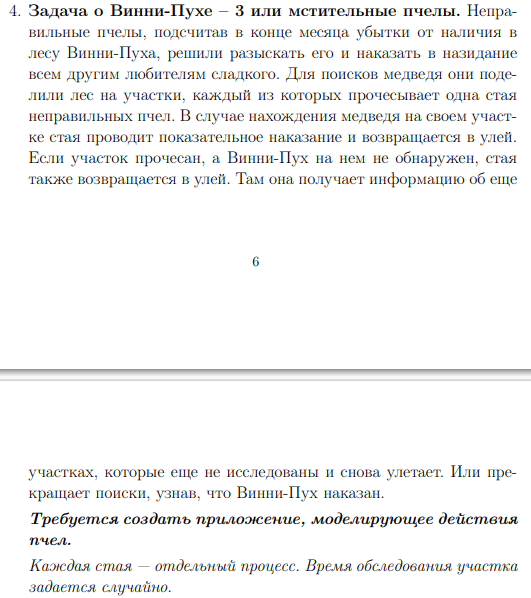
Индивидуальное домашнее задание №2.

## Мусаев Умахан Рашидович. БПИ234. Вариант 4.

Условие:



ОТЧЕТ:

Решение на 4-5(Папка solution4\_5.):  
Архитектура решения:

* Один процесс-родитель запускает множество дочерних процессов (стаи пчел)
* Механизмы IPC: Именованные POSIX семафоры и разделяемая память POSIX
* Структура программы: Монолитный исполняемый файл

Ключевые компоненты:

1. Разделяемая память хранит:
   * Общее количество участков леса
   * Количество исследованных участков
   * Индекс участка, где находится Винни-Пух
   * Флаги состояния поиска
2. Именованный семафор /winnie\_mutex\_sem используется для взаимного исключения при доступе к разделяемой памяти
3. Основные процессы:
   * Родительский процесс: Создает разделяемую память, инициализирует данные, запускает дочерние процессы
   * Дочерние процессы (стаи пчел): Исследуют участки, обновляют общую информацию через разделяемую память

Принцип работы:

1. Программа инициализирует разделяемую память и определяет участок, где находится Винни-Пух.
2. Создаются процессы для каждой стаи пчел.
3. Стаи последовательно исследуют участки:
   1. Стая берет следующий неисследованный участок
   2. Имитирует время поиска
   3. Проверяет, находится ли там Винни-Пух
   4. Обновляет счетчик исследованных участков
4. Поиск завершается, если Винни-Пух найден или все участки исследованы.

Соответствие требованиям на оценку 4-5:

* Разработано консольное приложение с родительским процессом, запускающим дочерние процессы
* Использованы именованные POSIX семафоры
* Обмен данными через разделяемую память в стандарте POSIX
* Реализовано корректное завершение программы как по условию задачи, так и по прерыванию с клавиатуры
* Предусмотрено удаление семафоров и разделяемой памяти при завершении программы

Запуск программы:

Перейдите в директорию solution4\_5:

cd solution4\_5

Запустите Makefile:

make

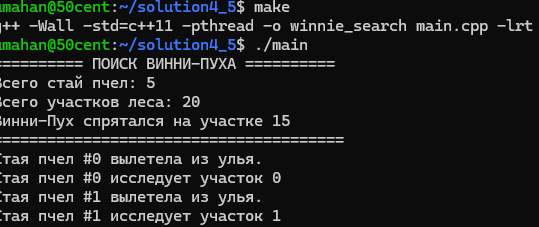
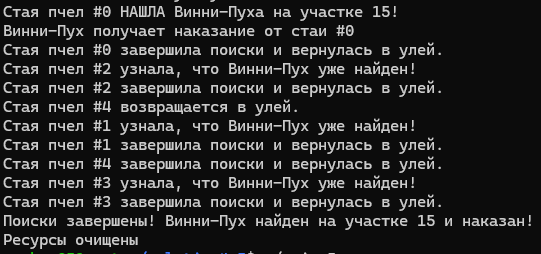
Запускаем программу:

./winnie\_search

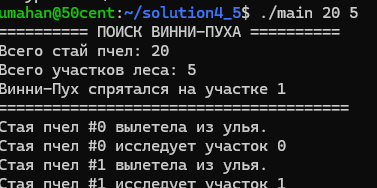
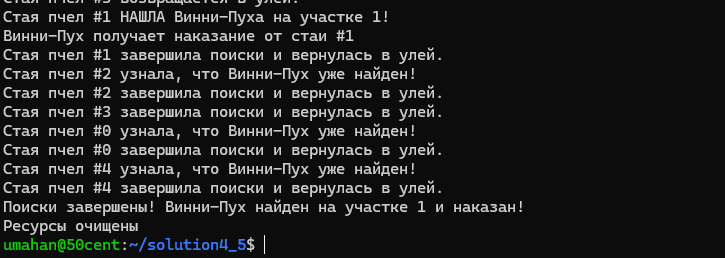
или

./ winnie\_search [количество участков] [количество стай]

Примеры выполнения программы:

ИЛИ(с параметрами)

Решение на 6-7(Папка solution6-7.):  
Архитектура решения:

Архитектура аналогична решению на 4-5 баллов, но с другим механизмом синхронизации.

Ключевые отличия от решения на 4-5 баллов:

* Использование неименованных POSIX семафоров вместо именованных
* Семафоры размещаются непосредственно в разделяемой памяти
* Инициализация и уничтожение семафоров происходит иначе (sem\_init/sem\_destroy)

Преимущества использования неименованных семафоров:

* Не требуется управление именами в файловой системе
* Автоматическое удаление при освобождении разделяемой памяти
* Семафор существует только пока есть процессы, использующие разделяемую память

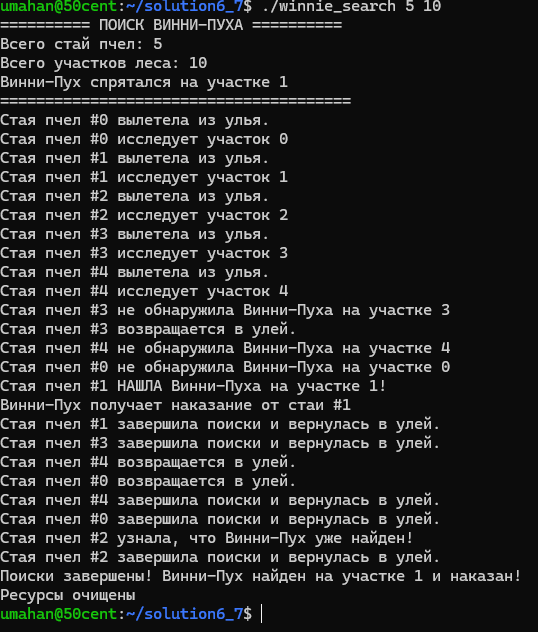
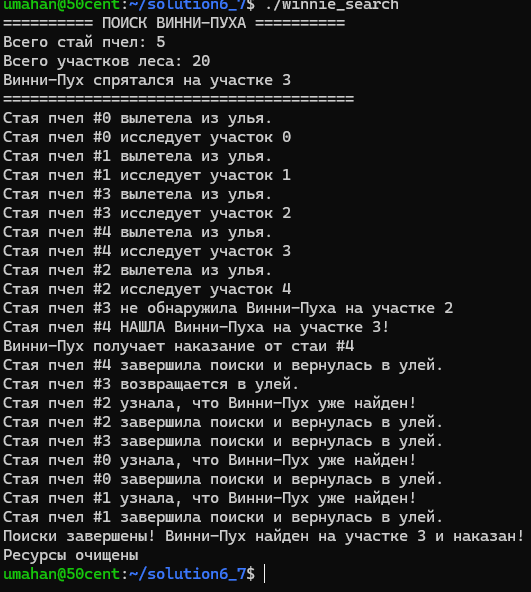
Соответствие требованиям на оценку 6-7:

* Разработана программа с альтернативным вариантом взаимодействия (неименованные POSIX семафоры)
* Обмен данными через разделяемую память в стандарте POSIX
* Использован вариант, не реализованный в предыдущей оценке

Запуск программы:

Запуск такой же, как и у решения на 4-5.

Пример выполнения программы:

Решение на 8(Папка solution8.):

Архитектура решения:

* Несколько независимых программ-процессов
* Механизмы IPC: Семафоры UNIX System V и разделяемая память UNIX System V
* Структура программы: Несколько исполняемых файлов

Ключевые компоненты:

1. hive\_server.cpp - Сервер улья:

* Создает и инициализирует разделяемую память
* Случайно выбирает участок для Винни-Пуха
* Отслеживает процесс поиска и выводит результаты

2. bee\_swarm.cpp - Стая пчел:

* Подключается к разделяемой памяти
* Исследует участки леса
* Сообщает о результатах поиска

3. Семафоры System V: используются для синхронизации доступа к разделяемой памяти

Отличия от предыдущих решений:

* Программы запускаются независимо друг от друга
* Используются IPC механизмы UNIX System V вместо POSIX
* Более сложная схема взаимодействия, так как процессы не связаны родственными отношениями

Соответствие требованиям на оценку 8:

* Разработано приложение, состоящее из отдельных программ-процессов, запускаемых независимо
* Использованы семафоры в стандарте UNIX SYSTEM V
* Обмен данными через разделяемую память в стандарте UNIX SYSTEM V

Запуск программы:  
Перейдите в директорию solution4\_5:

cd solution8

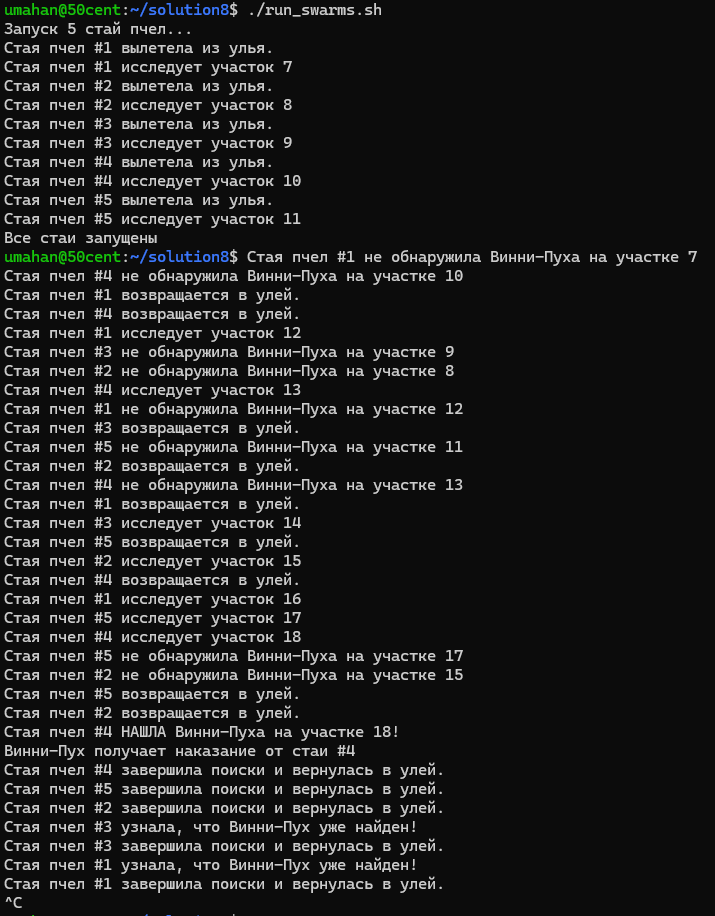
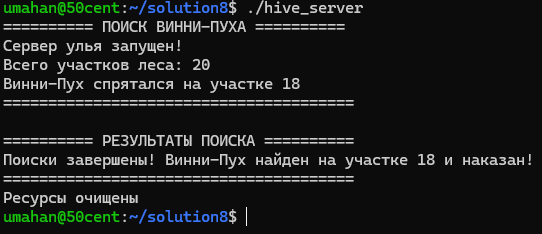
Запустите Makefile:

make

Запускаем программу:

1. В первом терминале пишем ./hive\_server или ./hive\_server [количество участков] (по умолчанию 20 участков, поэтому не обязательно указывать.)
2. Во втором терминале запускаем скрипт: ./run\_swarms.sh [количество стай] или запускаем стаи самостоятельно ./bee\_swarm 1, ./bee\_swarm 2 …

Пример выполнения программы:

 <- это запуск стаи.<- это сервер.

Для остановки скрипта нужно прописать Ctrl+C.

Решение на 9(Папка solution9.):  
Архитектура решения:

* Полностью независимые процессы
* Механизмы IPC: Семафоры UNIX System V и очереди сообщений UNIX System V
* Структура программы: Три исполняемых файла

Ключевые компоненты:

1. hive\_coordinator.cpp - Координатор улья:

* Инициализирует параметры поиска
* Координирует общий процесс
* Обрабатывает сообщения о результатах поиска

2. hive\_dispatcher.cpp - Диспетчер улья:

* Распределяет участки между стаями
* Передает информацию между координатором и стаями
* Отслеживает исследованные участки

3. bee\_swarm.cpp - Стая пчел:

* Запрашивает участки у диспетчера
* Исследует назначенные участки
* Сообщает о результатах поиска

Особенности реализации:

* Архитектура клиент-сервер: диспетчер выполняет роль сервера для стай
* Очереди сообщений System V для передачи запросов и результатов
* Семафоры System V для синхронизации доступа к общим данным
* Типизированные сообщения для маршрутизации между процессами

Преимущества:

* Процессы могут запускаться и завершаться независимо
* Более гибкая модель коммуникации, чем простая разделяемая память
* Полноценный обмен структурированными сообщениями

Соответствие требованиям на оценку 8:

* Разработано приложение из независимых программ-процессов
* Использованы семафоры UNIX SYSTEM V
* Использованы очереди сообщений UNIX SYSTEM V для взаимодействия
* Сохранены подходы для завершения приложения из предыдущих решений

Запуск программы:  
Перейдите в директорию solution9:

cd solution9

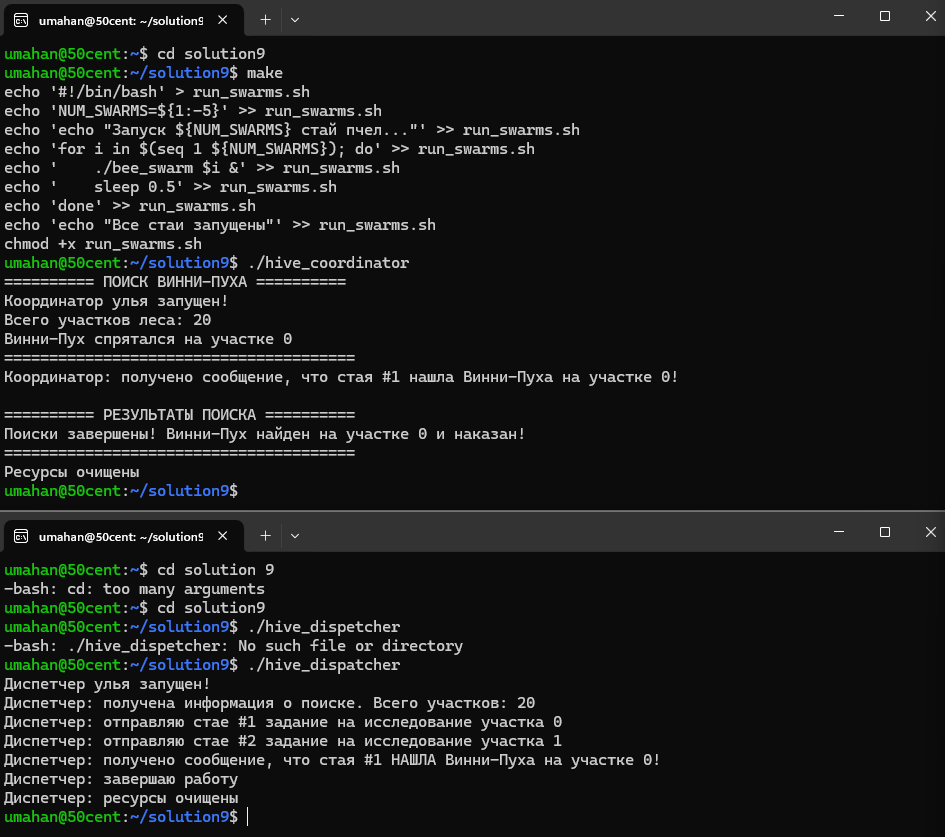
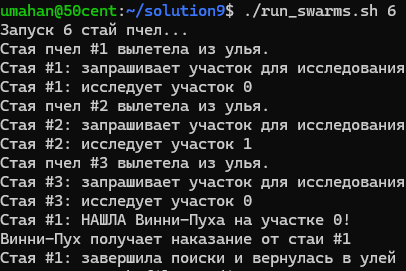
Запустите Makefile:

make

Запускаем программу:

1. В первом терминале пишем ./hive\_coordinator или ./hive\_coordinator[количество участков] (по умолчанию 20 участков, поэтому не обязательно указывать.)
2. Во втором терминале запускаем диспетчер, пишем ./hive\_dispatcher
3. Во втором терминале запускаем стаи пчел с помощью скрипта (с указанием количества стай): ./run\_swarms.sh [количество стай]

Пример выполнения программы:

На картинках видно работу координатора, диспетчера и стаи.

Для остановки скрипта нужно прописать Ctrl+C

Решение на 10(Папка solution10.):

Для оценки 10 разработано приложение с использованием оставшегося варианта взаимодействия.

Архитектура решения:

Архитектура аналогична решению на 9 баллов, но с использованием POSIX-механизмов IPC вместо System V.

Ключевые компоненты:

1. hive\_coordinator.cpp - Координатор улья
2. hive\_dispatcher.cpp - Диспетчер улья
3. bee\_swarm.cpp - Стая пчел

Особенности реализации:

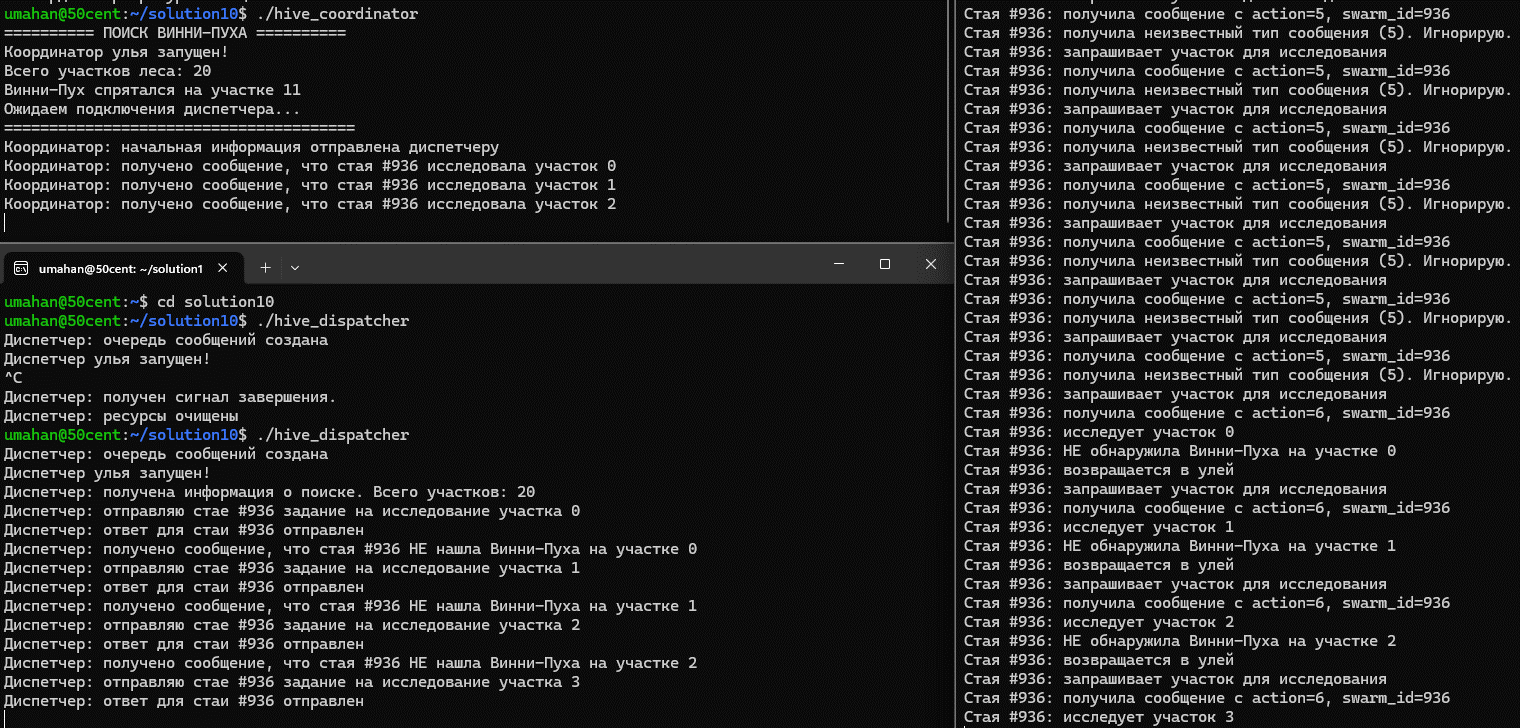
* Именованные POSIX семафоры для синхронизации доступа к общим данным
* Очереди сообщений POSIX для передачи запросов и результатов
* Фильтрация сообщений по идентификатору стаи
* Механизм повторных попыток для надежного подключения к очередям

Отличия от решения на 9 баллов:

* Использование POSIX API вместо System V API
* Объекты IPC именуются как пути в файловой системе
* Отличающиеся функции API для работы с очередями сообщений

Запуск программы:  
Запуск аналогичен решению на 9.

Пример работы:



На картинке три терминала (координатор, диспетчер, стаи), можно видеть, что Винни Пух удачно найден.