НИУ ВШЭ

Архитектура вычислительных систем

Индивидуальное домашнее задание №4

Вариант 15

Студент: Лебедев Андрей Андреевич БПИ234

Москва
2024

Цель задачи

Разработать многопоточное приложение для моделирования работы больницы. В приложении должны быть реализованы два дежурных врача, которые принимают пациентов и направляют их к одному из трех специалистов: стоматологу, хирургу или терапевту. Каждый специалист лечит пациентов по очереди, и программа должна обеспечивать правильную синхронизацию потоков.

Предметная область

В больнице работают:

- **Дежурные врачи**, которые принимают пациентов и направляют их к специалистам.
- **Специалисты** (стоматолог, хирург и терапевт), которые лечат пациентов по очереди.
- **Пациенты**, которые в ожидании получения лечения становятся в очередь.

Сценарий взаимодействия сущностей

Описание роли каждого субъекта в задаче:

- **Пациенты** — пациенты приходят в больницу и ждут приёма у дежурного врача. Каждый пациент после принятия дежурным врачом отправляется к одному из специалистов. После лечения пациент выписывается.
- **Дежурные врачи** — два дежурных врача, каждый из которых принимает пациентов по очереди, выслушивает их жалобы и направляет к специалистам. Для выбора специалиста используется случайный выбор.
- **Специалисты (стоматолог, хирург, терапевт)** — каждый специалист может лечить только одного пациента за раз. Пациенты поступают в очередь соответствующего специалиста и лечатся по порядку.

Взаимодействие сущностей:

1. Пациент приходит в больницу и встает в общую очередь к дежурному врачу.
2. Один из дежурных врачей принимает пациента из общей очереди, выслушивает жалобы и случайным образом отправляет его к одному из специалистов.
3. Пациент переходит в очередь к выбранному специалисту и ждет своей очереди.
4. Специалист лечит пациента по очереди, имитируя лечение с помощью задержки.
5. После лечения пациент выписывается, и его лечащий врач сообщает об этом.

Каждая сущность (пациент, дежурный врач и специалист) работает в своем потоке, и синхронизация потоков осуществляется с помощью мьютексов и условных переменных, чтобы избежать гонок и конфликтов при доступе к общим данным.

Модель параллельных вычислений

Для реализации параллельных вычислений используется модель многозадачности, где каждый субъект — пациент, врач и специалист — выполняется в отдельном потоке. Потоки взаимодействуют через очередь пациентов, которая защищена мьютексами, чтобы избежать одновременного доступа к данным.

Основные компоненты программы:

- **Потоки пациентов** — каждый пациент представляет собой отдельный поток, который генерирует свой ID и добавляется в общую очередь. Поток пациента завершает свою работу, когда он будет направлен к специалисту и вылечен.
- **Потоки дежурных врачей** — два дежурных врача работают параллельно, они забирают пациентов из общей очереди и отправляют их к специалистам.
- **Потоки специалистов** — три специалиста (стоматолог, хирург и терапевт) принимают пациентов по очереди из своих специализированных очередей и лечат их.

Синхронизация потоков:

- **Мьютексы** — используются для защиты данных, таких как очереди пациентов, от одновременного доступа.
- **Условные переменные** — используются для ожидания наличия пациентов в очереди и синхронизации работы между потоками, особенно для специалистов, которые должны ждать появления пациентов в своей очереди.

Входные данные программы

Программа принимает следующие входные данные:

- **Количество пациентов** — число пациентов, которые будут обслужены в рамках программы. Это значение можно задать через консольный ввод или через конфигурационный файл.
- **Конфигурационный файл (необязательно)** — если передан флаг `-f`, программа загружает число пациентов из конфигурационного файла.

Диапазоны входных данных:

- **NUM_PATIENTS** — диапазон значений от 1 до 1000 пациентов.
- **Конфигурационный файл** — должен содержать одно целое число, представляющее количество пациентов.

Пример конфигурационного файла:

```
10
```

Если конфигурационный файл не указан, то количество пациентов будет задано пользователем через консоль.

Реализация консольного приложения

Программа написана на языке C++ с использованием библиотеки POSIX Threads (pthread) для работы с потоками. В процессе выполнения программы пациенты, врачи и специалисты выполняются в отдельных потоках.

Программа предоставляет следующую функциональность:

- Ввод числа пациентов с консоли или из конфигурационного файла.
- Синхронизация работы потоков с помощью мьютексов и условных переменных.
- Вывод всех ключевых событий в консоль и в файл:
  - Принятие пациента дежурным врачом.
  - Направление пациента к специалисту.
  - Лечение пациента специалистом.
  - Завершение лечения пациента.

Варианты запуска приложения:

- Использование параметров по умолчанию. Значение NUM_PATIENTS вводится в консоль, выходной файл - output.txt

```
.\main.exe
```

- Передача значения NUM_PATIENTS в качестве аргумента

```
.\main.exe 25
```

- Чтение конфигурации и вывод в файл

```
.\main.exe -f config.txt -o output.txt
```

Вывод программы

Программа выводит подробную информацию о каждом важном событии, происходящем в больнице. Каждое действие сопровождается логом, который отображается на консоли и записывается в файл.

Пример вывода программы:

```
Please, enter NUMBER OF PATIENTS: 5
Number of patients: 5
Patient 1 arrives at the hospital.
Duty Doctor 1 sends Patient 1 to Dentist.
Dentist is treating Patient 1.
Dentist has finished treating Patient 1.
...
_____END_OF_PROGRAM_____
```

Пример вывода в файл:

```
Duty Doctor 1 sends Patient 1 to Dentist
Dentist is treating Patient 1
Dentist has finished treating Patient 1
...
```

Вывод включает информацию о том, какие врачи и специалисты работают с пациентами и как выполняются их действия в ходе работы программы.

Альтернативное решение задачи

В дополнение к основной программе было разработано альтернативное решение, использующее другие синхропримитивы, а именно `condition variables` (условные переменные), наряду с мьютексами для синхронизации потоков. В отличие от предыдущей программы, в этом решении используются `condition variables` для координации потоков специалистов и дежурных врачей. Основные изменения заключаются в следующем:

- **Использование условных переменных:** В отличие от первоначальной программы, где использовались только мьютексы и семафоры, в этой программе мьютексы сочетаются с условными переменными. Каждому специалисту (стоматологу, хирургу и терапевту) и дежурным врачам соответствуют свои условные переменные, которые управляют синхронизацией между потоками.
- **Новые условия для синхронизации:** При отправке пациента к специалисту, дежурный врач сигнализирует соответствующей условной переменной, чтобы специалист начал лечение пациента. Аналогично, каждый специалист использует условную переменную, чтобы ожидать появления пациента в своей очереди.
- **Логика работы потоков:** Все потоки, включая пациентов, дежурных врачей и специалистов, синхронизируются через `mutexes` и `condition variables`. Каждый поток дожидается своей очереди на выполнение, что позволяет избежать блокировок и гонок данных.

Сравнительный анализ с предыдущим решением:

- В обоих решениях используется многозадачность с потоками, но в альтернативном решении добавлены условные переменные для более гибкой синхронизации. Это позволяет улучшить производительность при больших объемах данных, так как потоки могут более эффективно ожидать события (например, когда очередь у специалиста пуста, поток будет приостановлен, пока не появится пациент).
- Оба решения обеспечивают идентичность поведения при одинаковых входных данных. В обоих случаях пациенты обрабатываются по очереди, каждый пациент получает лечение у одного из специалистов, и программа завершает работу после того, как все пациенты будут вылечены.
- Основное отличие заключается в том, что в альтернативном решении использованы `condition variables`, что улучшает гибкость синхронизации и позволяет уменьшить нагрузку на процессор при ожидании.

Результаты изменений:

Программа с условными переменными работает так же корректно, как и первоначальная версия, и результаты идентичны при одинаковых данных. Однако, благодаря использованию `condition`

`variables`, программа может работать более эффективно при большом числе пациентов, так как потоки не будут активно блокировать ресурсы, а будут ожидать необходимого события.

Пример вывода для альтернативной программы:

```
Please, enter NUMBER OF PATIENTS: 5
Number of patients: 5
Patient 1 arrives at the hospital.
Duty Doctor 1 sends Patient 1 to Dentist.
Dentist is treating Patient 1.
Dentist has finished treating Patient 1.
...
_____END_OF_PROGRAM_____
```

Альтернативное решение с использованием условных переменных продемонстрировало улучшение синхронизации потоков по сравнению с первоначальной программой. Оба решения дают одинаковые результаты, но новое решение использует более эффективные методы синхронизации, что может быть полезно при увеличении числа пациентов. Все пациенты были приняты, направлены к специалистам и успешно вылечены в обоих решениях.

Заключение

Решение задачи о больнице с использованием многозадачности и синхронизации потоков на базе POSIX Threads является успешным. Программа моделирует работу больницы, эффективно управляя потоками пациентов, врачей и специалистов, а также корректно синхронизируя их действия для обеспечения правильного функционирования.