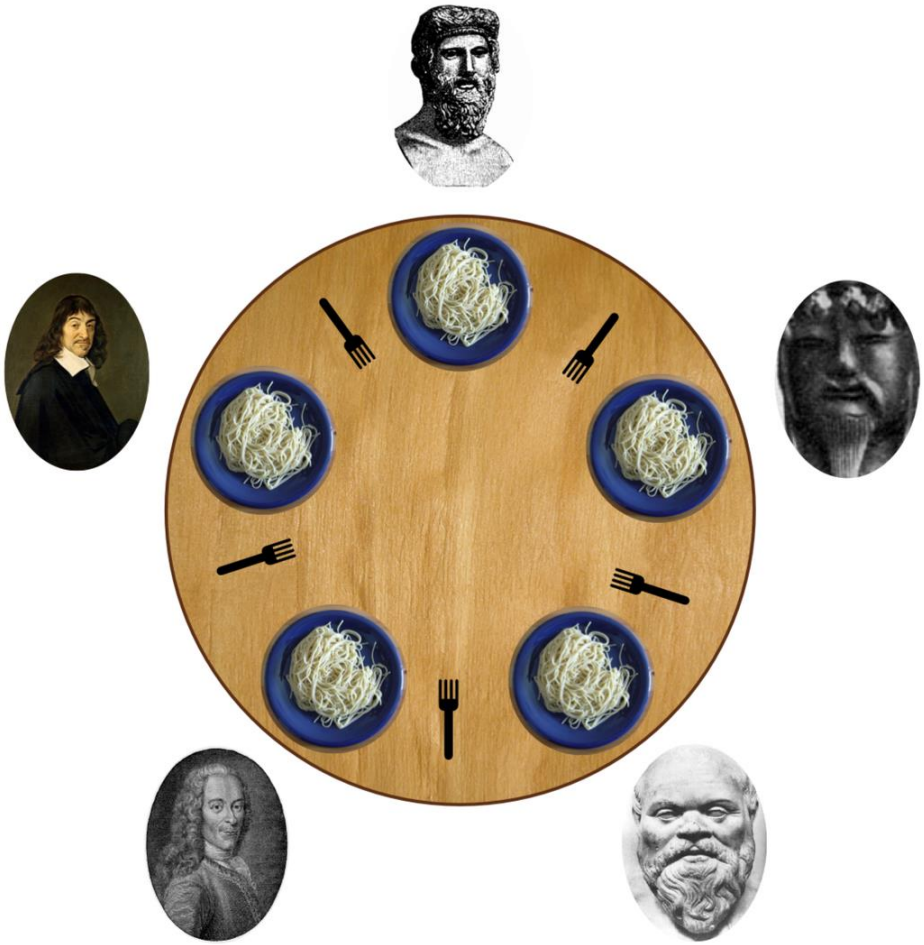
Уральский федеральный университет

Анализ первого домашнего задания

Созыкин Андрей Владимирович

к.т.н.
Заведующий кафедрой высокопроизводительных компьютерных технологий
Институт математики и компьютерных наук

Обедающие философы



Обедающие философы

Написать многопоточную программу моделирующую задачу обедающих философов. Требования:

- Избежать взаимоблокировок (deadlock)
- Избежать голодания (livelock)
- Обеспечить равномерное «питание» всех философов

Взаимоблокировки

Как обнаружить?

Взаимоблокировки

Как обнаружить?

- Убрать задержку во время «думания»

Взаимоблокировки

Как обнаружить?

- Убрать задержку во время «думания»

Взаимоблокировки у 1 человека

Livelock

Встречается в 1 решении

Схема решения

- Берем левую вилку
- Пытаемся взять правую вилку
- Если не получается, ждем одинаковый для всех интервал времени (sleep(10))

Равномерное распределение еды

Неравномерное распределение еды – ошибка, которая встречается чаще всего

Основная проблема:

- Тестирование на малом количестве философов (5 шт.)
- При увеличении количества философов (20 или 100) начинает проявляться неравномерность

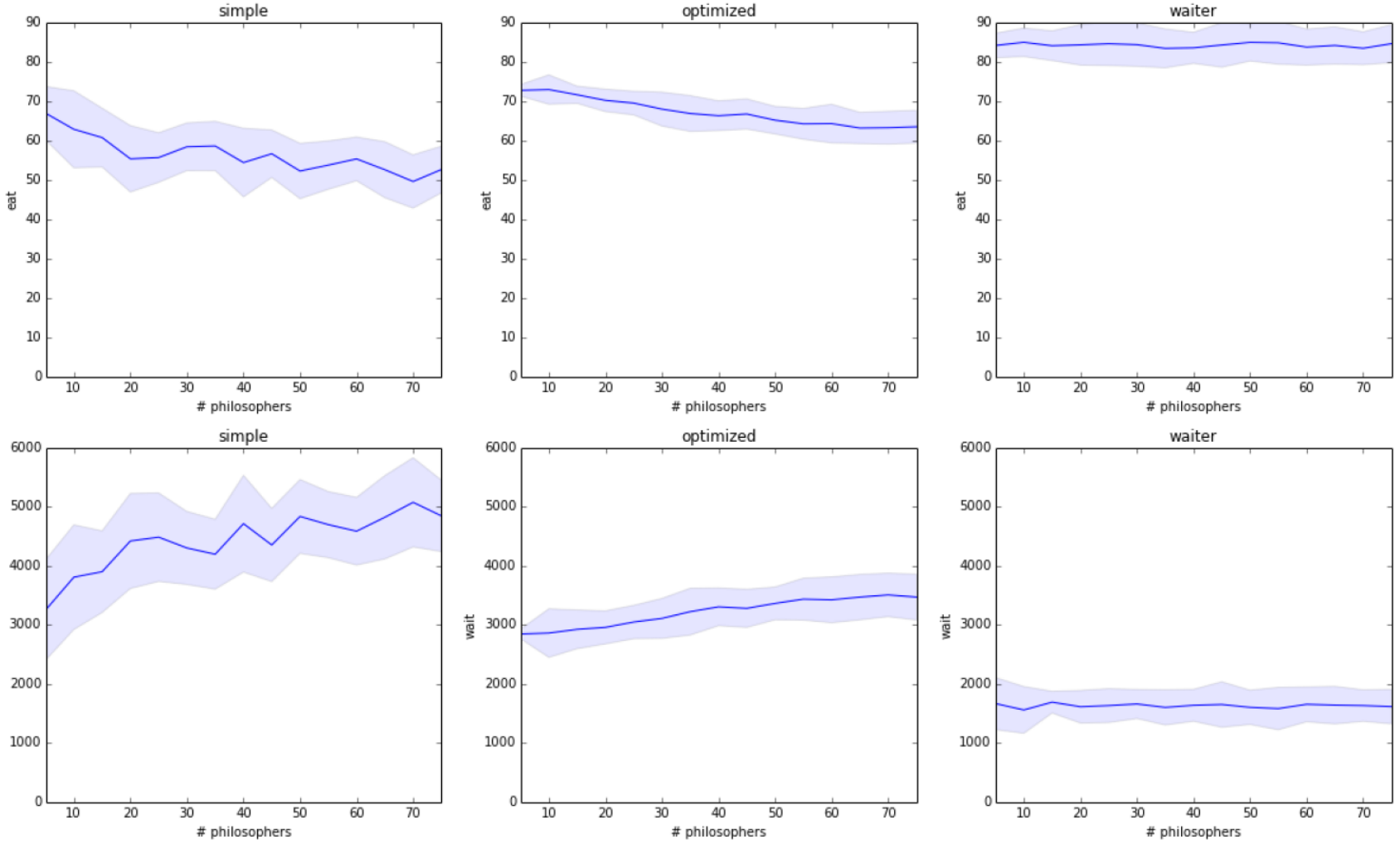
Хорошее решение:

- Реализовано равномерное распределение еды
- Проведено тестирование на разном количестве философов, результаты представлены в отчете

Равномерное распределение еды

Параметры	Реализация 1	Реализация 2
./phil 5 20 100 100 0	[1] 128 7385 [2] 127 7212 [3] 129 7580 [4] 124 7151 [5] 136 7409 среднее: 128 вариация: 3.5%	[1] 131 6949 [2] 142 5965 [3] 141 6544 [4] 142 6590 [5] 134 5298 среднее: 138 вариация: 3.7%
./phil 25 20 100 100 0	[1] 107 10120 [2] 103 10163 [3] 104 10007 [4] 98 10389 [5] 97 10750 [6] 94 10525 [7] 91 10474 [8] 89 11003 [9] 89 10609 [10] 93 10407 [11] 94 10779 [12] 90 10514 [13] 90 10265 [14] 97 10877 [15] 95 10356 [16] 96 10640 [17] 100 10503 [18] 97 10701 [19] 94 10507 [20] 97 9961 [21] 95 10106 [22] 105 10108 [23] 104 10127 [24] 102 9604 [25] 103 9631 среднее 97 вариация: 5.5%	[1] 148 5953 [2] 129 7124 [3] 149 5505 [4] 140 5766 [5] 136 5910 [6] 140 5646 [7] 136 7127 [8] 147 5487 [9] 139 6165 [10] 136 6567 [11] 144 5433 [12] 135 6716 [13] 141 5435 [14] 131 7274 [15] 142 5236 [16] 137 6661 [17] 139 5784 [18] 132 6784 [19] 138 5873 [20] 138 6551 [21] 143 6019 [22] 138 6273 [23] 133 6479 [24] 138 6506 [25] 144 5723 среднее 138 вариация: 3.7%
./phil 100 20 100 100 0	… среднее: 94 вариация: 4.5%	… среднее: 140 вариация: 3.8%

Равномерное распределение еды



Варианты решения задачи

Мало тривиальных решений

Иерархия ресурсов

Случайное время ожидания при захвате вилок

Официант (стол)

Общение между философами

Вариант 1. Иерархия ресурсов

Вилки пронумерованы от 0 до 4 (Количество философов-1)

Философ:

1. Всегда берёт сначала вилку с наименьшим номером, а потом вилку с наибольшим номером.
2. Кладёт сначала вилку с бóльшим номером, потом — с меньшим.

Вариант 1. Иерархия ресурсов

Отсутствие взаимоблокировки:

- Если четыре из пяти философов одновременно возьмут вилку с наименьшим номером, на столе останется вилка с наибольшим возможным номером.
- Пятый философ не сможет взять ни одной вилки
- Только один философ будет иметь доступ к вилке с наибольшим номером, так что он сможет есть двумя вилками.

Отсутствие голодания:

- Философ не отпустит вилку, пока не поест.

Вариант 1. Примеры работы

```
[1] 352 24197
[2] 391 20522
[3] 420 18173
[4] 455 14207
[5] 361 23043
```

```
[1] 360 24630
[2] 393 21596
[3] 414 19208
[4] 464 14066
[5] 385 23683
```

```
[1] 356 24730
[2] 406 19875
[3] 417 19256
[4] 467 13896
[5] 370 23556
```

Вариант 1. Равномерное «питание»

Философы едят одной и той же вилкой по очереди

В Вилке запоминаем номер Философа, который брал ее последним

Философ не может взять вилку при двух условиях:

- Сейчас очередь есть другого философа
- Другой философ голодный

Вариант 1. Примеры работы

```
[0] 375 22956 ms
[1] 374 21965 ms
[2] 378 20791 ms
[3] 384 21821 ms
[4] 390 20543 ms
```

Вариант 1. Примеры работы

```
[0] 370 24251 ms
[1] 363 22526 ms
[2] 363 23378 ms
[3] 364 23916 ms
[4] 365 22666 ms
[5] 377 23963 ms
[6] 377 22342 ms
[7] 372 23131 ms
[8] 363 23245 ms
[9] 356 22188 ms
[10] 363 23709 ms
[11] 370 22672 ms
[12] 377 22004 ms
[13] 387 20819 ms
[14] 385 21758 ms
```

Вариант 1. Упорядочиваем философов

Нумеруем философов

Четные философы берут сначала правую вилку - потом левую, нечетные наоборот

Равномерное «питание»:

- Философ проверяет, голодный ли сосед
- Если голодный, задержка (поток засыпает)

Вариант 2. Случайное время ожидания

Философ пытается захватить **левую** вилку в течение **случайного интервала времени**

- Если время закончено, философ отпускает вилку

Философ пытается захватить **правую** вилку в течение **случайного интервала времени**

- Если время закончено, философ отпускает вилку

В случае успешного захвата вилок философ ест

Если вилки захватить не удалось, задержка

- Случайное время, увеличивающееся экспоненциально с номером попытки (exponential backoff)

Вариант 2. Случайное время ожидания

```
[Philosopher 0] ate 414 times and waited 19376 ms
[Philosopher 1] ate 417 times and waited 18731 ms
[Philosopher 2] ate 412 times and waited 19222 ms
[Philosopher 3] ate 411 times and waited 19248 ms
[Philosopher 4] ate 420 times and waited 17592 ms
```

Вариант 2. Случайное время ожидания

```
[Philosopher 0] ate 425 times and waited 18846 ms
[Philosopher 1] ate 418 times and waited 17155 ms
[Philosopher 2] ate 396 times and waited 18403 ms
[Philosopher 3] ate 421 times and waited 17372 ms
[Philosopher 4] ate 437 times and waited 17585 ms
[Philosopher 5] ate 421 times and waited 19313 ms
[Philosopher 6] ate 429 times and waited 16575 ms
[Philosopher 7] ate 412 times and waited 19679 ms
[Philosopher 8] ate 423 times and waited 17378 ms
[Philosopher 9] ate 411 times and waited 19184 ms
[Philosopher 10] ate 421 times and waited 17522 ms
[Philosopher 11] ate 419 times and waited 20164 ms
[Philosopher 12] ate 434 times and waited 17536 ms
[Philosopher 13] ate 423 times and waited 18278 ms
[Philosopher 14] ate 428 times and waited 19173 ms
```

Вариант 3. Официант

Проголодавшийся философ обращается к официанту

Официант ставит запрос в очередь

- Проверяет, едят ли соседи
- Если не едят, то разрешает философу есть
- Если едят, то не разрешает
- Философ ждет некоторое время и повторяет запрос

Вариант 3. Официант

```
[1] 185 41300
[2] 186 41554
[3] 188 42278
[4] 187 41882
[5] 189 42383
[6] 188 42433
[7] 187 41404
[8] 191 42341
[9] 186 41636
[10] 189 42489
[11] 188 42671
[12] 189 42612
[13] 189 42357
[14] 191 42059
[15] 190 42322
```

Вариант 3. Официант

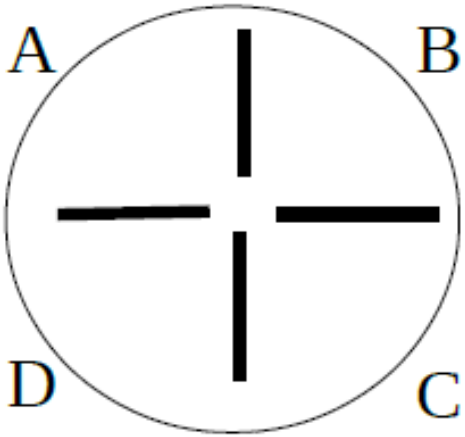
Официант «пускает» к столу N-1 философов

Реализация официанта в виде семафора

Вариант 3. Официант

Официант «пускает» к столу N-1 философов

Реализация официанта в виде семафора



Вариант 4. Общение между философами

The Drinking Philosophers Problem Chandy, K.M.; Misra, J. (1984).

Каждая вилка “принадлежит” некоторому философу и может быть либо чистой, либо грязной.

В начальный момент времени все вилки грязные.

Начать прием пищи можно только чистой вилкой.

После того, как философ поел, обе его вилки становятся грязными.

Вариант 4. Общение между философами

Когда философ собрался поесть он “заполучает” обе вилки:

1) Если вилка принадлежит ему и является грязной, философ моет ее.

2) Если вилка в данный момент времени принадлежит его соседу, то философ “просит” его отдать ему вилку. После того, как сосед закончит прием пищи, он сделает вилку чистой и передаст философу.

3) Философ не отдает вилку, вилку, которой обладает, пока вилка не станет грязной (то есть пока философ не поест ей хотя бы раз с момента получения).

Вариант 4. Общение между философами

При таком подходе преимущество всегда получает наиболее голодный философ:

- если философ обладает чистой вилкой, значит он проголодался и помыл ее раньше, чем у него ее попросили.
- если же вилка на момент запроса грязная, то запрашивающий философ проголодался раньше.

Crawler

Многопоточные структуры данных:

- Многопоточная очередь (ссылки для скачивания)
- Многопоточное множество (запоминание скачанных страниц)

Счетчик скаченных страниц:

- Защита mutex
- Атомарные переменные

Потоки:

- Пул потоков
- vector (массив) с потоками

Отдельно счетчик и mutex

```
void Crawler::run_single_thread(UrlQueue &url_queue, FileStorage &file_storage,
    const unsigned max_num_pages, std::mutex &num_pages_mutex,
    unsigned &num_pages) {
...
    std::unique_lock<std::mutex> num_pages_lock(num_pages_mutex);
    if (num_pages >= max_num_pages) {
        return;
    }
    const unsigned id = num_pages;
    ++num_pages;
    num_pages_lock.unlock();
...
}
```

Инициализация libcurl

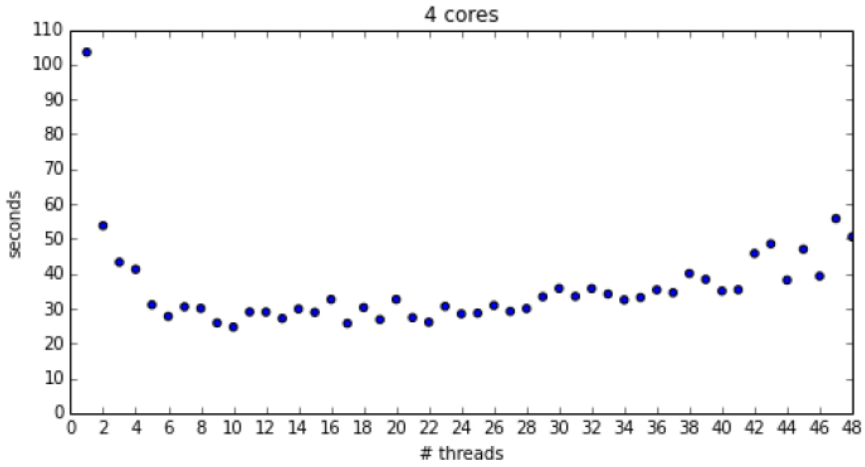
```
/* Must initialize libcurl before any threads are started */
curl_global_init(CURL_GLOBAL_ALL);


/* Thread code */
CURL *curl;
curl = curl_easy_init();
```

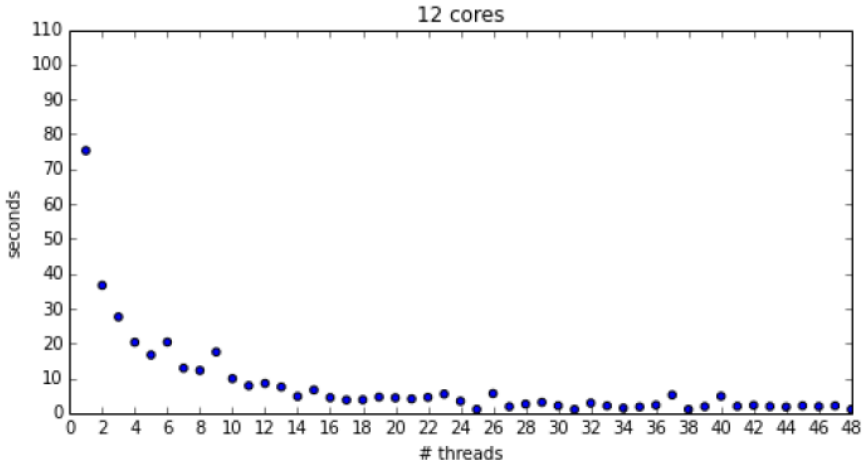
If you did not already call curl_global_init, curl_easy_init does it automatically. This may be lethal in multi-threaded cases, since curl_global_init is not thread-safe, and it may result in resource problems because there is no corresponding cleanup.

http://curl.haxx.se/libcurl/c/multithread.html

http://curl.haxx.se/libcurl/c/curl_easy_init.html

Отчеты.


4 cores
seconds
threads
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48
0 10 20 30 40 50 60 70 80 90 100 110


12 cores
seconds
threads
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48
0 10 20 30 40 50 60 70 80 90 100 110

Отчеты. Гипотеза

Ясно, что при использовании N таких потоков время работы программы сократится в лучшем случае в N раз.

Это справедливо только в том случае, когда число вычислительных ядер не меньше числа потоков: если число потоков окажется больше, там прироста производительности происходить не будет, и начиная с некоторого момента накладные расходы на переключение контекста станут велики и начнётся замедление.

Домашнее задание 3. Задача 1

Анализ графа пользователей Твиттера

- Данные: (user_id, follower_id)
- Вычислить распределение количества подписчиков
- Определить Top50 пользователей по количеству подписчиков
- Вычислить среднее количество подписчиков

Домашнее задание 3. Задача 2

Построить инвертированный индекс для русской и английской Википедий

- Данные: (docid, content)
- Формат индекса:

  (word, [<docid1, tf-idf1>, <docid2, tf-idf2>, …])
- Статьи должны быть отсортированы в порядке убывания tf-idf
- Ограничить список N наиболее релевантными статьями
- Исключить из индекса Top20 высокочастотных слов

Вопросы?