Звіт

до лабораторної роботи 2

з навчальної дисципліни «Операційні системи»

студента 3 курсу ФКНК групи МІ-32

Красікова Антона Ігоровича

«Dining philosophers problem»

Умова завдання:

Розробити та запрограмувати алгоритм вирішеня задачі про філософів, що обідають – https://en.wikipedia.org/wiki/Dining_philosophers_problem.

Рішення має враховувати:

ексклюзивність ресурсів (виделку як ресурс може взяти лише один (з двох adjacent neighboring) філософ у кожний момент часу, і тільки якщо вона вільна)

deadlock-free (щоб система не могла стати в тупик)

livelock-free (щоб філософи не померли від голодування, коли їм довго не вдається взяти виделки, бо вони якимось чином блокуються сусідами), ситуація нескінченного очікування

Перевага надається рішенням, які:

реалізують однаковий алгоритм дій усіх філософів (не залежать від номера філософа, тощо)

не мають централізованої сутності (офіціанта, оракула, тощо), який керує розподілом ресурсів-виделок

не передбачають комунікацію між філософами (щоб вони домовились про виделки чи інші спільні дії)

є масштабованим на будь-яку кількість філософів (і виделок відповідно) і не мають очевидних bottlenecks

можна довести формально їх коректність і виконуваність всіх вказаних вимог

Всі програми-філософи мають діяти за однаковим алгоритмом. Всі вони рівноправні (і мають бути взаємозамінні), жоден не має пріоритету над іншими.

Хід роботи:

У даному випадку маємо задачу з захопленням ресурсів, в якій можливе взаємоблокування (клінч), коли один потік захопив ресурс A та чекає звільнення ресурсу B, у той час як інший потік захопив ресурс B та чекає звільнення ресурсу A.

Теорія каже, що коли ресурси захоплюються в однаковому порядку (тобто ресурси можуть бути впорядковані, і жоден потік не може захопити спочатку ресурс з більшим номером, а потім з меншим), то взаємоблокування неможливе.

Скористаємося цим для рішення поставленої задачі.

Кожного філософа реалізовано як клас, об’єкт якого має стан (думати; чекати на виделки; їсти; покласти виделки на стіл та думати далі). Крім того, у кожного об’єкта є змінні, що позначають номери виделок ліворуч та праворуч.

Виделка — це просто м’ютекс, точніше (оскільки м’ютекси не копіюються) — вказівник на м’ютекс. Всі вони розміщуються у векторі, а номер виделки — це індекс в векторі.

Коли ми саджаємо філософа за стіл на місце `i`, у нього будуть поруч виделки `i` та `(i+1)%N`. Оскільки може здатися, що в цьому є якась залежність від номера філософа, в програмі є можливість перед тим, як садити філософів за стіл, перетасувати вилки, що доводить, що алгоритм **не** залежність від номера філософа.

Отже, маємо

```
int Count = 5;          // Кількість філософів - змінювана
int Timer = 2500;       // Максимальній час роздумів/їжі

vector<Philosopher*> table;    // Філософи
vector<mutex*>       forks;    // Виделки
vector<int>          forksNo;  // Нумерація виделок
```

Перед тим, як розсаджувати філософів, ми перераховуємо виделки

```
for(int i = 0; i < Count; ++i) forksNo.push_back(i);
```

та за необхідності перетасовуємо їх

```
if (FORKS_SHUFFLE) shuffle(forksNo.begin(),forksNo.end(),reng);
```

Після цього ми розкладаємо виделки та розсаджуємо філософів

```
    for(int i = 0; i < Count; ++i)
    {
        table.push_back(new Philosopher(i));
        forks.push_back(new mutex);
    }
```

Кожен філософ, сідаючи за стіл, бачить дві виделки с номерами

```
fork(forksNo[N]),Fork(forksNo[(N+1)%Count])
```

Філософ запам’ятовує, яка саме виделка має менший номер, а яка більший, щоб завжди брати першою виделку з меншим номером:

```
if (fork > Fork) ::swap(fork,Fork);
```

Тепер філософи приступають до своїх дій — кожен в своєму потоці.

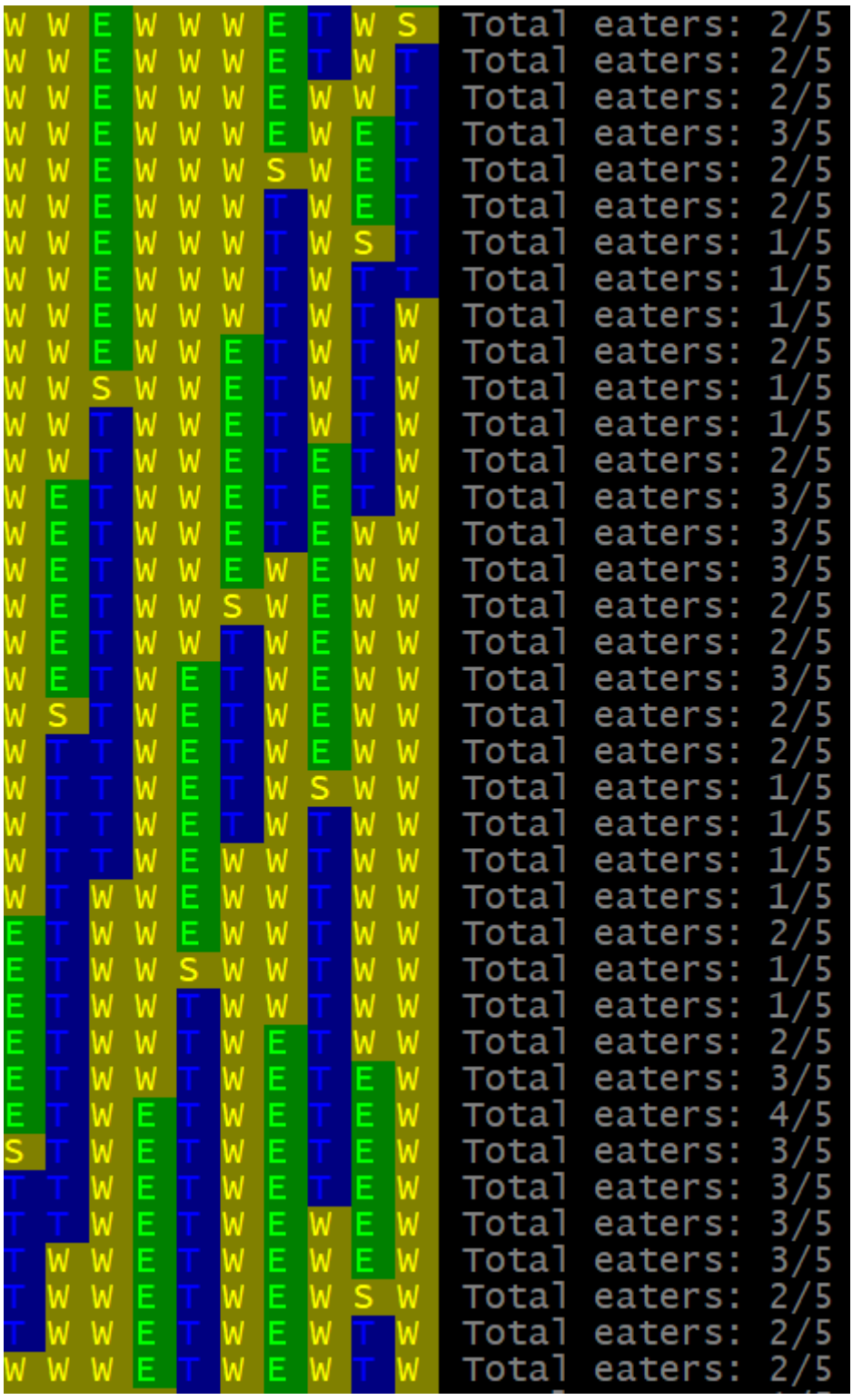
```
for(int i = 0; i < Count; ++i)
{
    thread t(&Philosopher::run,table[i]);
    t.detach();
}
```

Що саме роблять філософи? Кожна зміна стану філософа викликає вивід на екран інформації про стан усіх філософів (функція `report`). Дії філософа виглядають наступним чином:

```
void run() // Philosopher behaviour
{
    for(;;)
    {
        think();     // Думати
        get_forks(); // Взяти виделки
        eat();       // Їсти
        put_forks(); // Покласти виделки
    }
}
void think() { report(THINK); this_thread::sleep_for(distr(reng)*1ms);    }
void eat()   { report(EAT);   this_thread::sleep_for(distr(reng)*1ms);    }
void get_forks() { report(WAIT);  forks[fork]->lock();    // Спершу виделка з
                                  forks[Fork]->lock(); }  // меншим номером
void put_forks() { report(STOP);  forks[Fork]->unlock();
                                  forks[fork]->unlock(); }
```

Функція `report` виводить (кольором) стан кожного філософа, загальну кількість їдців у даний момент та максимально досягнуту кількість їдців.

Маємо такий вигляд екрану навіть після досить довгої роботи:

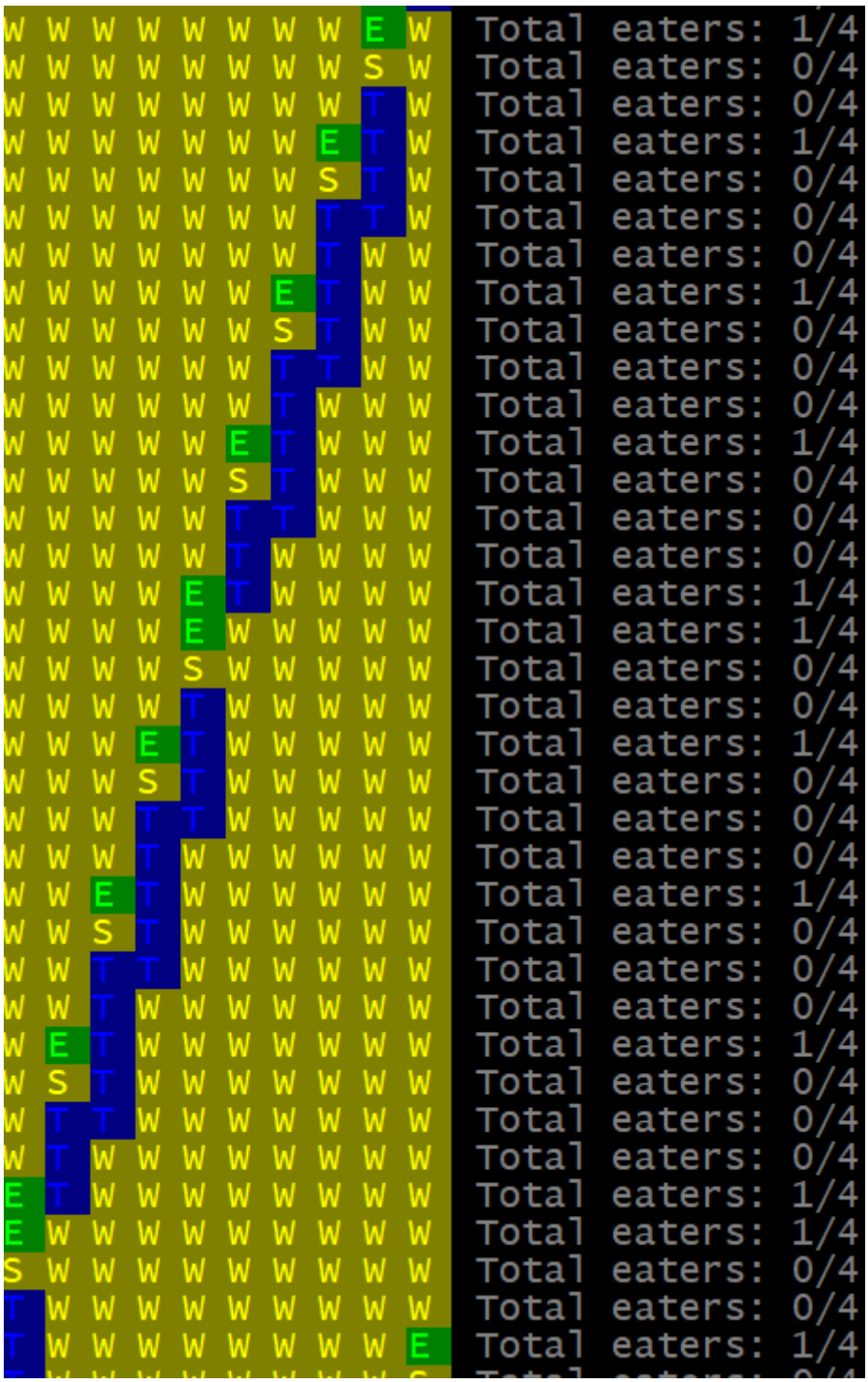


Тобто, як бачимо, для 10 філософів одночасно можуть їсти до 5 філософів (що очевидно), і навіть з довгим часом не спостерігаються жодні неприємності.

Перевіримо, що буде, якщо завжди брати першою виделку ліворуч, а потім праворуч, а не за номерами — тобто просто не будемо виконувати

```
::swap(fork,Fork);
```

Це через деякий час призводить до такої поведінки:



Їсти — як видно на рисунку — може тільки один філософ одночасно (та й то ми просто не дочекалися повного клінчу), а всі інші сидять і чекають, коли звільниться виделка.

Звідси можемо побачити, що на відміну від вказаної ситуації, наш алгоритм розсадки філософів добре працює, і філософи можуть їсти й розмірковувати, і нихто не залишиться голодним.