НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
Кафедра информатики и процессов управления (№17)

Курс «Современные операционные системы»

Лекция 8

Задачи взаимодействия процессов

Москва
2016

Содержание

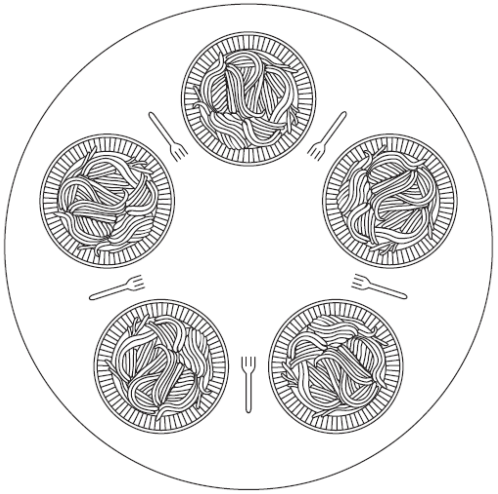
1. **Задача обедающих философов**

1. Задача обедающих философов

Задача. Пять философов сидят за круглым столом, и у каждого из них есть тарелка спагетти. Эти спагетти настолько скользкие, что есть их можно только двумя вилками. Между каждыми двумя тарелками лежит одна вилка.

Жизнь философа состоит из чередующихся периодов приема пищи и размышлений. Когда философ становится голоден, он старается поочередно в любом порядке завладеть правой и левой вилками. Если ему удастся взять две вилки, он на некоторое время приступает к еде, затем кладет обе вилки на стол и продолжает размышления.

Можно ли написать программу для каждого философа, который действует предполагаемым образом и никогда при этом не попадает в состояние зависания?



Ошибочное решение:

```
#define N 5                         /* количество философов */
void philosopher(int i) {           /* i: номер философа (от 0 до 4) */
     while (TRUE) {
          think( );                 /* философ размышляет */
          take_fork(i);             /* берет левую вилку */
          take_fork((i+1)%N);       /* берет правую вилку; */
          eat();                    /* ест спагетти */
          put_fork(i);              /* кладет на стол левую вилку */
          put_fork((i+1)%N);        /* кладет на стол правую вилку */
     }
}
```

Если все пять философов одновременно берут левую от себя вилку. Тогда никто из них не сможет взять правую вилку, что приведет к взаимной блокировке.

Чтобы избежать взаимной блокировки и зависания нужно защитить пять операторов, следующих за вызовом `think`, двоичным семафором. Перед тем как брать вилки, философ должен выполнить в отношении переменной `mutex` операцию `down`. А после того как он положит вилки на место, он должен выполнить в отношении переменной `mutex` операцию `up`.

С теоретической точки зрения это вполне достаточное решение. Но с практической — в нем не учтен вопрос производительности: в каждый момент времени может есть спагетти только один философ, а не два.

Максимум параллелизма для произвольного числа философов достигается, если использовать массив `state`, в котором отслеживается, чем занимается философ: ест, размышляет или пытается поесть (пытается взять вилки). Перейти в состояние приема пищи философ может, только если в этом состоянии не находится ни один из его соседей. Соседи философа с номером `i` определяются макросами `LEFT` и `RIGHT`. Иными словами, если `i` равен `2`, то `LEFT` равен `1`, а `RIGHT` равен `3`.

Верное решение:

```
#define N 5                 /* количество философов */
#define LEFT (i+N-1)%N      /* номер левого соседа для i-го философа */
#define RIGHT (i+1)%N       /* номер правого соседа для i-го философа */
#define THINKING 0          /* философ размышляет */
#define HUNGRY 1            /* философ пытается взять вилки */
#define EATING 2            /* философ ест спагетти */

typedef int semaphore;  /* Семафоры — особый вид целочисленных переменных */
int state[N];           /* массив для отслеживания состояния каждого философа */
semaphore mutex = 1;    /* Взаимное исключение входа в критическую область */
semaphore s[N];         /* по одному семафору на каждого философа */
```

```
void philosopher(int i) {  /* i – номер философа (от 0 до N-1) */
     while (TRUE) {        /* бесконечный цикл */
          think();         /* философ размышляет */
          take_forks(i);   /* берет две вилки или блокируется */
          eat();           /* ест спагетти */
          put_forks(i);    /* кладет обе вилки на стол */
     }
}
void take_forks(int i) {   /* i – номер философа (от 0 до N-1) */
     down(&mutex);         /* вход в критическую область */
     state[i] = HUNGRY;    /* запись факта стремления философа поесть */
     test(i);              /* попытка взять две вилки */
     up(&mutex);           /* выход из критической области */
     down(&s[i]);          /* блокирование, если вилки взять не удалось */
}
void put_forks(i) {        /* i – номер философа (от 0 до N-1) */
     down(&mutex);         /* вход в критическую область */
     state[i] = THINKING;  /* философ наелся */
     test(LEFT);           /* проверка готовности к еде соседа слева */
     test(RIGHT);          /* проверка готовности к еде соседа справа */
     up(&mutex);           /* выход из критической области */
}
void test(i) {             /* i – номер философа (от 0 до N-1) */
     if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING){
          state[i] = EATING;
          up(&s[i]);
     }
}
```