Лекция 35
Condition variables

Ожидание наступления события

- Часто требуется, чтобы одна нить ждала наступление некоторого условия
- Например, главная нить может дожидаться завершения расчетов созданных нитей чтобы объединить результаты расчетов нитей
- Вариант решения: мьютекс + активное ожидание — не подходит
- Вариант решения: использовать канал — требует использования операций ввода-вывода

Условные переменные

- Механизм для рассылки уведомлений
- Одна или несколько нитей ждут наступления события (заблокированы)
- При наступлении события нить посылает уведомление ожидающим нитям, пробуждая одну из них или все
- Для блокировки доступа к условной переменной используется мьютекс

Условные переменные pthread

```
int pthread_cond_init(pthread_cond_t *c,
                      const pthread_contattr_t *a);
int pthread_cond_destroy(pthread_cond_t *c);
int pthread_cond_wait(pthread_cond_t *c,
                      pthread_mutex_t *m);
int pthread_cond_broadcast(pthread_cond_t *c);
int pthread_cond_signal(pthread_cond_t *c);
```

- Перед использованием условная переменная должна быть проинициализирована

Отправка нотификаций

- Если в момент выполнения pthread_cond_signal или pthread_cond_broadcast целевая нить не находится в ожидании в pthread_cond_wait, НОТИФИКАЦИЯ ПОТЕРЯЕТСЯ!
- Поэтому нужна переменная-флаг (обычно bool или int), которая устнавливается в 1
- Для блокировки доступа к ней нужен мьютекс

Использование condvar

```
pthread_cond_t c; // для ожидания
pthread_mutex_t m; // для блокировки
volatile int f; // для передачи значения

// отсылка уведомления
pthread_mutex_lock(&m);
f = 1;
pthread_cond_signal(&c);
pthread_mutex_unlock(&m);

// ожидание уведомления
pthread_mutex_lock(&m);
while (f == 0) pthread_cond_wait(&c, &m);
f = 0;
pthread_mutex_unlock(&m);
```

Классические задачи синхронизации

- Барьер (barrier)
- Обедающие философы (dining philosophers)
- Читатели и писатели (readers-writers)
- Производители-потребители (producers-consumers)
- Спящий парикмахер (sleeping barber)

Пример: ожидание всех рабочих нитей (барьер)

- Пусть есть N рабочих нитей и есть главная нить, которая ожидает прохождения контрольной точки

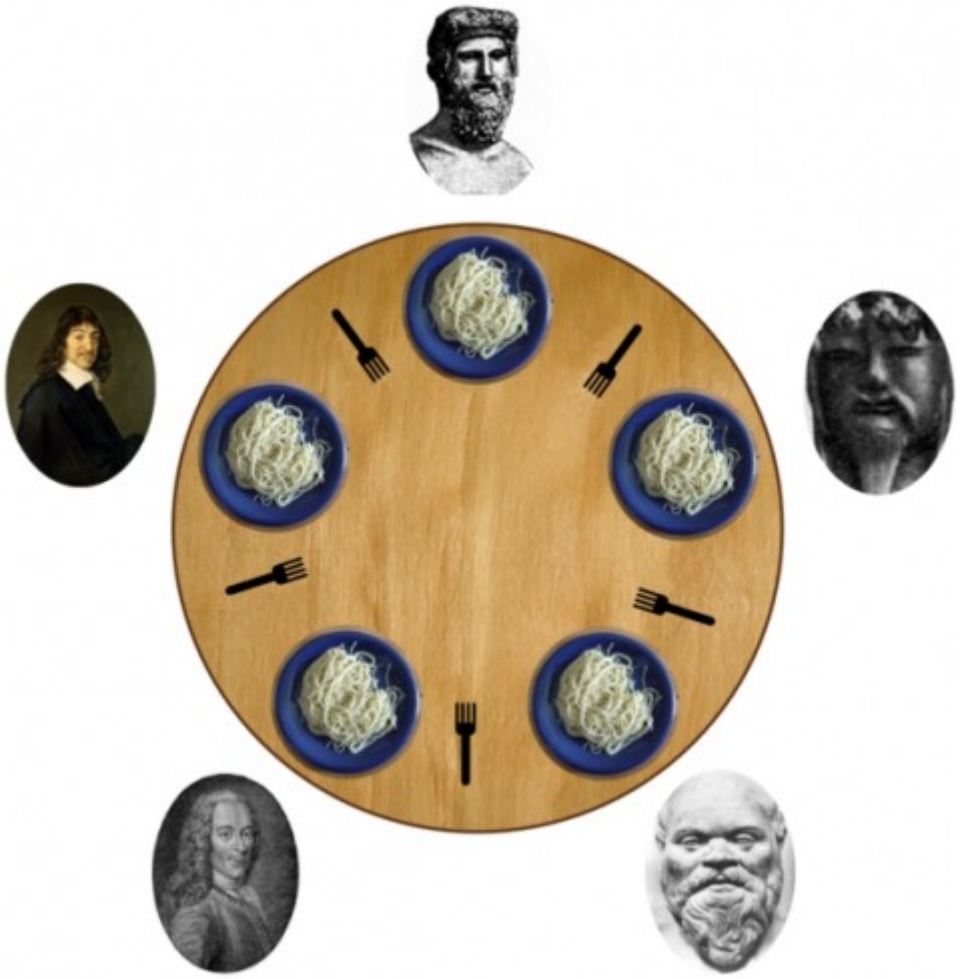
```
// условная переменная
pthread_mutex_t wait_mutex;
pthread_cond_t wait_cond;
int wait_count;
// рабочие нити
pthread_mutex_lock(&wait_mutex);
if (++wait_count == N)
  pthread_cond_signal(&wait_cond);
pthread_mutex_unlock(&wait_mutex);
```

Пример: ожидание всех рабочих нитей

- Пусть есть N рабочих нитей и есть главная нить, которая ожидает прохождения контрольной точки

```
// условная переменная
pthread_mutex_t wait_mutex;
pthread_cond_t wait_cond;
int wait_count;
// главная нить
pthread_mutex_lock(&wait_mutex);
while (wait_count != N)
  pthread_cond_wait(&wait_cond, &wait_mutex);
pthread_mutex_unlock(&wait_mutex);
```

Обедающие философы



Наивное решение

```
void philosopher ( int i ) {
  while (TRUE) {
    think () ;
    take_fork ( i ) ;
    take_fork ( ( i + 1 ) % N ) ;
    eat () ;
    put_fork ( i ) ;
    put_fork ( ( i + 1 )  % N ) ;
  }
  return;
}
```

Deadlock

- Возможна ситуация, когда все философы одновременно захотят есть и возьмут левую от себя вилку — никто не сможет начать есть
-
- «Обедающие философы» показывает важность корректного порядка занятия ресурсов при входе в критическую секцию

Избежание блокировки

- При N = 2 задача сводится к:

- Процесс 1:

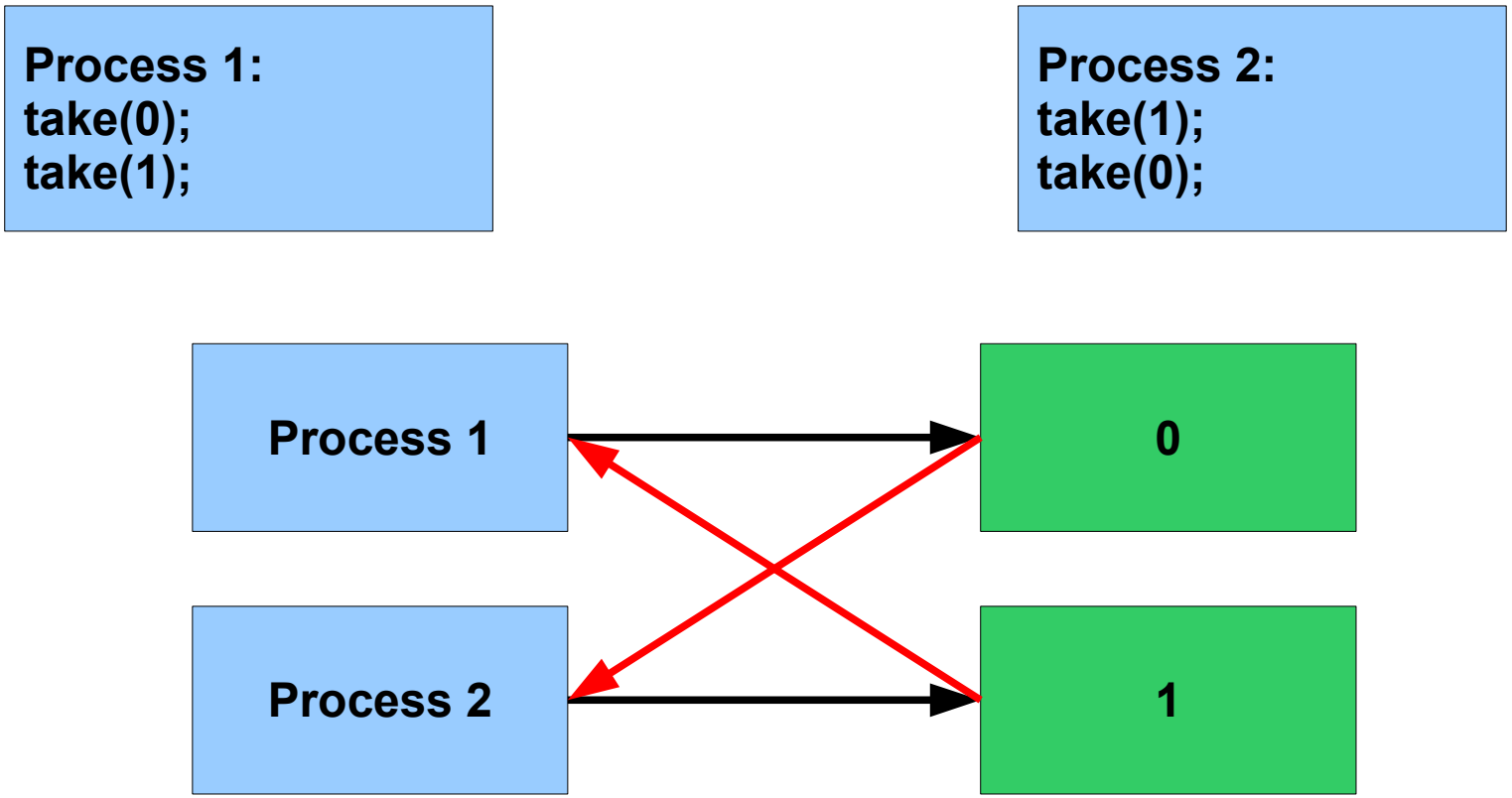
```
take_fork(0);
take_fork(1);
```

- Процесс 2:

```
take_fork(1);
take_fork(0);
```

- Изменение порядка взятия вилок в процессе 2 решает проблему!

Обнаружение тупиков



- Захваченный ресурс — дуга от процесса к ресурсу
- Ожидаемый ресурс — дуга от ресурса к процессу
- Если в графе есть цикл, система попала в состояние тупика

Избежание блокировки

- Каждый процесс может сначала взять вилку с меньшим номером, потом взять вилку с большим номером
- Недостатки:
  - Операция по взятию вилок неатомарна
  - Могут появляться цепочки философов, которые взяли вилку с меньшим номером, но ждут вилку с большим номером — такая цепочка разрушится только когда философ с максимальным номером закончит есть
  - Требуется отношение порядка на множестве вилок

Избежание блокировки

- Проверяем состояние соседей философа под мьютексом
- Если философ не может начать есть, он засыпает на условной переменной
- Когда состояние изменится, его разбудят
- Недостатки:
  - Мьютекс на весь стол, только один философ может проверить состояние
  - Немасштабируемо

Читатели и писатели

- Дана некоторая разделяемая область память
- К этой структуре данных может обращаться произвольное количество «читателей» и произвольное количество «писателей»
- Несколько читателей могут получить доступ одновременно, писатели в этот момент не допускаются
- Только один писатель может получить доступ, другие писатели и читатели должны ждать

Решение 1

- Первое решение: читатель может войти в критическую секцию, если нет писателей
- Это решение несправедливо, так как отдает предпочтение читателям
- Плотный поток запросов от читателей может привести к тому, что писатель никогда не получит доступа к критической секции: ситуация «голодания» (starvation)

Решение 2

- Отдадим предпочтение писателям, то есть читатель не входит в критическую секцию, если есть хотя бы один ожидающий писатель
- Данное решение отдает приоритет писателям, и тоже несправедливо
- Возможно «голодание» (starvation) читателей

Решение 3

- Третье решение: не отдавать никому приоритета, просто использовать мьютекс
- Не используется возможность одновременного чтения

Решение 4

- Формируем очередь запросов
- Несколько идущих подряд в очереди запросов на чтение могут выполняться параллельно
- Запросы на запись выполняются в эксклюзивном режиме

Производители-потребители (producer-consumer problem)

- Дан буфер фиксированного размера (N), в котором размещается очередь.
- Производители добавляют элементы в конец очереди, если буфер заполнился, производители засыпают
- Потребители забирают элементы из начала очереди, если буфер пуст, потребители засыпают

Спящий парикмахер (sleeping barber)

- В парикмахерской имеется одно кресло для стрижки и N кресел для ожидающих посетителей
- Если нет посетителей, парикмахер спит
- Если приходит посетитель и кресло для стрижки свободно, посетитель садится в него и парикмахер начинает его стричь
- В противном случае посетитель садится в кресло для ожидающих
- Если все кресла заняты, посетитель уходит