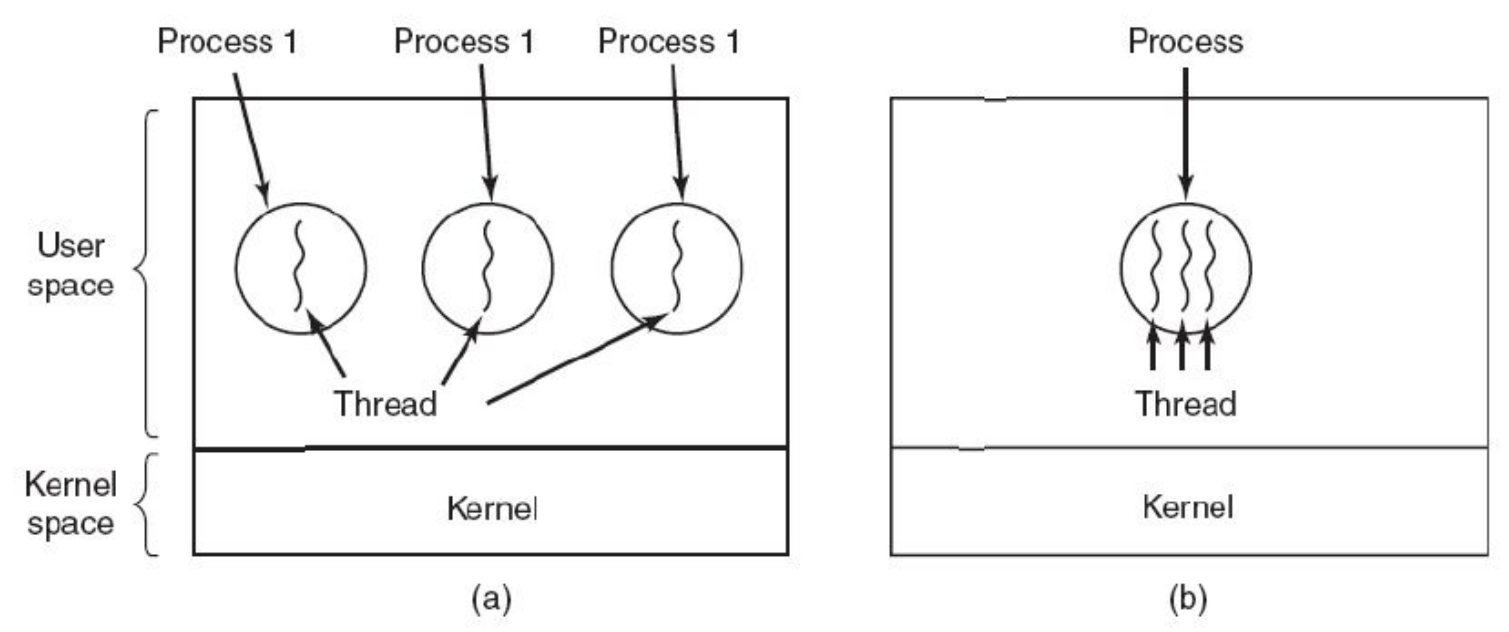
Параллельное программирование

Проблемы многопоточного программирования

Процессы


Process 1
Process 1
Process 1
User space
Thread
Kernel space
Kernel
(a)
Process
Thread
Kernel
(b)

Реализация потоков

- Потоки на уровни ядра ОС:
  - Win32,Posix (Unix), LWKT(BSD)
- Библиотеки:
  - OpenMP, Boost.Threads, Intel Threading Building Block
- На уровне языка:
  - Java, C#, C++11, Erlang

Взаимодействие потоков

- Mutex (Mutualle Exclusive Access)
- Semaphore
- Critical Section
- Condition Variable
- ……

Race Condition

- Ошибка проектирования многопоточной системы или приложения, при которой работа системы или приложения зависит от того, в каком порядке выполняются части кода.
- Варианты решения
  - Ввод локальной переменной
  - Синхронизация

Race Condition

```
void Hello(const char* name){
    for (int i = 0; i < 10; i++){
        printf("%s: Hello world!\n", name);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    std::thread tr1(Hello,"Ivan");
    std::thread tr2(Hello, "Alex");


    tr1.join();
    tr2.join();

    return 0;
}
```

Race Condition

```
int x;
void func1(){
    while (true)
    {
        x++;
    }
}

void func2(){
    while (true)
    {
        if (x % 1000 == 0){
            printf("x = %d\n", x);
        }
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    std::thread tr1(func1);
    std::thread tr2(func2);
    tr1.join();
    tr2.join();

    return 0;
}
```

Race Condition

```
int x;
void func1(){
    while (true)
    {
        std::lock_guard<std::mutex> guatd(mutex1);
        x++;
    }
}

void func2(){
    while (true)
    {
        std::lock_guard<std::mutex> guatd(mutex1);
        if (x % 1000 == 0){
            printf("x = %d\n", x);
        }
    }
}
```

Race Condition

```
total = (liner(a) + liner(b))*step / 2;
#pragma omp parallel
#pragma omp for private(x)
for (int i = 0; i < eteration_count; ++i){
    x = a + i*step;
    total += liner(x)*step;
}
```

Race Condition

```
total = (liner(a) + liner(b))*step / 2;
#pragma omp parallel
#pragma omp for private(x) reduction(+:total)
for (int i = 0; i < eteration_count; ++i){
    x = a + i*step;
    total += liner(x)*step;
}
```

Deadlock

Ситуация в многозадачной среде, при которой несколько процессов находятся в состоянии бесконечного ожидания ресурсов, занятых самими этими процессами.

Deadlock

Deadlock

```
int x;
int y;

void func1(){
    while (true)
    {
        std::lock_guard<std::mutex> guatdy(mutex2);
        y++;
        std::lock_guard<std::mutex> guatd(mutex1);
        x++;
    }
}

void func2(){
    while (true)
    {
        std::lock_guard<std::mutex> guatd(mutex1);
        if (x % 1000 == 0){
            printf("x = %d\n", x);
        }
        std::lock_guard<std::mutex> guatdy(mutex2);
        if (y % 500 == 0){
            printf("y = %d\n", y);
        }
    }
}
```

Livelock

Это слово означает такую ситуацию: система не «застревает» (как в обычной взаимной блокировке), а занимается бесполезной работой, её состояние постоянно меняется — но, тем не менее, она «зациклилась», не производит никакой полезной работы.

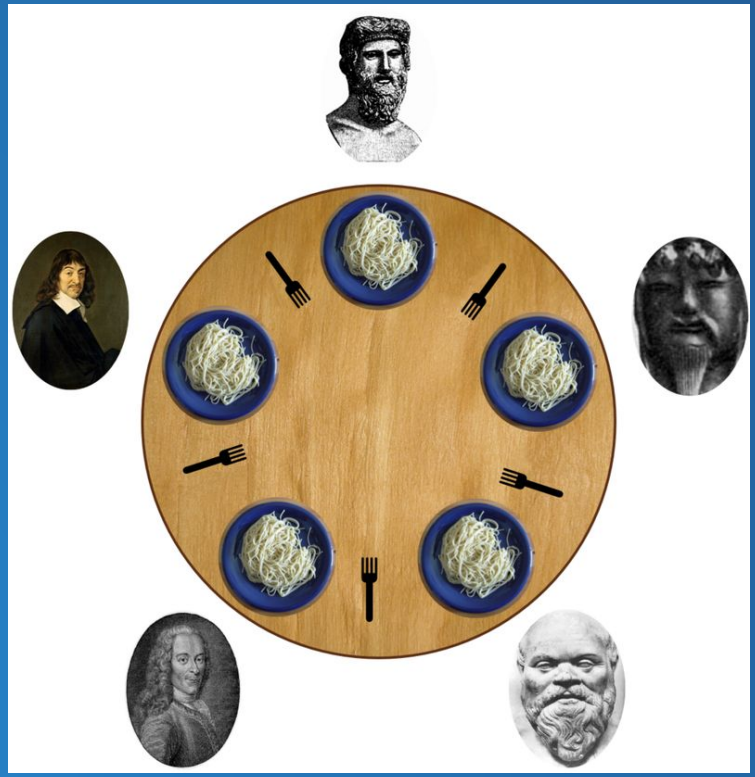
Livelock

```
void func1(){
    while (true)
    {
        if (std::try_lock(mutex2) == -1)
            y++;
        else
            printf("Cannot access to y\n");

        if (std::try_lock(mutex1) == -1){
            x++;
        }
        else{
            printf("Cannot access to x\n");
        }
    }
}

void func2(){
    while (true)
    {
        if (std::try_lock(mutex1) == -1){
            if (x % 1000 == 0){
                printf("x = %d\n", x);
            }
        }
        if (std::try_lock(mutex2) == -1){
            if (y % 500 == 0){
                printf("y = %d\n", y);
            }
        }
    }
}
```

Проблема обедающих философов



Проблема обедающих философов

Вариант1:

- Размышлять, пока не освободится левая вилка. Когда вилка освободится — взять её.
- Размышлять, пока не освободится правая вилка. Когда вилка освободится — взять её.
- Есть
- Положить левую вилку
- Положить правую вилку
- Повторить алгоритм сначала

Проблема обедающих философов

Варианты решения:

- Размышлять, пока не освободится левая вилка. Когда вилка освободится — взять её.
- Размышлять, пока не освободится правая вилка. Когда вилка освободится — взять её.
- Есть
- Положить левую вилку
- Положить правую вилку
- Повторить алгоритм сначала

Потенциальный Deadlock!!

Проблема обедающих философов

Вариант2:

- Размышлять, пока не освободится левая вилка. Когда вилка освободится — взять её.
- Размышлять, пока не освободится правая вилка или пройдет 5 минут. Когда вилка освободится — взять её. Если пойдет 5 минут, то кладем левую вилку, и повторяем алгоритм.
- Есть
- Положить левую вилку
- Положить правую вилку
- Повторить алгоритм сначала

Проблема обедающих философов

Вариант2:

- Размышлять, пока не освободится левая вилка. Когда вилка освободится — взять её.
- Размышлять, пока не освободится правая вилка или пройдет 5 минут. Когда вилка освободится — взять её. Если пойдет 5 минут, то кладем левую вилку, и повторяем алгоритм.
- Есть
- Положить левую вилку
- Положить правую вилку
- Повторить алгоритм сначала

Потенциальный Livelock!!

Проблема обедающих философов

Решения:

- Официант
- Иерархия ресурсов