Взаимодействие процессов

Причины взаимодействия процессов

- Повышение скорости работы.
- Совместное использование данных.
- Модульная конструкция системы.
- Удобство работы пользователя

Категории средств обмена информацией

- Сигнальные.
  Используются, как правило, для извещения процесса о наступлении какого-либо события.
- Канальные.
  "Общение" процессов происходит через линии связи, предоставленные операционной системой
- Разделяемая память.
  Два или более процессов могут совместно использовать некоторую область адресного пространства .

Логическая организация механизма передачи информации

- Порядок установления связи.
- Адресация – прямая и непрямая
- Количество участников сеанса связи
- Направленность связи
  - симплексная
  - полудуплексная
  - дуплексная
- Способ завершения сеанса связи

Особенности передачи информации с помощью линий связи

Буферизация
- -Буфер нулевой емкости или отсутствует.
- -Буфер ограниченной емкости.
- -Буфер неограниченной емкости.

Две модели передачи данных
- - Поток
- -Сообщения

Надежность средств связи

Условия надежности средств связи

- Не происходит потери информации.
- Не происходит повреждения информации.
- Не появляется лишней информации.
- Не нарушается порядок данных в процессе обмена.

Активности и атомарные операции

- Активности- последовательное выполнение ряда действий, направленных на достижение определенной цели.
- Атомарные, операции- неделимые
  P: a b c Q: d e f
  PQ: a b c d e f
  a b c d e f
  a b d c e f
  …
  a d b c e f
  .....
  d e f a b c

Псевдопараллельное выполнение

P: x = 2
y = x-1

Q: x = 3
y = x+1

Результаты (x, y): (3, 4), (2, 1), (2, 3) и (3, 2)

набор активностей детерминирован, если всякий раз при псевдопараллельном исполнении для одного и того же набора входных данных он дает одинаковые выходные данные.

Достаточные условия Бернстайна

- Набор входных переменных программы - R(P)
- Набор выходных переменных программы - W(P)

Если для двух данных активностей P и Q:

- пересечение W(P) и W(Q) пусто,
- пересечение W(P) с R(Q) пусто,
- пересечение R(P) и W(Q) пусто,

тогда выполнение P и Q детерминировано. Если эти условия не соблюдены, возможно, параллельное выполнение P и Q детерминировано, а может быть, и нет.

Пример условий Бернстайна

P: x=u+v; y=x*w;

получаем

R(P) = {u, v, x, w},

W(P) = {x, y}

Механизмы синхронизации выполнения программ упорядочивают доступ программ к данным

Недетерминированный набор программ имеет race condition (состояние гонки , состояние состязания)

Задачу упорядоченного доступа к разделяемым данным (устранение race condition) в том случае, когда нам не важна его очередность, можно решить, если обеспечить каждому процессу эксклюзивное право доступа к этим данным.

Критическая секция

Критическая секция – это часть программы, исполнение которой может привести к возникновению race condition для определенного набора программ.

Реализация взаимоисключения для критических секций программ с практической точки зрения означает, что по отношению к другим процессам, участвующим во взаимодействии, критическая секция начинает выполняться как атомарная операция.

```
while (some condition) {
        entry section
        critical section
        exit section
        remainder section}
```

Требования, предъявляемые к алгоритмам

- Задача должна быть решена чисто программным способом на обычной машине, не имеющей специальных команд взаимоисключения.
- В программе не должно быть предположений о скорости или количестве процессоров.
- Два процесса не должны одновременно находиться в критич. областях. (Условие взаимоисключения)
- Процесс, находящийся вне критической области, не может блокировать другие процессы. (Условие прогресса)
- Невозможна ситуация, в которой процесс вечно ждет попадания в критическую область.(Условие ограниченного ожидания)

Запрет прерываний

- while (some condition) {

  запретить все прерывания

  critical section

  разрешить все прерывания

  remainder section }

Переменная-замок

```
shared int lock = 0;
/* shared означает, что переменная является
   разделяемой */
while (some condition) {
while(lock);
   lock = 1;
critical section
lock = 0;
remainder section
   }
```

Строгое чередование

```
shared int turn = 0;
while (some condition) {
while(turn != i);
critical section
turn = 1-i;
remainder section }
```

Флаги готовности

```
shared int ready[2] = {0, 0};
while (some condition) {
ready[i] = 1;
while(ready[1-i]);
critical section
ready[i] = 0;
remainder section }
```

Алгоритм Петерсона

```
shared int ready[2] = {0, 0};
shared int turn;
while (some condition) {
ready[i] = 1;
turn =i;
while(ready[1-i] && turn == i);
critical section
ready[i] = 0;
remainder section }
```

Два процесса не должны одновременно находиться в критич. областях. (Условие взаимоисключения)

Процесс, находящийся вне критической области, не может блокировать другие процессы. (Условие прогресса)

Невозможна ситуация, в которой процесс вечно ждет попадания в критическую область.(Условие ограниченного ожидания)

Алгоритм булочной

Обозначения: (a,b) < (c,d), если a < c или если a == c и b < d

```
shared enum {false, true}
    choosing[n];
    shared int number[n];
    while (some condition) {
choosing[i] = true;
number[i] = max(number[0], ...,number[n-1]) + 1;
choosing[i] = false;
for(j = 0; j < n; j++){
    while(choosing[j]);
    while(number[j] != 0 && (number[j],j) < (number[i],i));    }
          critical section
    number[i] = 0;
    remainder section }
```

Аппаратная поддержка взаимоисключений

```
int Test_and_Set (int *target){
    int tmp = *target;
    *target = 1;
    return tmp; }


shared int lock = 0;
    while (some condition) {
    while(Test_and_Set(&lock));
    critical section
    lock = 0;
    remainder section }
```

Семафоры

Семафор представляет собой целую переменную, принимающую неотрицательные значения, доступ любого процесса к которой, за исключением момента ее инициализации, может осуществляться только через две атомарные операции:

P (от датского слова proberen – проверять)
V (от verhogen – увеличивать).

P(S): пока S == 0 процесс блокируется;
иначе S = S – 1;

V(S): S = S + 1;

Решение проблемы producer-consumer с помощью семафоров

```
Semaphore mutex = 1;
Semaphore empty = N; /* где N – емкость буфера*/
Semaphore full = 0;
```

```
Producer:
while(1) {
        produce_item;
        P(empty);
        P(mutex);
        put_item;
        V(mutex);
        V(full);    }
```

```
Consumer:
while(1) {
        P(full);
        P(mutex);
        get_item;
        V(mutex);
        V(empty);
        consume_item;    }
```

Мьютекс (mutex, сокращение от mutual exclusion — взаимное исключение)

Мьютекс — переменная, которая может находиться в одном из двух состояний: блокированном или неблокированном.

Две процедуры

mutex Lock

mutex_unlock

Мониторы

```
monitor monitor_name {
//описание внутренних переменных ;
void m1(...){...
}
void m2(...){...
}
...
void mn(...){...
}
{
блок инициализации       внутренних переменных;
}
}
```

Решение проблемы producer-consumer с помощью мониторов 1

```
monitor ProducerConsumer {
condition full, empty;
int count;
void put() {
if(count == N) full.wait;
put_item;
count += 1;
if(count == 1) empty.signal;
 }
```

P(S): пока S == 0 процесс блокируется; иначе S = S – 1;
V(S): S = S + 1;

Решение проблемы producer-consumer с помощью мониторов 2

```
void get()
{
if (count == 0) empty.wait;
get_item();
count -= 1;
if(count == N-1) full.signal;
 }
{
count = 0;
 }
}
```

Решение проблемы producer-consumer с помощью мониторов 3

Producer:

```
while(1)
  {
produce_item;
ProducerConsumer.put();
}
```

Consumer:

```
        while(1)
        {
         ProducerConsumer.get();
        consume_item;
         }
```

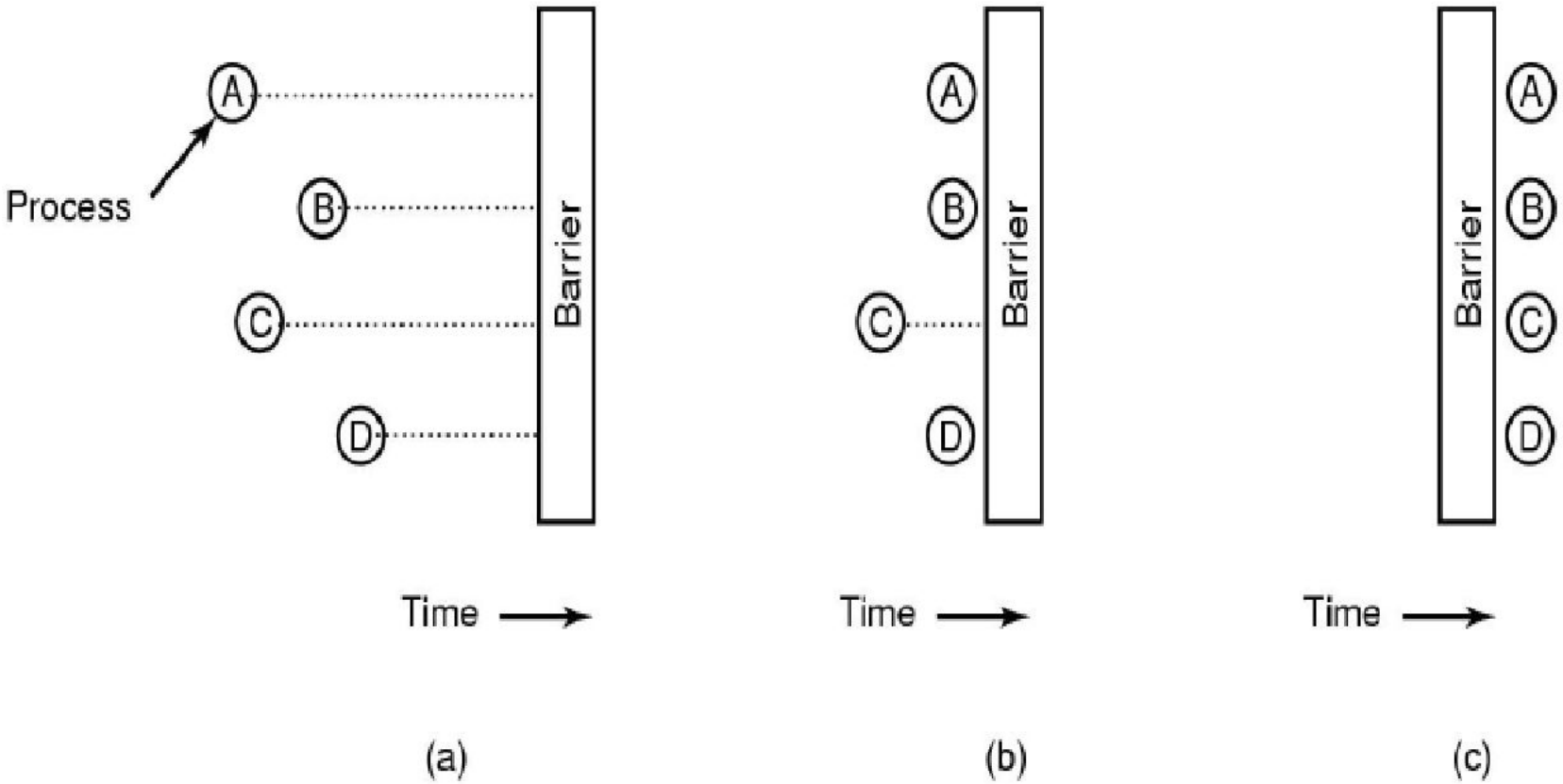
Решение проблемы производителя и потребителя с передачей сообщений 1

```
#define N 100    /* количество сегментов в буфере */
void producer(void)
{
int item;
message m:       /* буфер для сообщений */
while (TRUE) {
produce_item();    /* сформировать нечто, чтобы заполнить буфер*/
 /* ожидание прибытия пустого сообщения */
receive(consumer. &m);
 /* сформировать сообщение для отправки */
build_message(&m, item);
send(consumer. &m):      /* отослать элемент потребителю */
 }
}
```

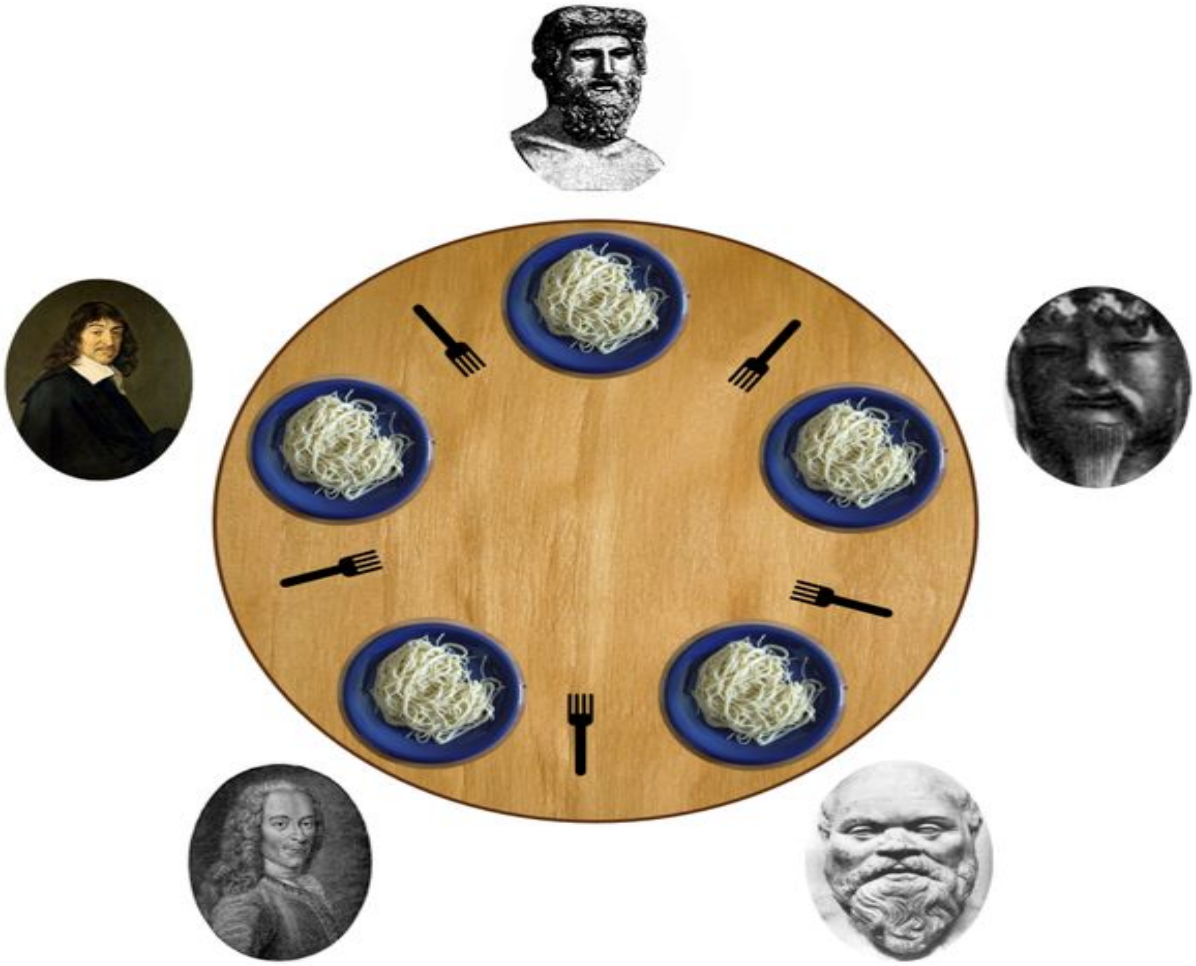
Решение проблемы производителя и потребителя с передачей сообщений 2

```
void consumer(void)
{
int item, i;
    message m;
    for (i - 0; i < N; i++)
send(producer, &m) ; /* отослать N пустых сообщений */
while (TRUE) {
receive(producer. &m);    /* получить сообщение с
    элементом */
item = extract_item(&m);   /* извлечь элемент из
    сообщения */
send(producer, &m):      /* отослать пустое сообщение */
    consume_item(item):      /* обработка элемента */
    }
}
```

Барьеры


Process
A
B
C
D
Barrier
Time
(a)
A
B
C
D
Barrier
Time
(b)
A
B
C
D
Barrier
Time
(c)

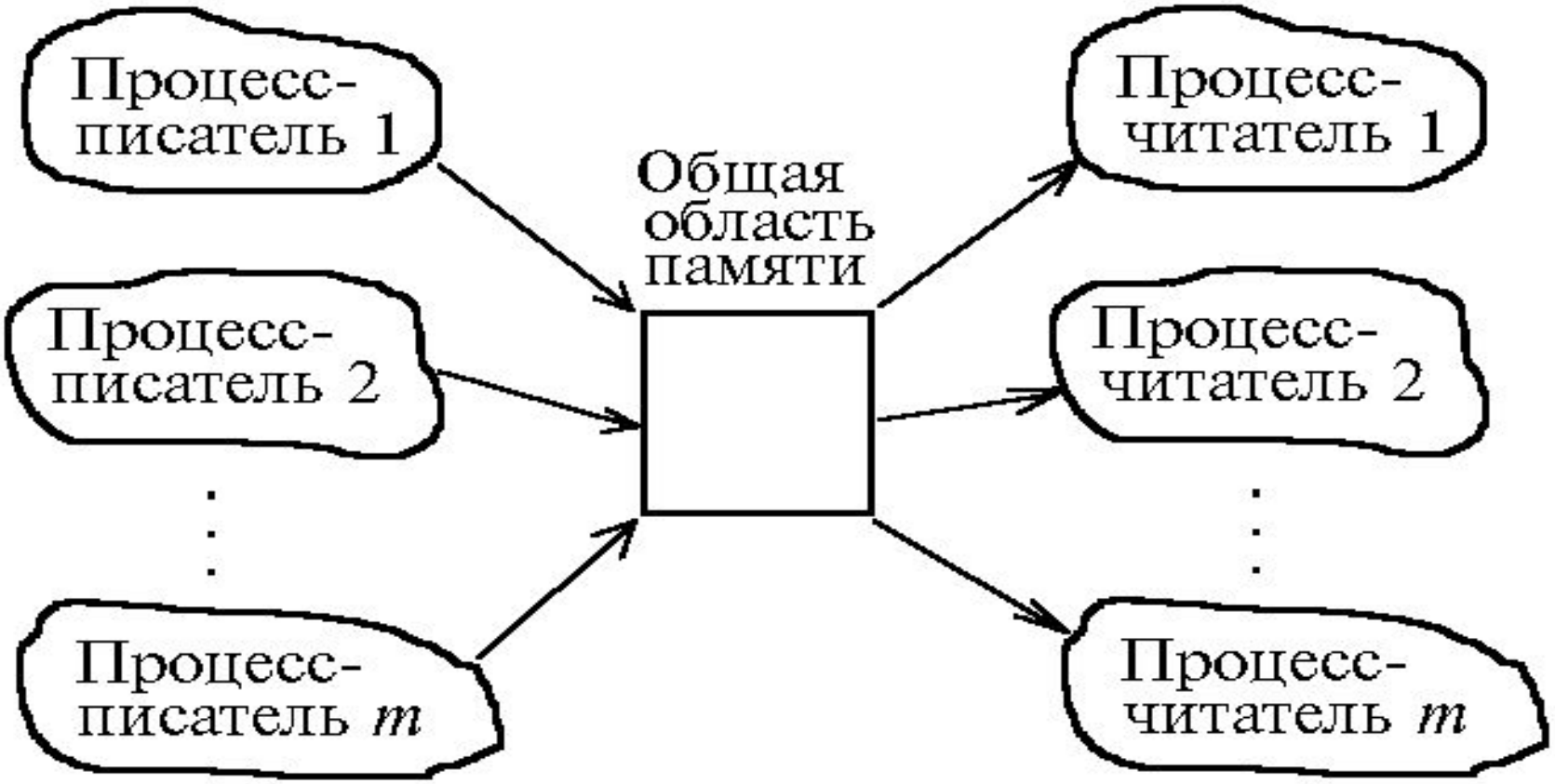
Проблема обедающих философов



Неверное решение проблемы обедающих философов

```
#define N 5   /* Количество философов */
void philosospher (int i)/* i - номер философа, от 0 до 4 */
{
while(TRUE) {
think();        /* Философ размышляет */
takefork(i);     /* Берет левую вилку */
takefork((i+l)% N);/* Берет правую вилку */
eat();  /* Спагетти, ням-ням */
putfork(i):      /* Кладет на стол левую вилку */
putfork((i+l) %  N): /* Кладет на стол правую вилку */ }
    }
```

Проблема читателей и писателей



Проблема спящего брадобрея

С праздником 14 марта

- "Математику только зачем учить надо, что она ум в порядок приводит" (Ломоносов)
- "Математика - гимнастика ума" (Суворов)

- "Наука только тогда достигает совершенства, когда она начинает пользоваться математикой" (Маркс)

"Высшая математика убивает креативность" (Фурсенко, министр образования и науки РФ)