Архитектура ЭВМ. Операционные системы

Власов Евгений

Для достижения поставленной цели различные процессы (возможно, даже принадлежащие разным пользователям) могут исполняться псевдопараллельно на одной вычислительной системе или параллельно на разных вычислительных системах, взаимодействуя между собой.

Для чего процессам нужно заниматься совместной деятельностью? Какие существуют причины для их кооперации?

- **Повышение скорости работы**. Пока один процесс ожидает наступления некоторого события (например, окончания операции ввода-вывода), другие могут заниматься полезной работой, направленной на решение общей задачи. В многопроцессорных вычислительных системах программа разбивается на отдельные кусочки, каждый из которых будет исполняться на своем процессоре.
- **Совместное использование данных**. Различные процессы могут, к примеру, работать с одной и той же динамической базой данных или с разделяемым файлом, совместно изменяя их содержимое.
- **Модульная конструкция какой-либо системы**. Типичным примером может служить микроядерный способ построения операционной системы, когда различные ее части представляют собой отдельные процессы, взаимодействующие путем передачи сообщений через микроядро.
- Наконец, это может быть необходимо просто для удобства работы пользователя, желающего, например, редактировать и отлаживать программу одновременно. В этой ситуации процессы редактора и отладчика должны уметь взаимодействовать друг с другом.

Межпроцессное взаимодействие

(англ. inter-process communication, IPC) — обмен данными между потоками одного или разных процессов. Реализуется посредством механизмов, предоставляемых ядром ОС или процессом, использующим механизмы ОС и реализующим новые возможности IPC. Может осуществляться как на одном компьютере, так и между несколькими компьютерами сети.

Условно все IPC можно разделить на обмен данными и синхронизацию обмена.

Более детально IPC делятся на следующие:

- механизмы обмена сообщениями;
- механизмы синхронизации;
- механизмы разделения памяти;
- механизмы удалённых вызовов (RPC).

Межпроцессное взаимодействие, наряду с механизмами адресации памяти, является **основой** для разграничения адресного пространства между процессами.

Сигнальные IPC

Передается минимальное количество информации – один бит, "да" или "нет". Используются, как правило, для извещения процесса о наступлении какого-либо события. Степень воздействия на поведение процесса, получившего информацию, минимальна. Все зависит от того, знает ли он, что означает полученный сигнал, надо ли на него реагировать и каким образом. Неправильная реакция на сигнал или его игнорирование могут привести к трагическим последствиям.

Канальные IPC

Обмен данными между процессами происходит через линии связи, предоставленные операционной системой, и напоминает общение людей по телефону, с помощью записок, писем или объявлений. Объем передаваемой информации в единицу времени ограничен пропускной способностью линий связи. С увеличением количества информации возрастает и возможность влияния на поведение другого процесса.

Разделяемая память

Два или более процессов могут совместно использовать некоторую область адресного пространства. Созданием разделяемой памяти занимается операционная система (если, конечно, ее об этом попросят). "Общение" процессов напоминает совместное проживание студентов в одной комнате общежития. Возможность обмена информацией максимальна, как, впрочем, и влияние на поведение другого процесса, но требует повышенной осторожности (если вы переложили на другое место вещи вашего соседа по комнате, а часть из них еще и выбросили). Использование разделяемой памяти для передачи/получения информации осуществляется с помощью средств обычных языков программирования, в то время как сигнальным и канальным средствам коммуникации для этого необходимы специальные системные вызовы. Разделяемая память представляет собой наиболее быстрый способ взаимодействия процессов в одной вычислительной системе.

Для оценки производительности различных механизмов IPC используют следующие параметры:

- *пропускная способность* (количество сообщений в единицу времени, которое ядро ОС или процесс способна обработать);
- *задержки* (время между отправкой сообщения одним потоком и его получением другим потоком).

Условия Бернстайна

- Необходимость синхронизации обмена данными можно установить с помощью достаточных условий. Если {W} – множество операций записи процесса, а {R} – множество операций чтения процесса, то выполнение

$$\{W(P)\} \bigcap \{R(Q)\} \neq \emptyset$$

$$\{W(P)\} \bigcap \{W(Q)\} \neq \emptyset$$

$$\{R(P)\} \bigcap \{W(Q)\} \neq \emptyset$$

служит достаточным для понимания синхронизации двух процессов или потоков.

Race condition

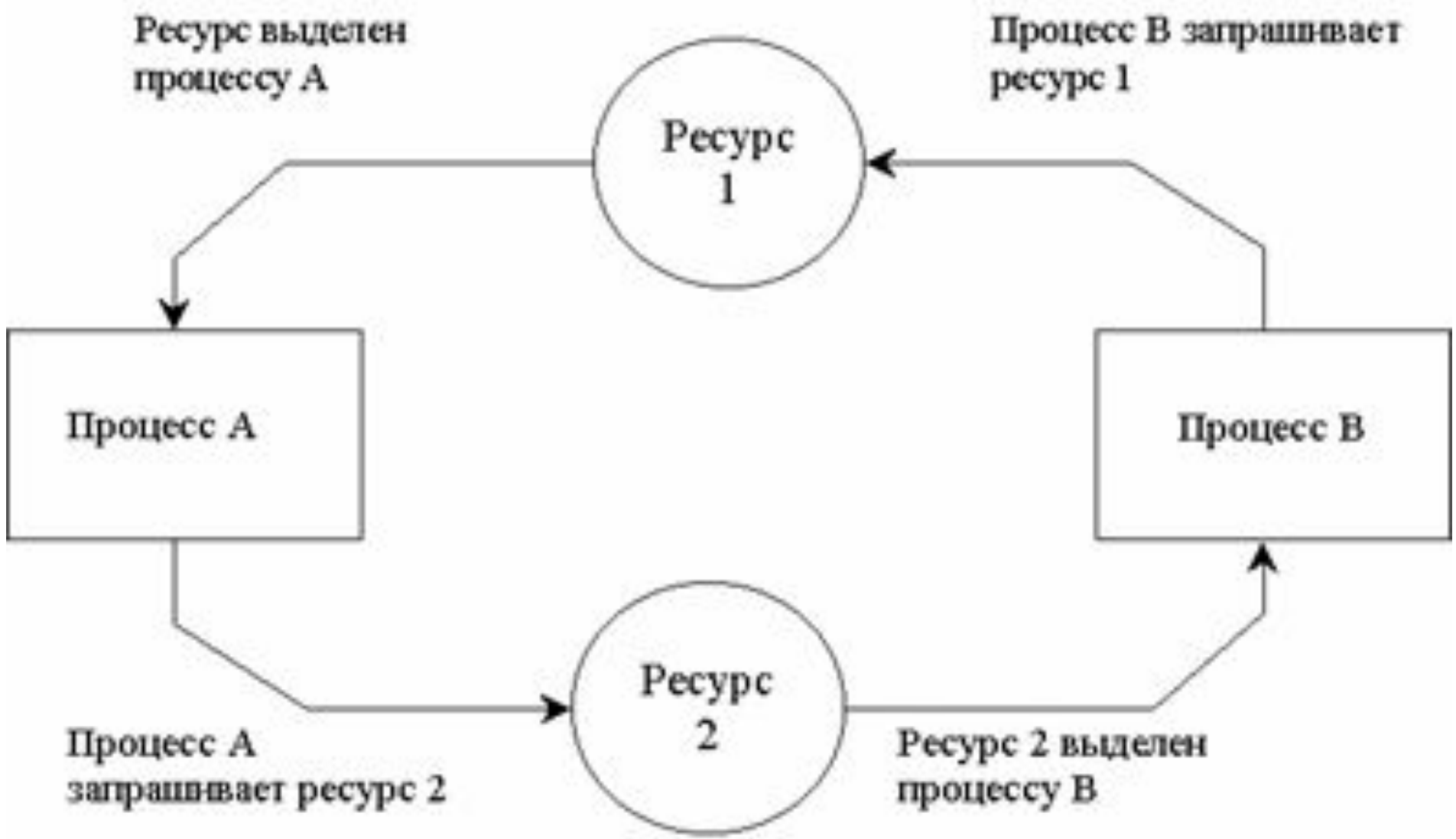
ошибка проектирования многопоточной системы или приложения, при которой работа системы или приложения зависит от того, в каком порядке выполняются части кода.

Критическая секция

это часть программы, исполнение которой может привести к возникновению race condition для определенного набора программ. Чтобы исключить эффект гонок по отношению к некоторому ресурсу, необходимо организовать работу так, чтобы в каждый момент времени только один процесс мог находиться в своей критической секции, связанной с этим ресурсом. Иными словами, необходимо обеспечить реализацию взаимоисключения для критических секций 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.

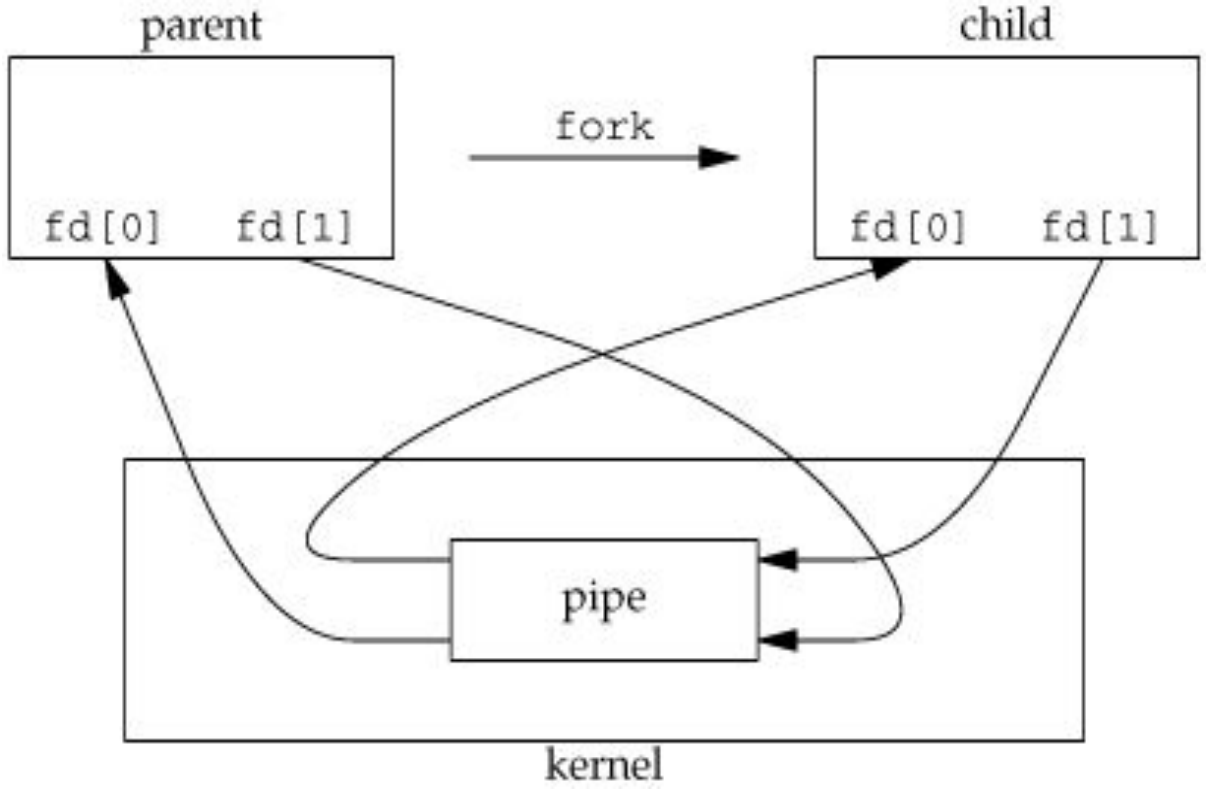
Взаимная блокировка

- Условие взаимного исключения. Каждый ресурс либо выделен в данный момент только одному процессу, либо доступен.
- Условие удержания и ожидания. Процессы, удерживающие в данный момент ранее выделенные им ресурсы, могут запрашивать новые ресурсы.
- Условие невыгружаемости. Ранее выделенные ресурсы не могут быть принудительно отобраны у процесса. Они должны быть явным образом высвобождены тем процессом, который их удерживает.
- Условие циклического ожидания. Должна существовать кольцевая последовательность из двух и более процессов, каждый из которых ожидает высвобождения ресурса, удерживаемого следующим членом последовательности.


Ресурс выделен процессу А
Процесс В запрашивает ресурс 1
Ресурс 1
Процесс А
Процесс В
Ресурс 2
Процесс А запрашивает ресурс 2
Ресурс 2 выделен процессу В

Неименованные каналы - pipe

Для обмена между родственными процессами используют `pipe`. Pipe представляет собой два файловых дескриптора: один для чтения, другой для записи.



Каналы не поддерживают произвольный доступ, т. е. данные могут считываться только в том же порядке, в котором они записывались.

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
 int main (int argc, char * argv[]) {
   int pipedes[2];
   pid_t pid;
   pipe(pipedes);
   pid = fork();
   if ( pid > 0 ) {
      char *str = "String passed via pipe\n";
      close(pipedes[0]);
      write(pipedes[1], (void *) str, strlen(str) + 1);
      close(pipedes[1]);
   } else {
      char buf[1024];
      int len;
      close(pipedes[1]);
      while ((len = read(pipedes[0], buf, 1024)) != 0)
      write(2, buf, len);
      close(pipedes[0]);
   }
   return 0;
}
```