Взаимодействие процессов

Что это

Довольно часто процессам необходимо взаимодействовать с другими процессами

Вопросы

1) Как один процесс может передавать информацию другому процессу?
2)Как обеспечить совместную работы процессов без создания взаимных помех?

3)Определение правильной последовательности на основе существующих взаимозависимостей

Передача информации

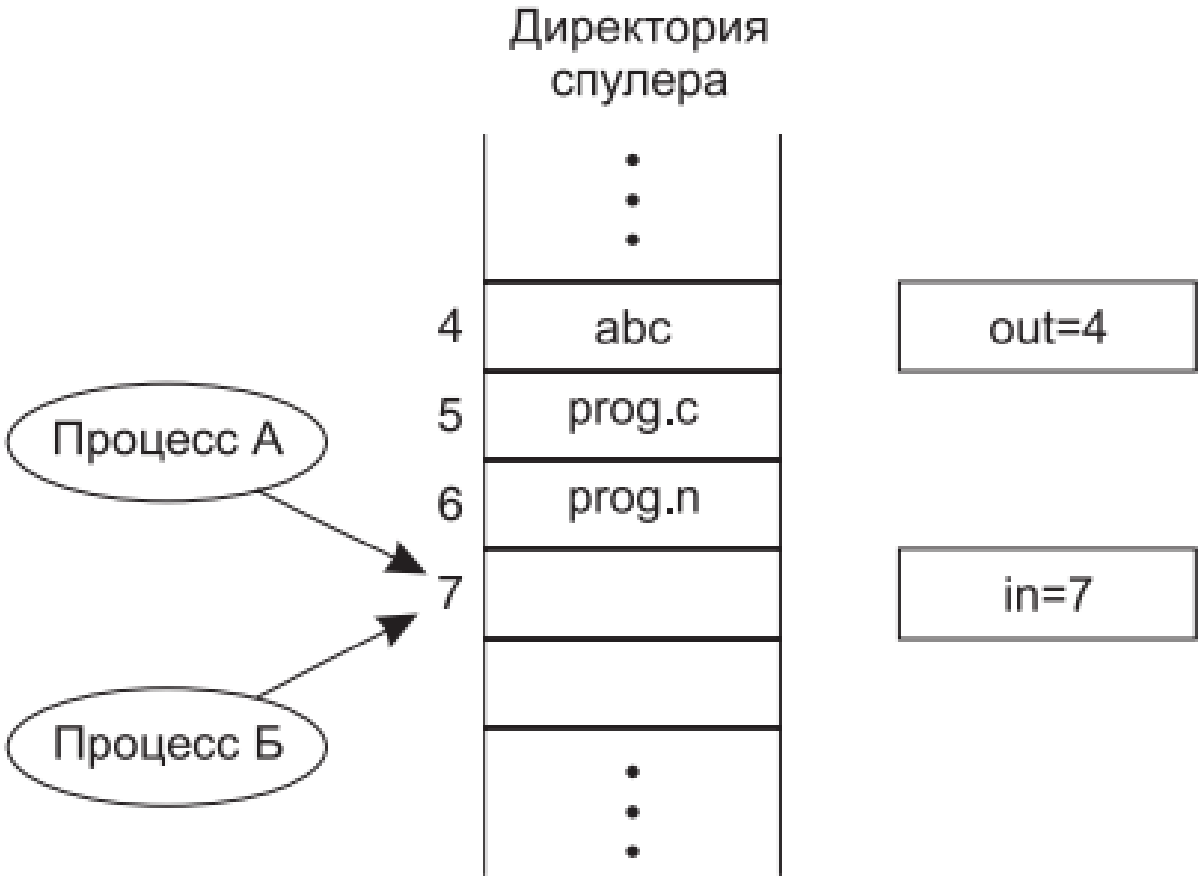
Первый из поставленных вопросов, касающийся передачи информации, применительно к потокам решается значительно легче, поскольку потоки имеют общее адресное пространство

Состязательная ситуация

В некоторых ОС совместно работающие процессы могут использовать общее хранилище данных, доступное каждому из них по чтению и по записи. Это общее хранилище может размещаться в ОП или оно может быть представлено каким-нибудь общим файлом.

Пример

Когда процессу необходимо распечатать какой-нибудь файл, он помещает имя этого файла в специальный каталог спулера. Другой процесс под названием демон принтера периодически ведет проверку наличия файлов для печати, и в том случае, если такие файлы имеются, распечатывает их и удаляет их имена из каталога.



Одновременное стремление двух процессов получить доступ к общей памяти

Критические области

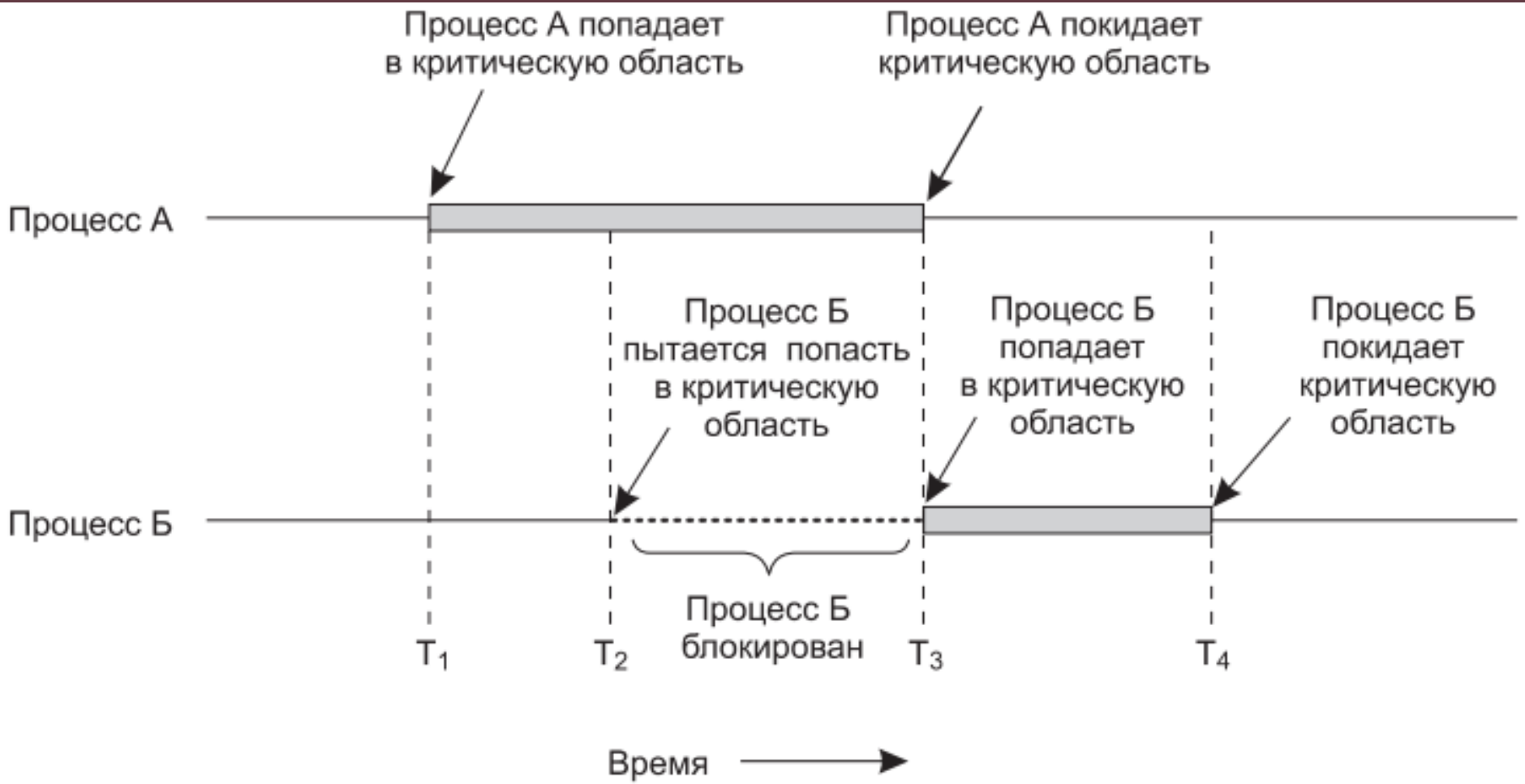
Как же избежать состязательной ситуации? Ключом к предупреждению проблемы в этой и во многих других ситуациях использования общей памяти, общих файлов и вообще чего-нибудь общего может послужить определение способа, при котором в каждый конкретный момент времени доступ к общим данным по чтению и записи может получить только один процесс. Иными словами, нам нужен способ взаимного исключения

Определение

Та часть программы, в которой используется доступ к общей памяти, называется критической областью или критической секцией. Если бы удалось всё выстроить таким образом, чтобы никакие два процесса не находились одновременно в своих критических областях, это позволило бы избежать состязаний.

Условия

1. Два процесса не могут одновременно находиться в своих критических областях.
2. Не должны выстраиваться никакие предположения по поводу скорости или количества центральных процессоров.
3. Никакие процессы, выполняемые за пределами своих критических областей, не могут блокироваться другими процессами.
4. Процессы не должны находиться в вечном ожидании входа в свои критические области.



Взаимное исключение использования критических областей

Запрещение прерываний

Простейшим решением является запрещение всех прерываний каждым процессом сразу после входа в критическую область и их разрешение сразу же после выхода из критической области.

Запрещение прерываний

Запрещение прерываний в большинстве своем является полезной технологией внутри самой операционной системы, но не подходит в качестве универсального механизма взаимных блокировок для пользовательских процессов.

Блокирующие переменные

Когда процессу требуется войти в свою критическую область, сначала он проверяет значение блокирующей переменной. Если оно равно 0, процесс устанавливает его в 1 и входит в критическую область. Если значение уже равно 1, процесс просто ждет, пока оно не станет равно нулю. Таким образом, нулевое значение означает, что ни один из процессов не находится в своей критической области, а единица означает, что какой-то процесс находится в своей критической области.

Блокирующие переменные

К сожалению, эта идея содержит точно такой же фатальный исход, который мы уже видели в примере с каталогом спулера

Блокирующие переменные

К сожалению, эта идея содержит точно такой же фатальный исход, который мы уже видели в примере с каталогом спулера

Строгое чередование

Изначально целочисленная переменная *turn*, показанная на рис. 2.17, равна нулю и отслеживает, чья очередь настала входить в критическую область и про-

```
while(TRUE) {
        while(turn!=0)          /*цикл*/;
        critical_region();
        turn=1;
        noncritical_region();
}
```

а

```
while(TRUE) {
        while(turn!=0)          /*цикл*/;
        critical_region();
        turn=0;
        noncritical_region();
}
```

б

Рис. 2.17. Предлагаемое решение проблемы критической области; процесс 0 (*а*); процесс 1 (*б*). В обоих случаях следует убедиться, что в коде присутствует точка с запятой, завершающая оператор while

Строгое чередование

Изначально целочисленная переменная *turn*, показанная на рис. 2.17, равна нулю и отслеживает, чья очередь настала входить в критическую область и про-