Ключевое слово coproc

Краткий обзор

```
команда coproc [ИМЯ] [перенаправления]
```

Описание

В Bash 4.0 появились *сопроцессы*, функция, безусловно, знакомая пользователям ksh. `coproc` Ключевое слово запускает команду как фоновое задание, настраивая каналы, подключенные как к его stdin, так и к stdout, чтобы вы могли взаимодействовать с ним двунаправленно. При желании у совместного процесса может быть имя `NAME` . Если `NAME` задано, следующая команда **должна быть составной командой**. Если `NAME` задано значение no, то команда может быть либо простой, либо составной.

Идентификатор процесса оболочки, созданной для выполнения сопроцесса, доступен через значение переменной с именем by `NAME` , за которым следует `_PID` суффикс. Например, имя переменной, используемое для хранения PID coproc, начинающегося с no `NAME` given, будет `COPROC_PID` (потому `COPROC` что по умолчанию `NAME`). Встроенная команда wait может использоваться для ожидания завершения сопроцесса. Кроме того, сопроцессами можно управлять с помощью их `jobspec` .

Вернуть статус

Статус возврата сопроцесса - это статус завершения его команды.

Перенаправления

Необязательные перенаправления применяются после настройки каналов. Несколько примеров:

```
# перенаправление stderr в канале
$ coproc { ls thisfiledoesntexist; читать; } 2>&1
[2] 23084
$ IFS= read -ru ${COPROC[0]} x; printf '%s\n' "$x"
ls: невозможно получить доступ к thisfiledoesntexist: нет такого файла или ка
талога
```

```
#пусть выходные данные сопроцесса будут переведены в стандартный вывод
$ { coproc mycoproc { awk '{print "foo" $0;fflush()}'; } >&3; } 3>&1
[2] 23092
$ echo bar >&${mycoproc[1]}
$ foobar
```

Здесь нам нужно сохранить предыдущий файловый дескриптор stdout, потому что к тому времени, когда мы перенаправляем fds сопроцесса, стандартный вывод уже перенаправлен в канал.

Подводные камни

Избегайте конечной подоболочки конвейера

Традиционный обходной путь Ksh, позволяющий избежать подоболочки при выполнении `command | while read` , заключается в использовании сопроцесса. К сожалению, поведение Bash отличается.

В Ksh вы бы сделали:

```
ЕСЛИ
 во время  # запустить сопроцесс |&
ls # производные ksh93 или mksh / pdksh= прочитать -rp  файл; выполнить print
-r -- "$file"; готово # прочитать его вывод
```

В bash:

```
#НЕ РАБОТАЕТ
$ coproc ls
[1] 23232
$ while  IFS= read -ru ${COPROC[0]} строка; сделать  printf '%s \ n' "$line";
сделано
bash: чтение: строка: недопустимая спецификация файлового дескриптора
[1]+ Сделано coproc COPROC ls
```

К тому времени, когда мы начнем чтение из выходных данных сопроцесса, дескриптор файла будет закрыт.

Смотрите Эту запись часто задаваемых вопросов на вики Грега (http://mywiki.wooledge.org/BashFAQ/024) для других обходных путей подоболочек конвейера.

Буферизация

В первом примере мы `fflush()` используем команду GNU awk. Как всегда, при использовании каналов операции ввода-вывода буферизуются. Давайте посмотрим, что произойдет с `sed` :

```
$ coproc sed s/^/foo/
[1] 22981
$ echo bar >&${COPROC[1]}
$ read -t  3 -ru ${COPROC[0]} _; (( $? > 127 )) &&  эхо  "ничего не прочитан
о"
ничего не прочитано
```

Несмотря на то, что этот пример совпадает с первым `awk` примером, `read` он не возвращается, потому что выходные данные ожидают в буфере.

Некоторые обходные пути и дополнительную информацию о проблемах буферизации см. В разделе Часто задаваемых вопросов в вики Грега (http://mywiki.wooledge.org/BashFAQ/009).

фоновые процессы

Файловые дескрипторы сопроцесса доступны только тому процессу, из которого `coproc` он был запущен. Они не наследуются подоболочками.

Вот не очень значимая иллюстрация. Предположим, мы хотим постоянно считывать выходные данные сопроцесса и `echo` результат:

```
#НЕ РАБОТАЕТ
$ coproc awk '{print "foo" $0;fflush()}'
[2] 23100
$ while  IFS= read -ru ${COPROC[0]} x; сделать  printf '%s\n' "$x"; готово &
[3] 23104
bash: строка 243: прочитано: 61: недопустимый дескриптор файла: Неверный деск
риптор файла
```

Это не удается, потому что файловые дескрипторы, созданные родительским файлом, недоступны для подоболочки, созданной & .

Возможный обходной путь:

```
# для стандартного вывода см. Перенаправления выше.
 # это не способ заставить сопроцесс печатать свои выходные
 данные #ПРЕДУПРЕЖДЕНИЕ: только для иллюстрации
$ coproc awk '{print "foo" $0;fflush()}'
[2] 23109
$ exec 3<&${COPROC[0]}
$ while  IFS= read -ru  3 x; do  printf '%s\n' "$x"; готово &
[3] 23110
$ echo bar >&${COPROC[1]}
$ foobar
```

Здесь fd 3 наследуется.

Примеры

Анонимный сопроцесс

В отличие от ksh, в Bash нет настоящих анонимных сопроцессов. Вместо этого Bash присваивает FDs массиву с именем по умолчанию `COPROC` , если `NAME` указано no. Вот пример:

```
$ coproc awk '{print "foo" $0;fflush()}'
[1] 22978
```

Эта команда запускается в фоновом режиме и `coproc` немедленно возвращается. Два новых файловых дескриптора теперь доступны через `COPROC` массив. Мы можем отправлять данные в нашу команду:

```
echo$ bar >&${COPROC[1]}
```

А затем прочитайте его вывод:

```
$ IFS= read -ru ${COPROC[0]} x; printf '%s\n' "$x"
foobar
```

Когда нам больше не нужна наша команда, мы можем отключить ее через ее pid:

```
$ kill $COPROC_PID
$
[1]+ Завершенный coproc COPROC awk '{print "foo" $0;fflush()}'
```

Названный сопроцесс

Использование именованного сопроцесса просто. Нам просто нужна составная команда (например, при определении функции), и результирующие FDS будут присвоены индексированному массиву `NAME` , который мы предоставляем вместо этого.

```
$ coproc mycoproc { awk '{печать "foo" $0;fflush()}' ;}
[1] 23058
$ echo bar >&${mycoproc[1]}
$ IFS= read -ru ${mycoproc[0]} x; printf '%s\n' "$x"
foobar
$ kill $mycoproc_PID
$
[1]+ Завершенный coproc mycoproc { awk '{print "foo" $0;fflush()}'; }
```

Перенаправление вывода скрипта в файл и на экран

```
{
# перенаправляем его вывод на стандартный вывод скрипта# запускаем tee в фоно
вом режиме#!/bin/bash coproc
tee
{лог-файл tee ;}>&3 ; } 3>&1# мы перенаправляем stding и stdout скрипта
 в наш coprocess
 exec >&${tee[1]} 2>&1
```

Соображения о переносимости

- `coproc` Ключевое слово не указано в POSIX(R)
- `coproc` Ключевое слово появилось в версии Bash 4.0-alpha
- `-p` Опция `print` загрузки Bash является NOOP и никоим образом не связана с сопроцессами Bash. Оно распознается только как опция для совместимости с ksh и не имеет никакого эффекта.
- `-p` Опция `read` встроенного Bash конфликтует со всеми kshe и zsh. Эквивалентом в этих оболочках является добавление `\?prompt` суффикса к первому аргументу имени переменной to `read` . т.Е., если первое заданное имя переменной содержит `?` символ, остальная часть аргумента используется в качестве строки запроса. Поскольку эта функция бессмысленна и избыточна, я предлагаю не использовать ее ни в одной из оболочек. Просто перед `read` командой укажите a `printf %s prompt >&2` .

Другие оболочки

ksh93, mksh, zsh и Bash поддерживают нечто, называемое "сопроцессами", которые выполняют примерно одно и то же. ksh93 и mksh имеют практически идентичный синтаксис и семантику для coproc. Оператор *списка*: `|&` добавляется к языку, который запускает предыдущий *конвейер* как сопроцесс (это еще одна причина не использовать специальный

`|&` оператор канала в Bash – его синтаксис конфликтует). `-p` Опция для `read print` встроенных элементов и может затем использоваться для чтения и записи в канал сопроцесса (чей FD еще не известен). Добавляются специальные перенаправления для перемещения последнего созданного сопроцесса в другой FD: `<&p` и `>&p` , после чего к нему можно получить доступ в новом FD с помощью обычного перенаправления, и затем может быть запущен другой сопроцесс, снова используя `|&` .

Сопроцессы zsh очень похожи на ksh, за исключением способа их запуска. zsh добавляет зарезервированное слово оболочки `coproc` в синтаксис конвейера (аналогично тому `time` , как работает ключевое слово Bash), так что следующий конвейер запускается как coproc. Затем можно получить доступ к файлам ввода и вывода coproc и перемещать их, используя те же `read` / `print -p` и перенаправления, которые используются оболочками ksh.

К сожалению, Bash решила пойти против существующей практики в своей реализации coproc, особенно учитывая, что это была последняя из основных оболочек, включившая эту функцию. Однако метод Bash выполняет то же самое, не требуя почти такого же дополнительного синтаксиса. `coproc` Ключевое слово достаточно легко обернуть в функцию таким образом, чтобы она принимала код Bash в качестве обычного аргумента и / или подобного stdin `eval` . Функциональность сопроцесса в других оболочках может быть аналогично обернута для `COPROC` автоматического создания массива.

Только один сопроцесс за раз

Название говорит само за себя, жалуйтесь в список рассылки bug-bash, если хотите большего. Смотрите http://lists.gnu.org/archive/html/bug-bash/2011-04/msg00056.html (http://lists.gnu.org/archive/html/bug-bash/2011-04/msg00056.html) для получения более подробной информации

Возможность использования нескольких сопроцессов в Bash считается "экспериментальной". Bash выдаст ошибку, если вы попытаетесь запустить более одного. Это может быть переопределено во время компиляции с `MULTIPLE_COPROCS` помощью опции. Однако на данный момент все еще существуют проблемы – см. Обсуждение списка рассылки выше.

Смотрите также

- Советы Энтони Тиссена по (http://www.ict.griffith.edu.au/anthony/info/shell/co-processes.hints) совместному процессу - отличное резюме всего, что связано с темой

Обсуждение

Энтони Тиссен (http://www.cit.griffith.edu.au/~anthony/), 2010/06/18 02:18 ()

Можете ли вы выполнить сопроцесс, используя только обычные дескрипторы файлов оболочки? или, возможно, с использованием именованных каналов.

mknod ls_output ls > ls_output &

во время чтения строки; делать

```
. . .
```

готово < ls_output

Ян Шампера, 2010/06/18 08:36 ()

В зависимости от ваших конкретных потребностей, конечно, есть возможность использовать

- именованные каналы
- замена процесса
- замена команды
- ...

Описанные здесь сопроцессы - это просто очень простое в использовании решение для этих задач. Его просто настроить, использовать и завершить сопроцесс.

Энтони Тиссен (http://www.ict.griffith.edu.au/anthony/), 2011/09/28 07:30 ()

Мне кажется, что речь идет о такой сложной задаче, как использование временных именованных каналов.

Конечно, большинство трудностей при обработке сопроцесса часто связаны с обработкой потоков данных, особенно если вам нужно больше, чем просто stdin / stdout.

С момента моего первоначального отзыва я написал то, что, как я надеюсь, является началом руководства по использованию совместных процессов. И да, это может стоить затраченных усилий.

http://www.ict.griffith.edu.au/anthony/info/shell/co-processes.hints (http://www.ict.griffith.edu.au/anthony/info/shell/co-processes.hints) Обратная связь добро пожаловать Энтони Тиссен A.Thyssen@griffith.edu.au (mailto:A.Thyssen@griffith.edu.au)

Ян Шампера, 2011/09/28 20:08 ()

Вау ... отличная коллекция знаний. Я сшил его здесь

Адам Бакстер (http://blog.voltagex.org), 2016/06/03 08:27 ()

Кажется, исчезло. Оно заархивировано по адресу https://web.archive.org/web/20151221031303/http://www.ict.griffith.edu.au/anthony/info/shell/co-processes.hints (https://web.archive.org/web/20151221031303/http://www.ict.griffith.edu.au/anthony/info/shell/co-processes.hints)

SZABÓ Gergely (http://linux.subogero.com), 2012/05/10 08:20 (), 2012/07/01 11:48 ()

Я нашел более простой способ чтения из coprocess. В приведенном ниже примере `szg` показан интерактивный калькулятор, способный преобразовывать шестнадцатеричные числа.

```
$ coproc S { szg; }
[1] 1234
$ echo XffffD >&${S[1]}
$ head -n 1 <&${S[0]}
65535
```

Джордж Касвелл (http://scope-eye.net), 2016/04/13 00:25 ()

Вы должны быть осторожны с тем, что вы прикрепляете к файловым дескрипторам в командной оболочке. Когда программы используют буферизованный ввод-вывод, они могут считывать больше данных, чем им действительно нужно. Например:

```
 coproc S { при чтении строки; повторять "$line"; повторять "Круто, д
а?"; готово; }
```

```
 echo "Тест" >&${S[1]}
```

```
 head -1 <&${S[0]}
```

```
 Тест
```

```
 echo "Test2" >&${S[1]}
```

```
 head -1 <&${S[0]}
```

```
 Тест2
```

Обратите внимание, что "head" никогда не печатает строку "Круто, да?"? Это потому, что внутренне "head" делает что-то вроде этого:

(во время вызова "getline ()" или "fgets ()" или чего-то еще ...):

- Считайте **столько данных, сколько я могу в данный момент**, в свой буфер.
- Сканируйте буфер для первого символа новой строки.
- Вырежьте строку из буфера от начала буфера до первого символа новой строки. Верните его.

Эта агрессивная буферизация выполняется для минимизации времени, затрачиваемого на фактический ввод-вывод. Но это означает, что есть большая вероятность, что процесс будет потреблять больше одной строки ввода из этого файлового дескриптора.

По сути, вам нужно назначить один процесс, отвечающий за чтение строк из этого файлового дескриптора. Оболочкой может быть этот процесс, или вы можете делегировать задачу другому процессу. Но большинство программ не пишутся с предположением, что они совместно используют свой входной файл.

Bash Hackers Wiki