

Операционные Системы

InterProcess Communication

May 11, 2017

Системные вызовы

- ▶ Какие бывают системные вызовы?
Сервисы ОС:

Системные вызовы

- ▶ Какие бывают системные вызовы?
Сервисы ОС:
 - ▶ работа с файловыми системами;

Системные вызовы

- ▶ Какие бывают системные вызовы?

Сервисы ОС:

- ▶ работа с файловыми системами;
- ▶ работа со временем (текущее время, нотификации и прочее)

Системные вызовы

- ▶ Какие бывают системные вызовы?

Сервисы ОС:

- ▶ работа с файловыми системами;
- ▶ работа со временем (текущее время, нотификации и прочее)
- ▶ создание, завершение и управление процессами;

Системные вызовы

- ▶ Какие бывают системные вызовы?

Сервисы ОС:

- ▶ работа с файловыми системами;
- ▶ работа со временем (текущее время, нотификации и прочее)
- ▶ создание, завершение и управление процессами;
- ▶ взаимодействие с другими процессами.

Создание процессов

- ▶ Для создания процессов в Unix-like системах используется вызов `fork`:

Создание процессов

- ▶ Для создания процессов в Unix-like системах используется вызов `fork`:

Создание процессов

- ▶ Для создания процессов в Unix-like системах используется вызов `fork`:
 - ▶ новый процесс является почти точной копией родителя;

Создание процессов

- ▶ Для создания процессов в Unix-like системах используется вызов `fork`:
 - ▶ новый процесс является почти точной копией родителя;
 - ▶ вызывает `fork` один поток, а возвращаются из `fork` уже два потока в двух разных процессах.

Создание процессов

- ▶ Что если в процессе несколько потоков, и один из них вызвал `fork`?

Создание процессов

- ▶ Что если в процессе несколько потоков, и один из них вызвал `fork`?
 - ▶ в новом процессе будет только один поток;

Создание процессов

- ▶ Что если в процессе несколько потоков, и один из них вызвал `fork`?
 - ▶ в новом процессе будет только один поток;
 - ▶ подумайте о блокировках в новом процессе.

Уничтожение процессов

- ▶ Уничтожение процессов состоит из двух частей:

Уничтожение процессов

- ▶ Уничтожение процессов состоит из двух частей:
 - ▶ один из потоков в процессе должен вызвать `exit`
 - ▶ `exit` принимает целочисленный код как аргумент - код возврата.

Уничтожение процессов

- ▶ Уничтожение процессов состоит из двух частей:
 - ▶ один из потоков в процессе должен вызвать `exit`
 - ▶ `exit` принимает целочисленный код как аргумент - код возврата.
 - ▶ родительский процесс (родной или приемный) должен дождаться завершения процесса, используя `waitpid/wait`
 - ▶ `wait/waitpid` могут вернуть код возврата, переданный в `exit`.

Уничтожение процессов

- ▶ Что если в контексте процесса работают несколько потоков?

Уничтожение процессов

- ▶ Что если в контексте процесса работают несколько потоков?
 - ▶ `exit` уничтожает процесс со *всеми* его потоками.

Уничтожение процессов

- ▶ Что если в контексте процесса работают несколько потоков?
 - ▶ `exit` уничтожает процесс со *всеми* его потоками.
- ▶ Что если родитель был уничтожен раньше ребенка?

Уничтожение процессов

- ▶ Что если в контексте процесса работают несколько потоков?
 - ▶ `exit` уничтожает процесс со *всеми* его потоками.
- ▶ Что если родитель был уничтожен раньше ребенка?
 - ▶ другой процесс становится родителем.

Уничтожение процессов

- ▶ Что если в контексте процесса работают несколько потоков?
 - ▶ `exit` уничтожает процесс со *всеми* его потоками.
- ▶ Что если родитель был уничтожен раньше ребенка?
 - ▶ другой процесс становится родителем.
- ▶ Что если не вызвать `waitpid/wait`?

Уничтожение процессов

- ▶ Что если в контексте процесса работают несколько потоков?
 - ▶ `exit` уничтожает процесс со *всеми* его потоками.
- ▶ Что если родитель был уничтожен раньше ребенка?
 - ▶ другой процесс становится родителем.
- ▶ Что если не вызвать `waitpid/wait`?
 - ▶ процесс будет находится в состоянии *Zombie*, пока родитель не вызовет `wait/waitpid`.

Запуск исполняемых файлов

- ▶ Для запуска исполняемого файла используется один из вызовов `exes*`:

Запуск исполняемых файлов

- ▶ Для запуска исполняемого файла используется один из вызовов `exec*`:
 - ▶ при вызове `exec*` ядро ОС "заменяет" старое адресное пространство процесса новым;

Запуск исполняемых файлов

- ▶ Для запуска исполняемого файла используется один из вызовов `exec*`:
 - ▶ при вызове `exec*` ядро ОС "заменяет" старое адресное пространство процесса новым;
 - ▶ передает управление точке входа исполняемого файла (или динамического компоновщика).

Запуск исполняемых файлов

- ▶ Что если в процессе несколько потоков и один из них вызвал `exec*`?

Запуск исполняемых файлов

- ▶ Что если в процессе несколько потоков и один из них вызвал `exit`?
- ▶ все другие потоки будут уничтожены.

Файловые дескрипторы

- ▶ В Unix *все есть файл*:

Файловые дескрипторы

- ▶ В Unix *все есть файл*:
 - ▶ *некоторые* ресурсы, предоставляемые ОС, имеют файловый интерфейс;

Файловые дескрипторы

- ▶ В Unix *все есть файл*:
 - ▶ *некоторые* ресурсы, предоставляемые ОС, имеют файловый интерфейс;
 - ▶ файловый интерфейс: `read/write/close`.

Файловые дескрипторы

- ▶ Файловый дескриптор - некоторый идентификатор ресурса

Файловые дескрипторы

- ▶ Файловый дескриптор - некоторый идентификатор ресурса
 - ▶ в Unix - это обычно просто целое число;

Файловые дескрипторы

- ▶ Файловый дескриптор - некоторый идентификатор ресурса
 - ▶ в Unix - это обычно просто целое число;
 - ▶ 0 - стандартный поток ввода,
 - ▶ 1 - стандартный поток вывода,
 - ▶ 2 - стандартный поток ошибок.

Файловые дескрипторы

- ▶ Способ получения дескриптора зависит от ресурса, которому он соответствует:

Файловые дескрипторы

- ▶ Способ получения дескриптора зависит от ресурса, которому он соответствует:
 - ▶ для обычных файлов можно использовать `open`;

Файловые дескрипторы

- ▶ Способ получения дескриптора зависит от ресурса, которому он соответствует:
 - ▶ для обычных файлов можно использовать `open`;
 - ▶ для каналов (`pipe`-ов) используют `pipe`;

Файловые дескрипторы

- ▶ Способ получения дескриптора зависит от ресурса, которому он соответствует:
 - ▶ для обычных файлов можно использовать `open`;
 - ▶ для каналов (`pipe`-ов) используют `pipe`;
 - ▶ есть много других функций, возвращающих файловый дескриптор.

Дублирование дескрипторов

- ▶ Иногда вам может потребоваться управлять значением файлового дескриптора

Дублирование дескрипторов

- ▶ Иногда вам может потребоваться управлять значением файлового дескриптора
 - ▶ например, чтобы перенаправлять стандартные потоки ввода/вывода в файл/из файла;

Дублирование дескрипторов

- ▶ Иногда вам может потребоваться управлять значением файлового дескриптора
 - ▶ например, чтобы перенаправлять стандартные потоки ввода/вывода в файл/из файла;
 - ▶ для этого можно использовать вызов `dup2`.

Виды ИРС, которые мы не будем рассматривать

- ▶ Сигналы.

Виды ИРС, которые мы не будем рассматривать

- ▶ Сигналы.

Виды IPC, которые мы не будем рассматривать

- ▶ Сигналы.
- ▶ Семафоры.

Виды IPC, которые мы не будем рассматривать

- ▶ Сигналы.
- ▶ Семафоры.
- ▶ Сокеты.

Виды IPC, которые мы будем рассматривать

- ▶ Каналы (неименованные каналы).

Виды IPC, которые мы будем рассматривать

- ▶ Каналы (неименованные каналы).
- ▶ FIFO (именованные каналы).

Виды IPC, которые мы будем рассматривать

- ▶ Каналы (неименованные каналы).
- ▶ FIFO (именованные каналы).
- ▶ Сегменты разделяемой памяти.

Виды IPC, которые мы будем рассматривать

- ▶ Каналы (неименованные каналы).
- ▶ FIFO (именованные каналы).
- ▶ Сегменты разделяемой памяти.
- ▶ Ptrace.

Каналы (Pipes)

- ▶ Канал - односторонний канал связи между процессами (или внутри процесса)

Каналы (Pipes)

- ▶ Канал - односторонний канал связи между процессами (или внутри процесса)
 - ▶ создается вызовом `pipe` - вызов возвращает два "файловых дескриптора";

Каналы (Pipes)

- ▶ Канал - односторонний канал связи между процессами (или внутри процесса)
 - ▶ создается вызовом `pipe` - вызов возвращает два "файловых дескриптора";
 - ▶ через один можно писать, из другого можно читать;

Каналы (Pipes)

- ▶ Канал - односторонний канал связи между процессами (или внутри процесса)
 - ▶ создается вызовом `pipe` - вызов возвращает два "файловых дескриптора";
 - ▶ через один можно писать, из другого можно читать;
 - ▶ работа с `pipe` происходит почти как с обычным файлом: `read/write/close`.

Именованные каналы (FIFO)

- ▶ Почти как pipe, но у FIFO есть имя

Именованные каналы (FIFO)

- ▶ Почти как pipe, но у FIFO есть имя
 - ▶ создается с помощью специальной функции `mknod`;

Именованные каналы (FIFO)

- ▶ Почти как pipe, но у FIFO есть имя
 - ▶ создается с помощью специальной функции `mknod`;
 - ▶ работа с FIFO происходит почти как с обычным файлом: `open/read/write/close`;

Именованные каналы (FIFO)

- ▶ Почти как pipe, но у FIFO есть имя
 - ▶ создается с помощью специальной функции `mknod`;
 - ▶ работа с FIFO происходит почти как с обычным файлом: `open/read/write/close`;
 - ▶ при попытке открыть FIFO поток по умолчанию блокируется, пока кто-нибудь не откроет "другой конец" FIFO.

Сегменты разделяемой памяти

- ▶ Сегмент разделяемой памяти - участок памяти, к которому можно обращаться из нескольких процессов

Сегменты разделяемой памяти

- ▶ Сегмент разделяемой памяти - участок памяти, к которому можно обращаться из нескольких процессов
 - ▶ создать сегмент можно с помощью `shmget`;

Сегменты разделяемой памяти

- ▶ Сегмент разделяемой памяти - участок памяти, к которому можно обращаться из нескольких процессов
 - ▶ создать сегмент можно с помощью `shmget`;
 - ▶ получить указатель на сегмент памяти можно с помощью `shmat`;

Сегменты разделяемой памяти

- ▶ Сегмент разделяемой памяти - участок памяти, к которому можно обращаться из нескольких процессов
 - ▶ создать сегмент можно с помощью `shmget`;
 - ▶ получить указатель на сегмент памяти можно с помощью `shmat`;
 - ▶ "закрыть" сегмент можно с помощью `shmdt`.

Process Trace

- ▶ ptrace не является IPC в привычном понимании:

Process Trace

- ▶ ptrace не является IPC в привычном понимании:
 - ▶ позволяет одному процессу следить за другим;

Process Trace

- ▶ ptrace не является IPC в привычном понимании:
 - ▶ позволяет одному процессу следить за другим;
 - ▶ позволяет одному процессу подсмотреть в память другого;

Process Trace

- ▶ ptrace не является IPC в привычном понимании:
 - ▶ позволяет одному процессу следить за другим;
 - ▶ позволяет одному процессу подсмотреть в память другого;
 - ▶ позволяет одному процессу изменить память другого.