Архитектура ОС UNIX

Санкт-Петербургский государственный политехнический университет

9 ноября 2011

Средства межпроцессного взаимодействия

- Неименованные каналы.
- Семафоры.
- Разделяемая память.
- Очереди сообщений.

Средства межпроцессного взаимодействия

- Неименованные каналы.
- Семафоры.
- Разделяемая память.
- Очереди сообщений.

lock-файлы

Простейший способ организовать работу с разделяемым файлом — создать файл замок. Специальный файл существование которого означает что файл заблокирован.
Для создания файла-замка можно использовать флаг O_EXCL системного вызова open.
При открытии файла из другого процесса необходимо проверить что открытие файла завершилось с ошибкой EEXIST.

lock-файлы

```
while ((fl=open(LOCKFILE,O_CREAT|O_EXCL|O_RDWR
    ,0600))<0) {
  usleep(100000); /* задержка 0.1 секунды */
}
```

lock-файлы

Такой способ блокировки не работает на сетевых файловых системах, таких как nfs (т.к. сиситема не может гарантировать атомарность операйии из-за особенностей nfs).
Для файловой системы nfs необходимо применять следующий алгоритм:

- создать в каталоге с файлом блокировке файл с уникальным именем (например, имя компьютера);
- сделать ссылку с уникального файла на файл-замок;
- проверить с помощью системного вызова stat, что число ссылок на файл увеличилось до 2х.

lock-файлы

```
while (1) {
  if ((fdl = open(UNIQUEFILE, O_CREAT | O_RDWR,
      0600)) < 0) {
    /* ошибка, нужно принять какие-то меры... */
  }
  close(fdl);
  link(UNIQUEFILE, LOCKFILE);
  if (stat(UNIQUEFILE, &statbuf) < 0) {
    /* ошибка, нужно принять какие-то меры... */
  }
  if (statbuf.st_nlink == 2) break;
  unlink(UNIQUEFILE);
  /* создание замка неуспешно, нужно подождать */
  usleep(100000);
}
unlink(UNIQUEFILE);
/* создание файла-замка успешно, можем идти дальше
   */
```

lock-файлы

Достоинства:

- Могут работать на разных компьютерах.

Недостатки:

- Невозможно понять когда удалять «мусорные» lock-файлы.
- Сложная и долгая операция.
- Нет возможности приостановить работу процесса пока есть lock-файл

Неименованные каналы

Все операционные системы семейства UNIX предоставляют простейшее средство однонаправленной пересылки данных между процессами неименованные каналы (pipe).
Канал создаётся системным вызовом pipe.

```
#include <unistd.h>
int pipe(int filedes[2]);
```

Неименованные каналы

Системному вызову передаётся массив из двух элементов, в который этот системный вызов записывает номера двух файловых дескрипторов.
Два файловых дескриптора связаны друг с другом. Данные записываются в файловый дескриптор filedes[1]. Записанные данные можно прочитать из файлового дескриптора filedes[0].

Неименованные каналы

Запись и чтения данных из каналов осуществляется системными вызовами read и write, если канал для записи оказался закрыт и вызвана функуия write, то процесс получает сигнал SIGPIPE и выводит сообщение «Broken pipe».

Неименованные каналы

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main()
{
  int fd[2];
  int pid1, pid2;
  if (pipe(fd) < 0) { perror("pipe"); exit(1); }
  if ((pid1 = fork()) < 0) { perror("fork"); exit
      (1); }
  if (!pid1) {
    /* child 1: "ls -l" */
    dup2(fd[1], 1); close(fd[0]); close(fd[1]);
    execlp("ls", "ls", "-l", NULL);
    perror("execlp"); _exit(1);
  }
```

Неименованные каналы

```
    if ((pid2 = fork()) < 0) { perror("fork"); exit
        (1); }
    if (!pid2) {
      /* child 2: "wc -l" */
      dup2(fd[0], 0); close(fd[0]); close(fd[1]);
      execlp("wc", "wc", "-l", NULL);
      perror("execlp"); _exit(1);
    }
    close(fd[0]); close(fd[1]);
    wait(NULL); wait(NULL);
    return 0;
}
```

Неименованные каналы

Использование каналов предоставляет процессам возможность синхронизовать свою работу. Процесс-читатель может быть приостановлен до тех пор, пока в канале не появятся данные. Процесс-писатель будет приостановлен, пока в канале не освободится место для данных.

```
/* получить жетон */
read(fdp[0], &dummy, sizeof(dummy));
/* выполнить операцию */
fd = open(FILENAME, O_RDWR);
read(fd, &value, sizeof(value));
value++;
lseek(fd, 0L, SEEK_SET);
write(fd, &value, sizeof(value));
close(fd);
/* сдать жетон */
write(fdp[1], &dummy, sizeof(dummy));
```

Неименованные каналы

Достоинства:

- Не требуют модификации файловой системы, поэтому выполняются быстрее.
- Процесс может быть приостановлен (read) до тех пор пока не появятся данные.

Недостатки:

- Взаимодействующие процессы должны быть порождены одним процессом.

Именованные каналы

Именованные каналы также называются FIFO (first-in first-out). Именованные каналы отличаются от неименованных тем, что имеют точку привязки к файловой системе (имя канала). Перед использованием именованный канал должен быть создан с помощью вызова функции mkfifo.

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo ( const char *pathname, mode_t mode
    );
```

Именованные каналы

Для открытия именованного канала используется системный вызов open. Именованный канал может открываться только на чтение или только на запись, но не на чтение и запись одновременно.
Процесс, открывающий именованный канал на запись, будет заблокирован до тех пор, пока не появится процесс, открывший именованный канал на чтение. То же самое верно и в обратную сторону.
Процесс, который собирается сам и читать из именованного канала, и писать в него, должен открыть канал на чтение в неблокирующемся режиме, после этого открыть канал на запись в нормальном режиме, и после сбросить неблокирующий режим на дескрипторе чтения из канала.

Именованные каналы

```
long flags;
/* открываем дескриптор чтения */
fdr = open(PIPE, O_RDONLY | O_NONBLOCK);
/* открываем дескриптор записи */
fdw = open(PIPE, O_WRONLY);
/* сбрасываем флаг неблокирующего доступа */
flags = fcntl(fdr, F_GETFL);
fcntl(fdr, F_SETFL & ~O_NONBLOCK)
```

Именованные каналы

Достоинства:

- Взаимодействующие процессы не обязаны находиться в родственных отношениях.
- Процесс может быть приостановлен до тех пор пока не появятся данные в буфере.

Недостатки:

- Не применимы когда процессы работают на разных компьютерах.

Семафоры

Семафоры одно из старейших средств разделения доступа к критическим ресурсам параллельно работающих процессов.
Дейкстра определил две операции над семафорами: P(s) и V (s). Операция P блокирует семафор, операция V деблокирует семафор.
Операции проверки и блокирования семофоров являются атомарными.
Семафоры хранятся в адресном пространстве ядра и не освобождаются после завершения процесса.

Семафоры

Каждый массив семафоров должен иметь имя, уникальное в системе, чтобы разные никак не связанные друг с другом процессы могли использовать данный массив семафоров. Это число имеет тип key_t и представляет собой целое число. Для облегчения получения этого идентификатора используется функция ftok.

```
#include <sys/types.h>
#include <sys/ipc.h>
key_t ftok(char *pathname, int proj);
```

Она принимает в качестве параметров имя файла, считывает его номер файлового дескриптора и номер устройства, а также добавляет 8 значимых бит из переменной proj. Поэтому файл не должен удаляться или пересоздаваться, чтобы не получить другой идентификатор.

Семафоры

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semget(key_t key, int nsems, int semflg);
```

Функция semget позволяет связать массив семафоров с ключом key. Этот идентификатор должен использоваться при всех обращения к функциям работы с этим массивом семафоров. Параметр nsems задаёт количество семафоров в массиве. Он может быть равен 0, если процесс не пытается создать новый массив семафоров. Параметр semflg определяет флаги создания массива семафоров и права доступа к создаваемому массиву семафоров.

Семафоры

- IPC_CREAT — Если этот флаг установлен, массив семафоров будет создан.
- IPC_EXCL — Если установлен этот флаг, и установлен флаг IPC_CREAT, выполнение завершится с ошибкой, если массив семафоров с таким ключом уже существует.

Если функция завершилась успешно, возвращается идентификатор массива семафоров.

Семафоры

Управление массивом семафоров выполняется при помощи функции semctl.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int  semctl(int semid, int semnum, int cmd,
    unsigned short *arg);
```

Аргумент semid определяет идентификатор массива семафоров, над которым следует про извести операцию. Аргумент semnum задаёт номер семафора в массиве, аргумент cmd задаё выполняемую команду.

Управление массивом семафоров

Команда IPC_RMID удаляет массив семафоров. Все другие процессы, которые заблокированы на этом массиве семафоров разблокируются.
Команда GETALL возвращает значение всех семафоров в массив arg типа unsigned short. Аргумент semnum игнорируется.
Команда GETVAL возвращает значение семафора с номером semnum в массиве семафоров. Аргумент arg игнорируется.
Команда SETALL устанавливает значение всех семафоров. Массив новых значений семафоров передаётся в параметре arg.
Команда SETVAL устанавливает значение семафора с номером semnum. Новое значение семафора передаётся в аргументе arg.

Операции с массивом семафоров

С массивом семафоров можно выполнить следующие действия: блокировка, разблокировка, ожидание.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semop(int semid, struct sembuf *sops, unsigned
    nsops);
```

Функция позволяет задавать сразу несколько операций над массивом семафоров semid.Операции должны находится в массиве, адрес которого передаётся в аргументе sops.

Операции с массивом семафоров

Структура определена следующим образом:

```
struct sembuf
{
    short sem_num;  /* номер семафора (0 - первый) */
    short sem_op;   /* операция над семафором */
    short sem_flg;  /* флаги операции */
};
```

Операции над семафорами будут выполнены в том и только том случае, когда все операции завершатся успешно. Операции над семафорами выполняются атомарно, то есть никакой другой процесс не может наблюдать ситуацию, когда часть операций над семафорами уже завершилась, а часть — нет.

Операции с массивом семафоров

Поле sem_op задает операцию над семафором:

- Если значение sem_op положительно, операция добавляет это значение к значению семафора.
- Если значение sem_op равно 0, операция проверяет значение семафора. Если значение семафора равно 0, операция завершается успешно, и проверяется следующая операция в массиве операций.
- Если значение sem_op отрицательно, значение семафора уменьшается на заданную величину.