IPC: параллельность, синхронизация

Проблемы с «legacy»

- Как должна вести себя система, если во время обработки сигнала придет другой сигнал.
- Предыдущий обработчик ставится после первого вызова пользовательского обработчика или нет.
- Какие сигналы обрабатываются, а какие нет
- Чтобы сигнал не прервал процесс, надо установить обработчик на каждый сигнал
- Один сигнал может придти по разным причинам, как установить причину
- Как дождаться сигнала (while(1);?)
  - pause лучше, но тоже не избирателен.

Новые интерфейсы

- int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);

```
struct  sigaction {
        void    (*sa_handler)(int);
        void    (*sa_sigaction)(int, siginfo_t *, void *);
        int     sa_flags;               /* see signal options below */
        sigset_t sa_mask;               /* signal mask to apply */
};
```

- int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
- int sigpending(sigset_t *set);
- int sigsuspend(const sigset_t *mask);

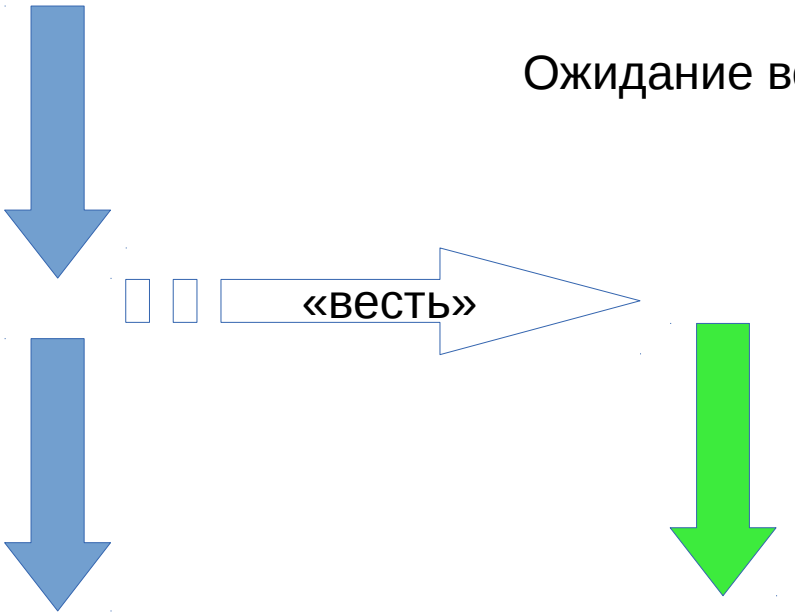
- Все ли проблемы закрыты?
  - Послали сигналы A,B,A, можем получить:
    - A,B
    - B,A
- sigqueue(pid_t pid, int signo, const union sigval value);
  - SIGRTMIN...SIGRTMAX
- pthread_kill(pthread_t thread, int sig);
- sigwait(const sigset_t * restrict set, int * restrict sig);

Синхронизация

Процесс А

Процесс В

Ожидание вести от А



Виды

- Каналы:
  - Если канал пуст, то read ждет
- Сигналы:
  - sigwait/pause + обработчик

Борьба за ресурс

```
#include <pthread.h>
#include <stdio.h>
int a=0;

void * f(void *p){
  int i;
  for(i=0;i<100000;i++)
    a++;
  return p;
}

int main(){
  pthread_t t1,t2;
  pthread_create(&t1,NULL,f,NULL);
  pthread_create(&t2,NULL,f,NULL);
  pthread_join(t1,NULL);
  pthread_join(t2,NULL);
  printf("a=%d\n",a);
}
```

Результаты:

- a=101070
- a=134935
- a=110567
- a=108283

Для современных процессоров надо увеличить число нитей и константу 100000

- Ресурс: переменная a
- Критическая секция: a++ (не атомарна)
  - A → R
  - R++
  - R → A
  - Кеш
- Решение:
  - Семафоры Дейкстры

Семафоры Дейкстры

- Переменная S (целая)
- P(S):
  - Если S>0, то S--
  - Иначе ждать S>0
- V(S):
  - S++
- P,S — атомарны
- Пример: тележки в супермаркете.

2 набора IPC-функций

- [Объект][Действие] — старый интерфейс (Sys V):
  - semget/semop
  - shmget/shmctl
  - msgget/msgsnd
- [Объект]_[Действие] — новые POSIX-функции:
  - sem_open/sem_init
  - shm_open/shm_unlink
  - mq_open/mq_send

POSIX-семафоры

- sem_init(sem_t *sem, int pshared, unsigned int value);
- sem_destroy(sem_t *sem);
- sem_post(sem_t *sem);
- sem_wait(sem_t *sem);
- sem_t * sem_open(const char *name, int oflag, ...);
- sem_close(sem_t *sem);

Двоичные семафоры

- Можно использовать с max=1
- Можно mutex:
  - pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);
  - pthread_mutex_lock(pthread_mutex_t *mutex);
  - pthread_mutex_unlock(pthread_mutex_t *mutex);
  - pthread_mutex_destroy(pthread_mutex_t *mutex);

Отличие mutex от семафора

- PTHREAD_MUTEX_DEFAULT или PTHREAD_MUTEX_NORMAL
- PTHREAD_MUTEX_ERRORCHECK
- PTHREAD_MUTEX_RECURSIVE

Аппаратно

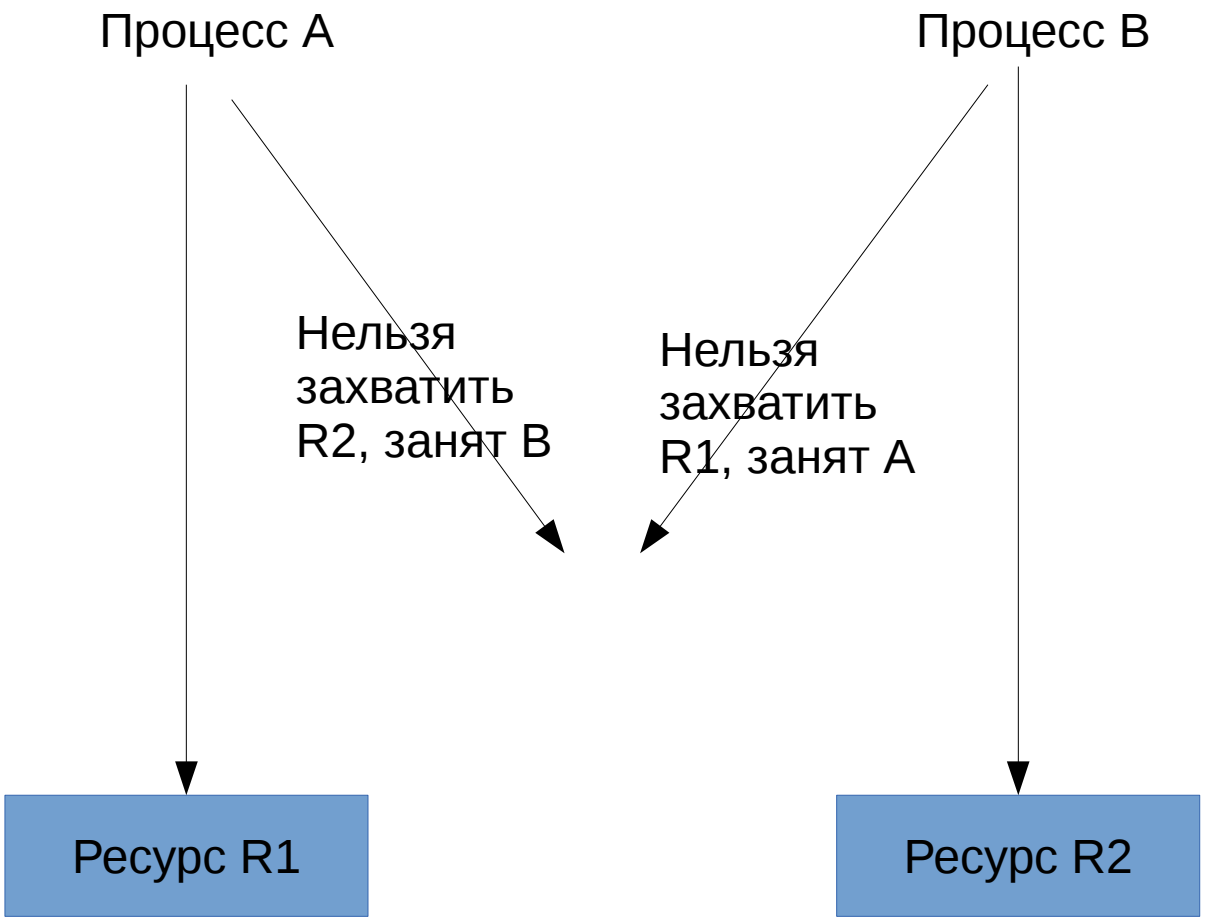
Блокировка шины данных (префикс lock для intel):

- lock btr/bts:
  - Проверить и сбросить/установить бит
- lock cmpxchg/cpxchgq/cmpexch

Mutex vs spinlock

- Mutex: переключение (syscall/int) → ядро → блокировка шины, освобождение процессора
- Spinlock: внутри нитей одного процесса
- Сценарий: большое время работы в критической секции:
  - Mutex: занято одно ядро (работает одна нить)
  - Spinlock: заняты все ядра

Взаимная блокировка (deadlock)



A:
- Захват R1
- Захват R2
- Работа

B:
- Захват R2
- Захват R1
- Работа

Обедающие философы

- Дано:
  - N философов в саду
  - Беседка на N персон, N вилок
- Время от времени философы хотят есть:
  - Зайти в беседку
  - Взять левую вилку (левой рукой), взять правую
  - Есть
- Проблема:
  - если придут одновременно, то умрут от голода
  - если пускать лишь по одному, то большая очередь — умрут от голода

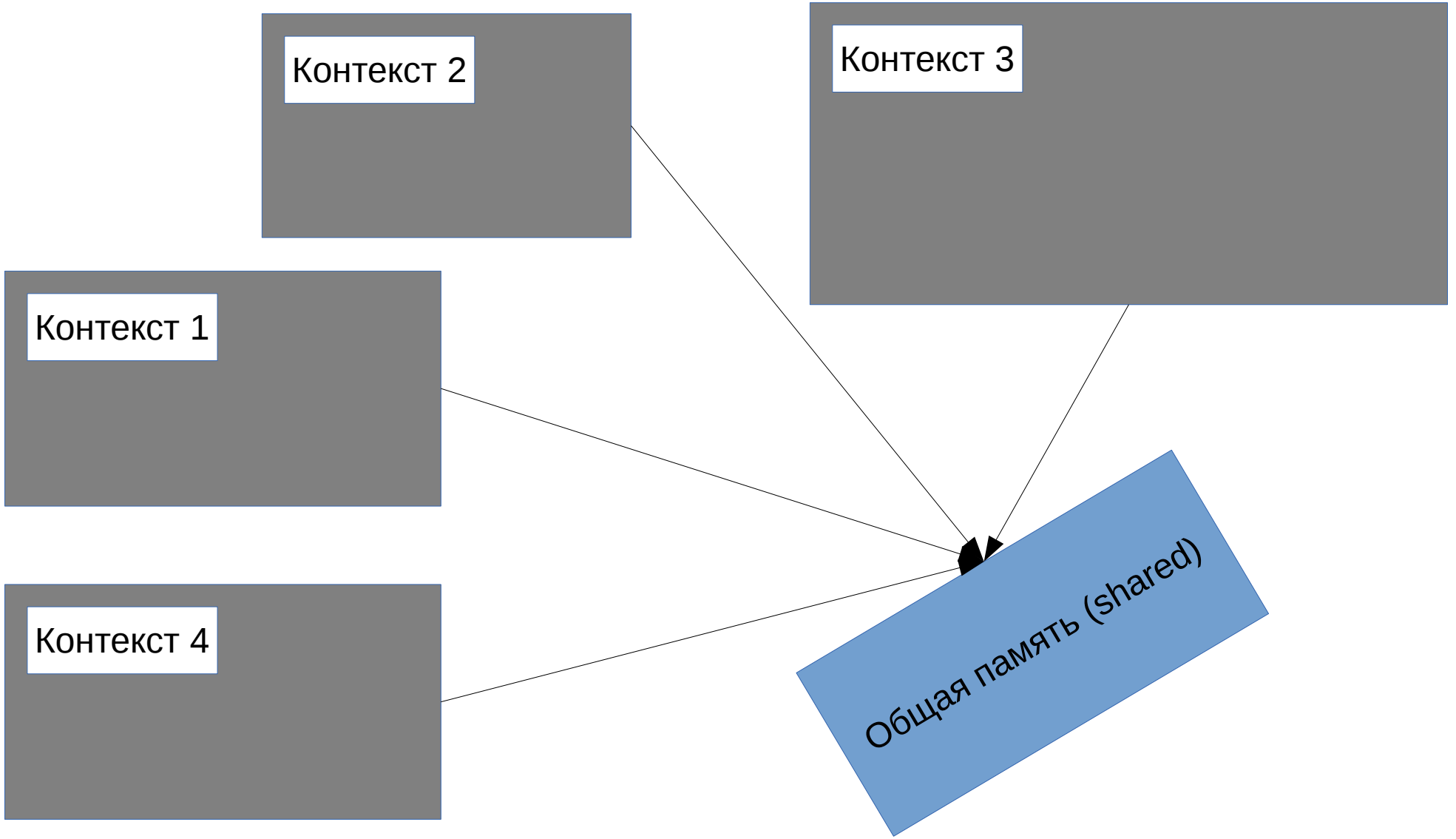
Обход множества дескрипторов

- Чтобы узнать, есть данные или нет, нужен системный вызов. А если дескрипторов много?
- Решение:
  - poll
  - select
- Принцип схож: сообщаем множество (select) или список (poll) интересующих нас дескрипторов, получаем информацию, в каких из них появились данные.

Обход множества дескрипторов: проблемы

- Процесс передает данные в своей памяти (принадлежит процессу)
- Во время обработки системного вызова ядром нет гарантии, что эти данные в физической памяти (а не на диске)
  - Выгрузка в swap может происходить на другом ядре
- Реализация (bsd): копировать аргументы в память ядра
- Развитие:
  - Linux: epoll
  - FreeBSD: kqueue (дескрипторы + сигналы + ...)

Разделяемая память



Разделяемая память

- У каждого из процессов есть исключительно своя память и память (S), к которой имеют доступ все четыре
- У каждого из них свои адреса (часть контекста)

Еще раз про страницы

Виртуальная1	Физическая10
Виртуальная2	Физическая3
Виртуальная3	swap
Виртуальная4	file

I386: страница=4Kb:

- 20 бит — адрес страницы
- 12 бит — смещение внутри страницы
- Ширина таблицы: 20+20

Варианты:

- Нет правила (0x0, 0x1,... — нулевая страница)
- Выгружена на диск
- Надо загрузить с диска

Не смогли преобразовать — прерывание

Есть общая, но осталась личная

K1.0	Физ1
K1.1	Физ3
S1.0	Физ100
S1.1	Физ40

K2.0	Физ10
K2.1	Физ33
S1.0	Физ100
S1.1	Физ40

K3.0	Физ7
K3.1	Физ8
S1.0	Физ100
S1.1	Физ40

K4.0	Физ20
K4.1	Физ13
S1.0	Физ100
S1.1	Физ40