

Асинхронный ввод/вывод

Системные вызовы и библиотеки Unix SVR4

Иртегов Д.В.

ФФ/ФИТ НГУ

Электронный лекционный курс подготовлен в рамках реализации

Программы развития НИУ-НГУ на 2009-2018 г.г.

Зачем нужен асинхронный в/в

- Создавать нить на каждую операцию ввода/вывода дорого
- Порядок исполнения нитей не гарантируется
- Некоторые устройства передают данные только после явного запроса (`select/poll` не подходит)
- Задержки при работе с такими устройствами чувствительны для приложений жесткого реального времени

Асинхронный ввод/вывод

- Две реализации
 - POSIX 1b
 - Solaris AIO

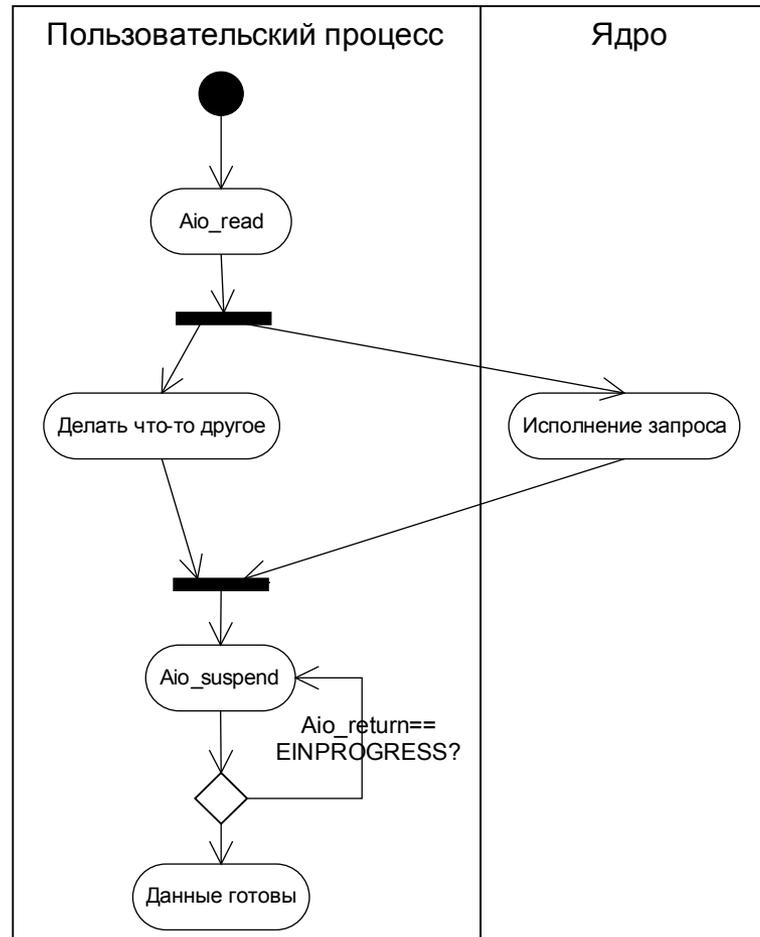
Реализация POSIX 1b

- `aio_read`, `aio_write`, `aio_suspend`
- Используют структуру `aio_cb`
- Размещены в библиотеке `librt.so`
 - Компилировать с ключом `-lrt`
- Поддерживаются в Linux

Solaris AIO

- `aioread`, `aiowrite`
- Нестандартные
- Размещены в библиотеке `libaio`
 - Компилировать с ключом `-laio`

Схема асинхронного ввода



aio_read/write(3RT)

```
cc [ flag... ] file... -lrt [ library... ]  
#include <aio.h>
```

```
int aio_write(struct aiocb *aiocbp);  
int aio_read(struct aiocb *aiocbp);
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

Успех/неуспех

struct aiocb

```
int          aio_fildes; // file descriptor
off_t       aio_offset; // file offset
volatile void* aio_buf;  // location of buffer
size_t      aio_nbytes; // length of transfer
int         aio_reqprio;
            // request priority offset
struct sigevent aio_sigevent;
            // signal number and value
int         aio_lio_opcode;
            // operation to be performed
```

lio_listio(3RT)

```
cc [ flag... ] file... -lrt [ library... ]  
#include <aio.h>  
  
int lio_listio(int mode,  
               struct aiocb *restrict const list[],  
               int nent, struct sigevent *restrict sig);
```

aio_suspend(3RT)

```
cc [ flag... ] file... -lrt [ library... ]  
#include <aio.h>
```

```
int aio_suspend(  
    const struct aiocb * const list[],  
    int nent,  
    const struct timespec *timeout);
```

aio_return(3RT)

```
cc [ flag... ] file... -lrt [ library... ]  
#include <aio.h>
```

```
ssize_t aio_return(struct aiocb *aiocbp);
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

Количество прочитанных байт, если запрос завершился успешно

-1, если запрос завершился ошибкой или еще не завершился.

Если запрос еще не завершился,
errno==EINPROGRESS

aio_error(3RT)

```
cc [ flag... ] file... -lrt [ library... ]  
#include <aio.h>
```

```
int aio_error(const struct aiocb *aiocbp);
```

Пример использования

```
const char req[]="GET / HTTP/1.0\r\n\r\n";
int main() {
    int s;
    static struct aiocb readrq;
    static const struct aiocb *readrqv[2]={&readrq, NULL};
/* Открыть сокет [...] */
    memset(&readrq, 0, sizeof readrq);
    readrq.aio_fildes=s;
    readrq.aio_buf=buf;
    readrq.aio_nbytes=sizeof buf;
    if (aio_read(&readrq)) /* ... */

    write(s, req, (sizeof req)-1);
    while(1) {
        aio_suspend(readrqv, 1, NULL);
        size=aio_return(&readrq);
        if (size>0) {
            write(1, buf, size);
            aio_read(&readrq);
        } else if (size==0) {
            break;
        } else if (errno!=EINPROGRESS) {
            perror("reading from socket");
        }
    }
}
```

Асинхронный ввод с сигналами

- При формировании запроса aio можно указать номер сигнала и параметры, которые следует передать обработчику
- При завершении запроса будет послан этот сигнал
- Solaris AIO использует SIGIO
- С POSIX AIO используют SIGIO, SIGPOLL, SIGUSR1/2, SIGRTXXX

Почему именно сигнал?

- Сигнал прерывает блокирующиеся системные вызовы

Установка обработчика сигнала с параметрами

```
void sigiohandler(int signo, siginfo_t *info,  
                  void *context);  
struct sigaction sigiohandleraction;  
  
memset(&sigiohandleraction, 0,  
       sizeof sigiohandleraction);  
sigiohandleraction.sa_sigaction=sigiohandler;  
sigiohandleraction.sa_flags=SA_SIGINFO;  
sigiohandleraction.sa_mask=set;  
sigaction(SIGIO, &sigiohandleraction, NULL);  
  
readrq.aio_sigevent.sigev_notify=SIGEV_SIGNAL;  
readrq.aio_sigevent.sigev_signo=SIGIO;  
readrq.aio_sigevent.sigev_value.sival_ptr=&readrq;
```

sigaction(2)

```
#include <signal.h>
```

```
int sigaction(int sig,  
              const struct sigaction *restrict act,  
              struct sigaction *restrict oact);
```

- struct sigaction

```
void (*sa_handler)();  
void (*sa_sigaction)(int, siginfo_t *, void *);  
sigset_t sa_mask;  
int sa_flags;
```

sigpause(3C)

```
#include <signal.h>  
int sigpause(int sig);
```

- Удаляет сигнал sig из маски и входит в pause(2)

Sigsetjmp/siglongjmp

```
#include <setjmp.h>
```

```
int sigsetjmp(sigjmp_buf env, int savemask);
```

```
void siglongjmp(sigjmp_buf env, int val);
```

- Siglongjmp сообщает среде исполнения, что мы вышли из обработчика сигнала (простой longjmp этого не делает).
- При выходе из обработчика сигнала восстанавливается маска сигналов процесса
- Используя параметр savemask, можно восстанавливать маску сигналов на момент sigsetjmp либо на момент прихода сигнала

Зачем использовать sigsetjmp/siglongjmp?

- Защита от ошибки потерянного пробуждения

```
aio_read(readrq);
```

<<< прилетает сигнал

```
pause();
```

Асинхронный ввод на сигналах (main)

```
if (aio_read(&readrq)) {
    perror("aio_read");
    exit(1);
}

/* Everything is ready, send HTTP request */
write(s, req, (sizeof req)-1);

/* main loop :) */
if (!sigsetjmp(toexit, 1)) {
    while(1) sigpause(SIGIO);
}
write(2, "That's all, folks\n", 18);
```

Асинхронный ввод на сигналах (обработчик)

```
sigjmp_buf toexit;

void sigiohandler(int signo, siginfo_t *info, void *context) {
    struct aiocb * req;
    if (signo!=SIGIO || info->si_signo!=SIGIO) return;

    req=(struct aiocb *)info->si_value.sival_ptr;
    if (aio_error(req)==0) {
        /* it's complete!!! */
        size_t size;
        size=aio_return(req);
        if (size==0) {
            /* EOF */
            siglongjmp(toexit, 1);
        }
        write(1, req->aio_buf, size);
        /* schedule next read */
        aio_read(req);
    }
}
```