Программирование в Linux

Межпроцессное взаимодействие. Каналы. Очереди. Общая память

Каналы — односторонние средства общения

Pipe
— Неименованный канал
— Два открытых дескриптора RO [0] и WO [1]
— Пользоваться могут только родственные процессы
— Существует пока открыт хоть один дескриптор

Fifo
— Именованный канал, очередь
— Объект в файловой системе
— Все, кто знает к нему путь, могут подключиться
— Существует до тех пор, пока не удалят (rm)

```c
#include <unistd.h>

/* On Alpha, IA-64, MIPS, SuperH, and SPARC/SPARC64; see NOTES */
struct fd_pair {
    long fd[2];
};
struct fd_pair pipe();

/* On all other architectures */
int pipe(int pipefd[2]);

#define _GNU_SOURCE             /* See feature_test_macros(7) */
#include <fcntl.h>              /* Obtain O_* constant definitions */
#include <unistd.h>

int pipe2(int pipefd[2], int flags);
```

```c
#include <sys/types.h>
#include <sys/stat.h>

int mkfifo(const char *pathname, mode_t mode);

#include <fcntl.h>           /* Definition of AT_* constants */
#include <sys/stat.h>

int mkfifoat(int dirfd, const char *pathname, mode_t mode);
```

Особые случаи

- Нет данных в канале/буфер заполнился — блокировка
- Именованный канал открыли только с одной стороны — блокировка
- Все WO дескрипторы закрыты — eof для читателя
- Все RO дескрипторы закрыты — SIGPIPE при записи
- Запись менее PIPE_BUF байт атомарна (данные не перемешиваются при записи из многих процессов)

Блокировки отключаются флагом O_NONBLOCK.
При завязке на SIGPIPE стоит выставлять O_CLOEXEC (автозакрытие при exec)

Специальные операции для каналов

```
#define _GNU_SOURCE         /* See feature_test_macros(7) */
#include <fcntl.h>

ssize_t splice(int fd_in, loff_t *off_in, int fd_out,
               loff_t *off_out, size_t len, unsigned int flags);
```

Переслать len байт из fd_in в fd_out, избегая копирований userspace/kernelspace.
Один из дескрипторов должен быть каналом

```
#include <fcntl.h>

ssize_t tee(int fd_in, int fd_out, size_t len, unsigned int flags);
```

Дублировать len байт из fd_in в fd_out, не извлекая из fd_in.
Оба дескриптора должны быть каналами

```
#include <fcntl.h>
#include <sys/uio.h>

ssize_t vmsplice(int fd, const struct iovec *iov,
                 unsigned long nr_segs, unsigned int flags);
```

fd — один из концов pipe.
Отобразить в канал страницы адресного пространства или
Прочитать из канала в эти страницы

```cpp
int main() {
    struct pipe_ends {
        int read_end;
        int write_end;
    };

    pipe_ends pipe_ends;
    if (pipe(reinterpret_cast<int(&)[2]>(pipe_ends)) != 0) {
        perror("pipe");
        return EXIT_FAILURE;
    }

    pid_t child = fork();

    if (child > 0) {
        close(pipe_ends.read_end);
        WritePipe(pipe_ends.write_end);
    }

    if (child == 0) {
        close(pipe_ends.write_end);
        ReadPipe(pipe_ends.read_end);
    }

    perror("fork");
    return EXIT_FAILURE;
}
```

```cpp
    static constexpr auto TerminateHander = [](int s
        terminated.store(true);
        termination_signal.store(s);
    };

    [[noreturn]] void WritePipe(int pipefd) {
        printf("Start writer (%d)\n", getpid());
        signal(SIGPIPE, TerminateHander);
        signal(SIGINT, SIG_IGN);

        std::vector<std::vector<Point>> points = {
            { {1, 2}, {3, 4} },
            { {5, 6} }
        };

        std::vector<iovec> data_ptrs;
        for (const auto& line : points) {
            data_ptrs.push_back(iovec {
                const_cast<void*>(
                    reinterpret_cast<const void*>(
                        line.data())),
                line.size() * sizeof(Point)
            });
        }

        while (!terminated.load()) {
            vmsplice(pipefd,
                     data_ptrs.data(),
                     data_ptrs.size(),
                     SPLICE_F_MORE);
            if (terminated.load()) {
                break;
            }
            sleep(1);
        }

        printf("Writer (%d) is terminated. "
               "Termination signal is %d \n",
               getpid(),
               termination_signal.load());
        close(pipefd);
        exit(EXIT_SUCCESS);
    }
```

```cpp
    [[noreturn]] void ReadPipe(int pipefd) {
        printf("Start reader (%d)\n", getpid());

        signal(SIGTERM, TerminateHander);
        signal(SIGINT, TerminateHander);

        Point cur;
        while (!terminated.load()) {
            auto sz = read(pipefd,
                           reinterpret_cast<void*>(&cur),
                           sizeof(cur));
            if (terminated.load()) {
                break;
            }
            if (sz != sizeof(cur)) {
                printf("Reader (%d) "
                       "read something unexpected",
                       getpid());
                break;
            }
            printf("Reader (%d) got Point {%d, %d}\n",
                   getpid(), cur.x, cur.y);
            sleep(1);
        }
        if (terminated.load()) {
            printf("Reader (%d) is manually terminated\n",
                   getpid());
        } else {
            printf("Reader (%d) is spuriously terminated\n",
                   getpid());
        }
        close(pipefd);
        exit(EXIT_SUCCESS);
    }
```

```cpp
int main() {
    signal(SIGINT, SIG_IGN);
    signal(SIGTERM, SIG_IGN);

    if (mkfifo(FIFO_FILE_NAME, 0644) != 0) {
        perror("mkfifo");
        return EXIT_FAILURE;
    }

    struct DeferDeletion {
        ~DeferDeletion() {
            remove(FIFO_FILE_NAME);
        }
    } deleter;

    pid_t reader = clone([](void*){
        RunReader();
        return EXIT_SUCCESS;
    }, reader_stack + STASK_SIZE, SIGCHLD, NULL);

    if (reader < 0) {
        perror("clone");
        return EXIT_FAILURE;
    }

    pid_t writer = clone([](void*){
        RunWriter();
        return EXIT_SUCCESS;
    }, writer_stack + STASK_SIZE, SIGCHLD, NULL);

    if (writer < 0) {
        perror("clone");
        kill(reader, SIGKILL);
        return EXIT_FAILURE;
    }

    waitpid(reader, NULL, 0);
    waitpid(writer, NULL, 0);
    return EXIT_SUCCESS;
}
```

```cpp
[[noreturn]] void RunWriter() {
    printf("Start writer (%d)\n", getpid());
    signal(SIGPIPE, TerminateHander);
    signal(SIGINT, SIG_IGN);

    int pipefd = open(FIFO_FILE_NAME, O_WRONLY);
    if (pipefd < 0) {
        perror("open");
        std::exit(EXIT_FAILURE);
    }

    std::vector<std::vector<Point>> points = {...
```

```cpp
[[noreturn]] void RunReader() {
    printf("Start reader (%d)\n", getpid());

    signal(SIGTERM, TerminateHander);
    signal(SIGINT, TerminateHander);

    int pipefd = open(FIFO_FILE_NAME, O_RDONLY);
    if (pipefd < 0) {
        perror("open");
        std::exit(EXIT_FAILURE);
    }

    Point cur;
    while (!terminated.load()) {...
    if (terminated.load()) {...
    close(pipefd);
    exit(EXIT_SUCCESS);
}
```

“popen” — подсоединяемся к stdin/stdout запускаемого процесса

```cpp
struct pipe_ends {
    int read_end;
    int write_end;
};

pid_t MyPopen(const char* command, pipe_ends* communication_pipe) {
    pipe_ends stdin_ends;
    pipe_ends stdout_ends;
    if (pipe2(reinterpret_cast<int(&)[2]>(stdin_ends), O_CLOEXEC) != 0) {
        return -1;
    }
    if (pipe2(reinterpret_cast<int(&)[2]>(stdout_ends), O_CLOEXEC) != 0) {
        return -1;
    }
    communication_pipe->read_end = stdout_ends.read_end;
    communication_pipe->write_end = stdin_ends.write_end;

    pid_t child = fork();
    if (child < 0) {
        return -1;
    }

    if (child != 0) {
        close(stdin_ends.read_end);
        close(stdout_ends.write_end);
        return child;
    }

    dup2(stdin_ends.read_end, STDIN_FILENO); // redefine STDIN/STDOUT
    dup2(stdout_ends.write_end, STDOUT_FILENO);
    execl("/bin/sh", "sh", "-c", command, NULL);
    perror("execl");
    exit(EXIT_FAILURE);
}
```

dup(fd) — клонировать дескриптор
dup2(fd, newfd) — клонировать дескриптор fd, назначить клону значение newfd

Очередь сообщений — высокоуровневый named pipe

Общение не отдельными байтами, а цельными пачками — сообщениями

POSIX (man mq_overview)

Library interface	System call
mq_close(3)	close(2)
mq_getattr(3)	mq_getsetattr(2)
mq_notify(3)	mq_notify(2)
mq_open(3)	mq_open(2)
mq_receive(3)	mq_timedreceive(2)
mq_send(3)	mq_timedsend(2)
mq_setattr(3)	mq_getsetattr(2)
mq_timedreceive(3)	mq_timedreceive(2)
mq_timedsend(3)	mq_timedsend(2)
mq_unlink(3)	mq_unlink(2)

System V (man svipc)

msgget — открыть/создать очередь
msgctl — настроить параметры
msgsnd — отправить сообщение
msgrcv — принять сообщение

Существуют до тех пор, пока не удалят, или до выгрузки OS

POSIX mq — очередь с приоритетами

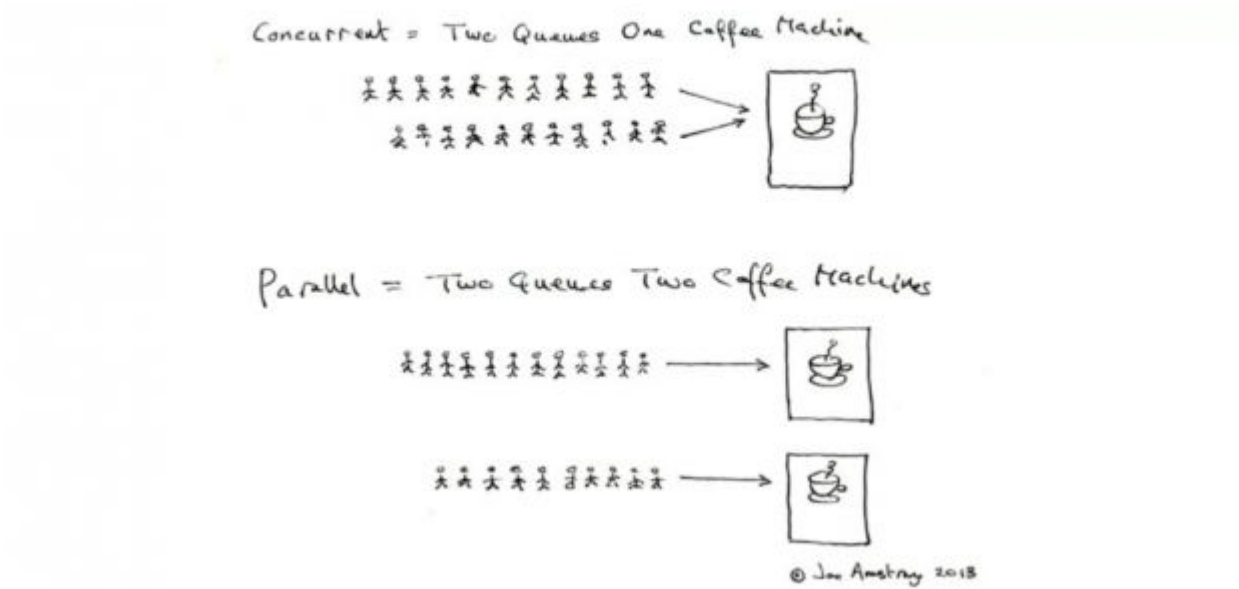
```cpp
constexpr auto MQUEUE_NAME = "/my_mqueue";

struct Point
{
    int x;
    int y;
};
```

```cpp
int main()
{
    signal(SIGINT, SIG_IGN);
    signal(SIGTERM, SIG_IGN);

    mq_attr queue_attrs;
    queue_attrs.mq_maxmsg = 10; // max messges c
    queue_attrs.mq_msgsize = sizeof(Point);
    mqd_t mq = mq_open(MQUEUE_NAME,
                       O_CREAT | O_RDWR,
                       0644, &queue_attrs);
    if (mq == mqd_t(-1))
    {
        perror("mq_open");
        return EXIT_FAILURE;
    }
    mq_close(mq);
    struct DeferDeletion
    {
        ~DeferDeletion()
        {
            mq_unlink(MQUEUE_NAME);
        }
    } deleter;
```

```cpp
[[noreturn]] void RunWriter()
{
    printf("Start writer (%d)\n", getpid());
    signal(SIGINT, TerminateHander);
    signal(SIGTERM, TerminateHander);

    int pipefd = mq_open(MQUEUE_NAME, O_WRONLY);
    if (pipefd < 0)...
    mq_attr attrs;
    mq_getattr(pipefd, &attrs);
    printf("Queue attrs: maxmessagesize=%ld, "
           "max_messages=%ld\n",
           attrs.mq_msgsize,
           attrs.mq_maxmsg);

    if (sizeof(Point) != attrs.mq_msgsize)...

    std::vector<Point> points =
        {{1, 2}, {3, 4}, {5, 6}};

    while (!terminated.load())
    {
        for (const auto &p : points)
        {
            mq_send(pipefd,
                    (const char *)(&p),
                    sizeof(p), 0);
        }
        sleep(1);
    }

    printf("Writer (%d) is terminated. "
           "Termination signal is %d \n",
           getpid(),
           termination_signal.load());
    mq_close(pipefd);
    exit(EXIT_SUCCESS);
}
```

```cpp
[[noreturn]] void RunReader()
{
    printf("Start reader (%d)\n", getpid());

    signal(SIGTERM, TerminateHander);
    signal(SIGINT, SIG_IGN);

    mqd_t pipefd = mq_open(MQUEUE_NAME, O_RDONLY);
    if (pipefd < 0)...
    mq_attr attrs;
    mq_getattr(pipefd, &attrs);
    printf("Queue attrs: maxmessagesize=%ld, "
           "max_messages=%ld\n",
           attrs.mq_msgsize,
           attrs.mq_maxmsg);

    if (sizeof(Point) != attrs.mq_msgsize)...

    Point cur;
    timespec timeout{1, 0};
    while (!terminated.load())
    {
        auto sz = mq_timedreceive(pipefd,
                                  (char *)&cur,
                                  sizeof(cur),
                                  nullptr, // ignore prio
                                  &timeout);
        if (sz < 0 && errno == ETIMEDOUT)
        {
            printf("Reader timeout!\n");
            break;
        }
        if (terminated.load())...
        if (sz != sizeof(cur))...
        printf("Reader (%d) got Point {%d, %d}\n",
               getpid(), cur.x, cur.y);
        sleep(1);
    }
    if (terminated.load())...
    else...
    mq_close(pipefd);
    exit(EXIT_SUCCESS);
}
```

Конкурентное vs параллельное исполнение



Shared memory

Анонимная

mmap(... MAP_ANONYMOUS | MAP_SHARED)

Существует пока все использующие процессы не размапят или не завершатся.

Именованная

POSIX: (man shm_overview)

```
fd = shm_open(“/name”)
ftruncate(fd, N)
ptr = mmap(..., fd);
…
shm_unlink(“/name”);
```

Именованная

System V: (man svipc)

shmget — создать
shmat — получить отображение
shmctl — настроить
shmdt — размаппить отображение

Именованная память существует, пока не удалят или до выгрузки OS

Синхронизация конкурентного доступа

Семафор — счетчик с блокировкой. Можно увеличивать и уменьшать
Попытка опустить значение ниже нуля влечет блокировку.

Мьютекс — бинарный семафор.

POSIX семафоры (man sem_overview)

sem_open — открыть/создать именованный семафор
sem_close
sem_unlink — удалить именованный семафор
sem_init — создать анонимный семафор в указанной области памяти
sem_destroy
sem_post — увеличить счетчик на единицу
sem_wait — уменьшить счетчик на единицу

Есть System V семафоры.

Fast userspace mutex (futex)

- Семафоры — объекты ядра. Работа с ними — по два переключения в kernelspace.
- Переключение нужно только для блокировки/уведомления.
- Часто семафор свободен. И переключение в kernelspace избыточно
- Можно перенести проверку в userspace

```
23  static int
24  futex(uint32_t *uaddr, int futex_op, uint32_t val,
25        const struct timespec *timeout, uint32_t *uaddr2, uint32_t val3)
26  {
27      return syscall(SYS_futex, uaddr, futex_op, val,
28                     timeout, uaddr2, val3);
29  }
```

```
33  static void fwait(uint32_t *futexp) {
34      while (1) {
35          /* Is the futex available? */
36          const uint32_t one = 1;
37          if (atomic_compare_exchange_strong(futexp, &one, 0))
38              break; /* Yes */
39
40          /* Futex is not available; wait. */
41          long s = futex(futexp, FUTEX_WAIT, 0, NULL, NULL, 0);
42          if (s == -1 && errno != EAGAIN)
43              errExit("futex-FUTEX_WAIT");
44      }
45  }
```

```
50  static void fpost(uint32_t *futexp) {
51      const uint32_t zero = 0;
52      if (atomic_compare_exchange_strong(futexp, &zero, 1)) {
53          long s = futex(futexp, FUTEX_WAKE, 1, NULL, NULL, 0);
54          if (s == -1)
55              errExit("futex-FUTEX_WAKE");
56      }
57  }
```

IPC namespace, изоляция и просмотр через fs

Все System V объекты IPC изолируются при создании нового IPC namespace.
Объекты именуются уникальными idшниками, найти их через vfs затруднительно.

POSIX:

mq_* — изолируются через IPC namespace.
Можно примонтировать vfs, для мониторинга

```
# mkdir /dev/mqueue
# mount -t mqueue none /dev/mqueue
```

Именованные POSIX семафоры и shared memory через IPC namespace не изолируются.
Нужно изолировать через mount namespace (/dev/shm)