Interprocess Communication

Operating Systems

School of Data & Computer Science
Sun Yat-sen University

Lecture Notes: os_sysu@163.com
Instructor: Guoyang Cai
email: isscgy@mail.sysu.edu.cn

Contents

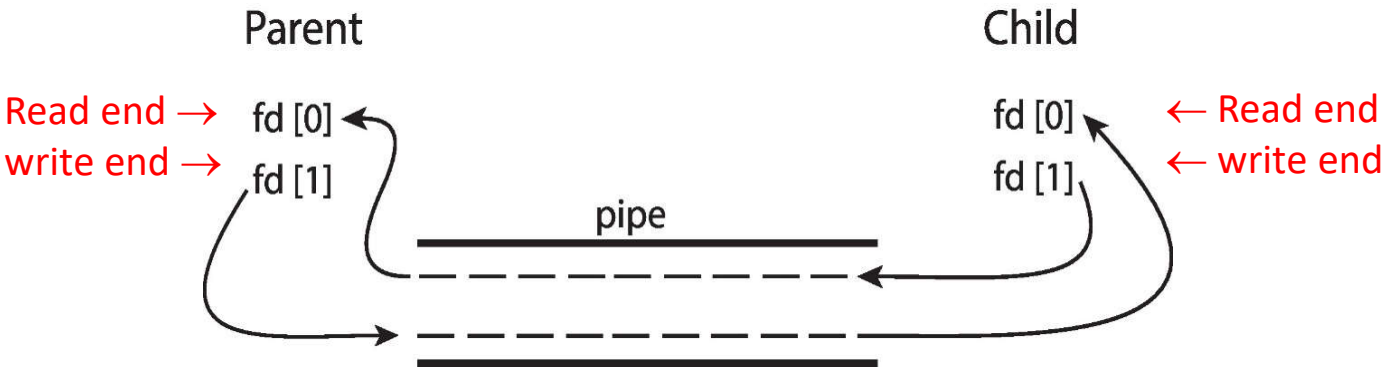
Pipes

- A pipe acts as a conduit allowing two processes to communicate.
  - a byte-stream without any structure
- Some issues in implementing a communication:
  - Is communication unidirectional or bidirectional?
  - In the case of two-way communication, is it half or full-duplex?
  - Must there exist a relationship (e.g., parent-child) between the communicating processes?
  - Can the pipes be used over a network, or must the processes reside on the same host?

Pipes

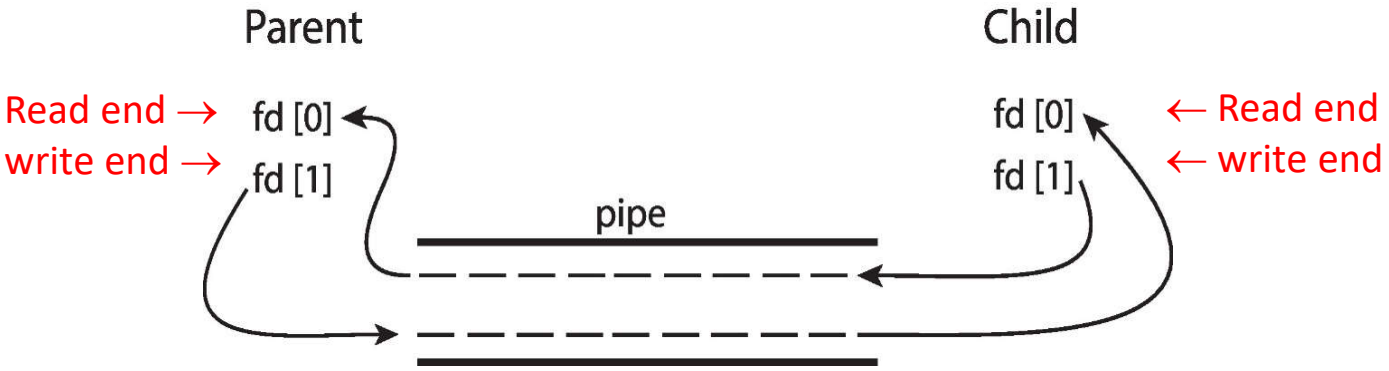
- Ordinary Pipes
  - Ordinary Pipes (无名管道) allow communication in standard producer-consumer style.
    - Producer writes to one end (the *write-end* of the pipe).
    - Consumer reads from the other end (the *read-end* of the pipe).
  - Ordinary pipes are therefore unidirectional. Two pipes must be used if two-way communication is required.
  - On UNIX systems, ordinary pipes are constructed using the function

```
pipe(int fd[])
```



Pipes

- Ordinary Pipes
  - An ordinary pipe is treated as a special type of file. The created pipe is accessed through the int fd[] file descriptors.
    - fd[0] the read end of the pipe
    - fd[1] the write end of the pipe
  - Thus, pipes can be accessed using ordinary read() and write() system calls.
  - An ordinary pipe cannot be accessed from outside the process that created it. Typically, a parent process creates a pipe and uses it to communicate with a child process created via fork(). The child inherits the pipe like an opened file from its parent process.



Pipes

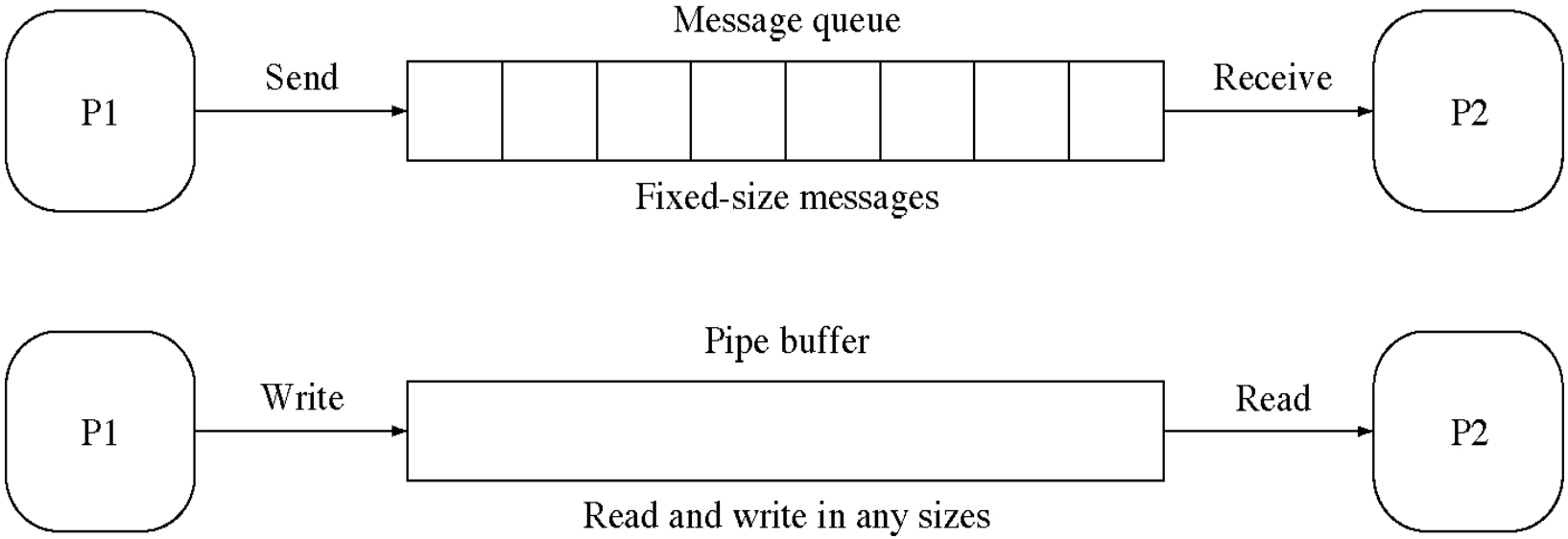
- Named Pipes
  - Named Pipes (命名管道) are more powerful than ordinary pipes.
    - Communication is bidirectional.
    - No parent-child relationship is necessary between the communicating processes.
  - Several processes can use an established named pipe for communication.
  - Named Pipes are provided on both UNIX-like and Windows systems.

Pipes

- Named Pipes in UNIX/Linux
  - Named Pipes are referred to as FIFOs, created with the mkfifo() system call.
    - Once created, they appear as typical files in the file system and manipulated with the ordinary open(), read(), write(), and close() system calls.
  - A FIFO will continue to exist until it is explicitly deleted from the file system.
  - FIFOs allow bidirectional communication and half-duplex transmission.
    - If data must travel in both directions, two FIFOs are typically used.
  - The communicating processes must reside on the same machine/host.
    - use *sockets* if inter-machine communication is required

Pipes

- Message-passing vs. Pipes



Linux: Pipes

- Linux User Level Limits
  - can be redefined in /etc/security/limits.conf

```
isscgy@ubuntu:/mnt/os-2020$ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 7645
max locked memory       (kbytes, -l) 65536
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 7645
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
isscgy@ubuntu:/mnt/os-2020$
```

- The pipe capacity is 512(byte)*8 = 4096 (byte) = 4KiB = 1 (page).

Linux: Pipes

Functions

Ordinary Pipes

```
int pipefd[2];
int pipe(int pipefd);

int fcntl(int pipefd[0|1], int cmd);
int fcntl(int pipefd[0|1], int cmd, long arg);

ssize_t write(int pipefd[1], void* buf, size_t count);

ssize_t read(int pipefd[0], void* buf, size_t count);

close(pipefd[0]);
close(pipefd[1]);
```

Linux: Pipes

Functions

Named Pipes

```
#define FIFO pathname /* pathname: "/tmp/my_fifo" */

unlink(FIFO); /*delete a name and possibly the file it refers to */

mkfifo(FIFO, 0666);
int fdw = open(FIFO, O_RDWR);

mkfifo(FIFO, 0444);
int fdr = open(FIFO, O_RDONLY);

ssize_t write(int fdw, void* buf, size_t count);
    /* ssizt_t = signed int, sizt_t = unsigned int */

ssize_t read(int fdr, void* buf, size_t count);

close(fdw);
close(fdr);
```

Linux: Pipes

- Pipe buffer size and pipe capacity
  - Algorithm 10-1: single pipe buffer (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#define ERR_EXIT(m) \
    do { \
        perror(m); \
        exit(EXIT_FAILURE); \
    } while (0)

int main(int argc, char *argv[])
{
    int pipefd[2];
    int bufsize;
    char *buffer;
    int flags, ret, lastwritten, count, totalwritten;

    if(pipe(pipefd) == -1) /* create an ordinary pipe */
        ERR_EXIT("pipe()");
    flags = fcntl(pipefd[1], F_GETFL);
    fcntl(pipefd[1], F_SETFL, flags | O_NONBLOCK); /* set write_end NONBLOCK */
    bufsize = atoi(argv[1]);
    printf("testing buffer size = %d\n", bufsize);
    buffer = (char *)malloc(bufsize*sizeof(char));
    if(buffer == NULL || bufsize == 0)
        ERR_EXIT("malloc()");
```

Linux: Pipes

- Pipe buffer size and pipe capacity
  - Algorithm 10-1: single pipe buffer(2)

```
    count = 0;
    while (1) {
        ret = write(pipefd[1], buffer, bufsize);
            /* bufsize is better to be 2^k */
        if(ret == -1) {
            perror("write()");
            break;
        }
        lastwritten = ret;
        count++;
    }
    totalwritten = (count-1)*bufsize + lastwritten;
    printf("single pipe buffer count =  %d, last written = %d bytes\n", count,
lastwritten);
    printf("total written = %d bytes = %d KiB\n", totalwritten,
totalwritten/1024); /* pipe buffer */

    return 0;
}
```

Linux: Pipes

- Pipe buffer size and pipe capacity
  - Algorithm 10-1: single pipe buffer(2)

```
isscgy@ubuntu:/mnt/os-2020$ ./a.out 1024
testing buffer size = 1024
write(): Resource temporarily unavailable
single pipe buffer count =  64, last written = 1024 bytes
total written = 65536 bytes = 64 KiB
isscgy@ubuntu:/mnt/os-2020$ ./a.out 4096
testing buffer size = 4096
write(): Resource temporarily unavailable
single pipe buffer count =  16, last written = 4096 bytes
total written = 65536 bytes = 64 KiB
isscgy@ubuntu:/mnt/os-2020$ ./a.out 4097
testing buffer size = 4097
write(): Resource temporarily unavailable
single pipe buffer count =  11, last written = 4096 bytes
total written = 45066 bytes = 44 KiB
isscgy@ubuntu:/mnt/os-2020$ ./a.out 32768
testing buffer size = 32768
write(): Resource temporarily unavailable
single pipe buffer count =  2, last written = 32768 bytes
total written = 65536 bytes = 64 KiB
isscgy@ubuntu:/mnt/os-2020$ ./a.out 32769
testing buffer size = 32769
write(): Resource temporarily unavailable
single pipe buffer count =  2, last written = 28673 bytes
total written = 61442 bytes = 60 KiB
isscgy@ubuntu:/mnt/os-2020$
```

, count,

Linux: Pipes

- Pipe buffer size and pipe capacity
  - Algorithm 10-1: single pipe buffer(2)

```
isscgy@ubuntu:/mnt/os-2020$ ./a.out 1024
testing buffer size = 1024
write(): Resource temporarily unavailable
single pipe buffer count =  64, last written = 1024 bytes
total written = 65536 bytes = 64 KiB
isscgy@ubuntu:/mnt/os-2020$ ./a.out 4096
testing buffer size = 4096
write(): Resource temporarily unavailable
single pipe buffer count =  16, last written = 4096 bytes
total written = 65536 bytes = 64 KiB
isscgy@ubuntu:/mnt/os-2020$ ./a.out 4097
testing buffer size = 4097
write(): Resource temporarily unavailable
single pipe buffer count =  11, last written = 4096 bytes
total written = 45066 bytes = 44 KiB
isscgy@ubuntu:/mnt/os-2020$ ./a.out 32768
testing buffer size = 3276
write(): Resource temporar
single pipe buffer count =
total written = 65536 byte
isscgy@ubuntu:/mnt/os-2020
testing buffer size = 3276
write(): Resource temporar
single pipe buffer count =
total written = 61442 byte
isscgy@ubuntu:/mnt/os-2020$
```

, count,

Differs from PIPE_BUF in that PIPE_SIZE is the length of the actual memory allocation, whereas PIPE_BUF makes atomicity guarantees.
check the header of include/linux/pipi_fs_i.h
#define PIPE_SIZE PAGE_SIZE /* 4 KiB */
#define PIPE_DEF_BUFFERS 16 /* 4*16 = 64 KiB */

Linux: Pipes

- Pipe buffer size and pipe capacity
  - Algorithm 10-2: Total pipes capacity.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#define PIPE_SIZE 64*1024 /* 64 KiB */
```

```
int main(void)
{
    char buf[PIPE_SIZE];
    int testfd[600][2];
    int i;
    long int ret;

    for (i = 0; i < 600; i++) {
        if(pipe(testfd[i]) == -1) {
            perror("pipe()");
            break;
        }
        fcntl(testfd[i][1], F_SETFL, O_NONBLOCK);
        ret = write(testfd[i][1], buf, PIPE_SIZE);
        if(ret == -1 || ret != PIPE_SIZE) {
            perror("write()");
            break;
        }
    }
    printf("\nsingle pipe buffer = 64 KiB, pipes created: %d\n", i);
    printf("total used size: %ld bytes = %ld KiB, or %.0f MiB\n", (long
int)i*PIPE_SIZE, (long int)i*PIPE_SIZE/1024, (double)i*PIPE_SIZE/1024/1024);
    return 0;
}
```

Linux: Pipes

- Pipe buffer size and pipe capacity
  - Algorithm 10-2: Total pipes capacity.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#define PIPE_SIZE 64*1024 /* 64 KiB */
```

```
int main(void)
{
    char buf[PIPE_SIZE];
    int testfd[600][2];
    int i;
```

```
isscgy@ubuntu:/mnt/os-2020$ gcc alg.10-2-pipecapacity.c
isscgy@ubuntu:/mnt/os-2020$ ./a.out
pipe(): Too many open files

single pipe buffer = 64 KiB, pipes created: 510
total used size: 33423360 bytes = 32640 KiB, or 32 MiB
isscgy@ubuntu:/mnt/os-2020$
```

```
                perror("write()");
                break;
            }
        }
        printf("\nsingle pipe buffer = 64 KiB, pipes created: %d\n", i);
        printf("total used size: %ld bytes = %ld KiB, or %.0f MiB\n", (long
int)i*PIPE_SIZE, (long int)i*PIPE_SIZE/1024, (double)i*PIPE_SIZE/1024/1024);
        return 0;
}
```

Linux: Pipes

Ordinary Pipes in UNIX/Linux

Algorithm 10-3: pipe-ord-1.c(1)

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>
#define READ_END 0
#define WRITE_END 1
```

```
/* a blocking read version */
int main(void)
{
    char write_msg[BUFSIZ]; /* BUFSIZ = 8192bytes, saved from stdin */
    char read_msg[BUFSIZ];
    int pipefd[2]; /* pipefd[0] for READ_END, pipefd[1] for WRITE_END */
    int flags;
    pid_t pid;

    if(pipe(pipefd) == -1) { /* create a pipe */
        perror("pipe()");
        exit(EXIT_FAILURE);
    }
    flags = fcntl(pipefd[WRITE_END], F_GETFL);
    fcntl(pipefd[WRITE_END], F_SETFL, flags | O_NONBLOCK); /* non-blocking write */
    flags = fcntl(pipefd[READ_END], F_GETFL);
    fcntl(pipefd[READ_END], F_SETFL, flags); /* blocking read */

    pid = fork(); /* fork a child process */
    if(pid < 0) {
        perror("fork()");
        exit(EXIT_FAILURE);
    }
```

Linux: Pipes

- Ordinary Pipes in UNIX/Linux
  - Algorithm 10-3: pipe-ord-1.c(2)

```
    if(pid > 0) { /* parent process */
        while (1) {
            printf("Enter some text: ");
            fgets(write_msg, BUFSIZ, stdin);
            write(pipefd[WRITE_END], write_msg, BUFSIZ);
            if (strncmp(write_msg, "end", 3) == 0)
                break;
        }
    }
    else { /* child process */
        while (1) {
            read(pipefd[READ_END], read_msg, BUFSIZ);
            printf("\n\t\t\t\t\tpipe read = %s", read_msg);
            if(strncmp(read_msg, "end", 3) == 0)
                break;
        }
    }

    wait(0);
    close(pipefd[WRITE_END]);
    close(pipefd[READ_END]);
    exit(EXIT_SUCCESS);
}
```

Linux: Pipes

- Ordinary Pipes in UNIX/Linux
  - Algorithm 10-3: pipe-ord-1.c(2)

```
isscgy@ubuntu:/mnt/os-2020$ gcc alg.10-3-pipe-ord-1.c
isscgy@ubuntu:/mnt/os-2020$ ./a.out
Enter some text: hello world

                                        pipe read = hello world
Enter some text: good morning

                                        pipe read = good morning
Enter some text: end

                                        pipe read = end
isscgy@ubuntu:/mnt/os-2020$
```

```
            }
        }

        wait(0);
        close(pipefd[WRITE_END]);
        close(pipefd[READ_END]);
        exit(EXIT_SUCCESS);
    }
```

Linux: Pipes

Ordinary Pipes in UNIX/Linux

Algorithm 10-3: pipe-ord-2.c(1)

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>
#define READ_END 0
#define WRITE_END 1
```

```
/* a non-blocking read version */
int main(void)
{
    char write_msg[BUFSIZ]; /* BUFSIZ = 8192bytes, saved from stdin */
    char read_msg[BUFSIZ];
    int pipefd[2]; /* pipefd[0] for READ_END, pipefd[1] for WRITE_END */
    int flags;
    pid_t pid;

    if(pipe(pipefd) == -1) { /* create a pipe */
        perror("pipe()");
        exit(EXIT_FAILURE);
    }
    flags = fcntl(pipefd[WRITE_END], F_GETFL);
    fcntl(pipefd[WRITE_END], F_SETFL, flags | O_NONBLOCK); /* non-blocking write */
    flags = fcntl(pipefd[READ_END], F_GETFL);
    fcntl(pipefd[READ_END], F_SETFL, flags | O_NONBLOCK); /* non-blocking read */

    pid = fork(); /* fork a child process */
    if(pid < 0) {
        perror("fork()");
        exit(EXIT_FAILURE);
    }
```

Linux: Pipes

Ordinary Pipes in UNIX/Linux

Algorithm 10-3: pipe-ord-2.c(2)

```
    if(pid > 0) { /* parent process */
        while (1) {
            printf("Enter some text: ");
            fgets(write_msg, BUFSIZ, stdin);
            write(pipefd[WRITE_END], write_msg, BUFSIZ);
            if(strncmp(write_msg, "end", 3) == 0)
                break;
        }
    }
    else { /* child process */
        while (1) {
            ret = read(pipefd[READ_END], read_msg, BUFSIZ);
                /* success: ret = 8192; failure: ret = -1 */
            if(ret > 0) {
                printf("\n%*spipe read = %s", 40, " ", read_msg);
                if(strncmp(read_msg, "end", 3) == 0)
                    break;
            }
            else //sleep(1); /* check every second */
        }
    }

    wait(0);
    close(pipefd[WRITE_END]); close(pipefd[READ_END]);
    exit(EXIT_SUCCESS);
}
```

Linux: Pipes

- Ordinary Pipes in UNIX/Linux
  - Algorithm 10-3: pipe-ord-2.c(2)

```
isscgy@ubuntu:/mnt/os-2020$ gcc alg.10-3-pipe-ord-2.c
isscgy@ubuntu:/mnt/os-2020$ ./a.out
Enter some text: hello world

                                        pipe read = hello world
Enter some text: good morning

                                        pipe read = good morning
Enter some text: end

                                        pipe read = end
isscgy@ubuntu:/mnt/os-2020$
```

```
                    break;
                }
                else //sleep(1); /* check every second */
            }
        }

        wait(0);
        close(pipefd[WRITE_END]); close(pipefd[READ_END]);
        exit(EXIT_SUCCESS);
}
```

Linux: Pipes

- Named Pipe between Parent & Child processes
  - Algorithm 10-4: pipe-nam.c (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <sys/stat.h>

#define TEXT_SIZE 1024
```

```
int main(int argc, char *argv[])
{
    char write_msg[TEXT_SIZE];
    char read_msg[TEXT_SIZE];
    char fifoname[80];
    int fdw, fdr;
    pid_t pid;

    if(argc < 2) {
        printf("Usage: ./a.out pathname\n");
        return EXIT_FAILURE;
    }
        /* pathname can not in current directory */
    strcpy(fifoname, argv[1]);
    if(access(fifoname, F_OK) == -1) {
        if(mkfifo(fifoname, 0666) != 0) { /* creat a named pipe */
            perror("mkfifo()");
            exit(EXIT_FAILURE);
        }
        else
            printf("new fifo %s created …\n", fifoname);
    }
```

Linux: Pipes

- Named Pipe between Parent & Child processes
  - Algorithm 10-4: pipe-nam.c (2)

```
pid = fork(); /* fork a child process */
if(pid < 0) {
    perror("fork()");
    exit(EXIT_FAILURE);
}
if(pid > 0) { /* parent process */
    fdw = open(fifoname, O_WRONLY); /* blocking write */
    if(fdw < 0)
        perror("pipe write open()");
    else {
        while (1) {
            printf("Enter some text: ");
            fgets(write_msg, TEXT_SIZE, stdin);
            write(fdw, write_msg, TEXT_SIZE);
            if(strncmp(write_msg, "end", 3) == 0)
                break;
            sleep(1);
        }
    }
}
```

Linux: Pipes

- Named Pipe between Parent & Child processes
  - Algorithm 10-4: pipe-nam.c (3)

```
    else { /* child process */
         fdr = open(fifoname, O_RDONLY);  /* blocking read */
         if(fdr < 0)
             perror("pipe read open()");
         else {
             while (1) {
                 read(fdr, read_msg, TEXT_SIZE);
                 printf("\n\t\t\t\t\tpipe read_end = %s", read_msg);
                 if(strncmp(read_msg, "end", 3) == 0)
                     break;
             }
         }
     }

     wait(0);

     close(fdw);
     close(fdr);
     unlink(fifoname);

     exit(EXIT_SUCCESS);
}
```

Linux: Pipes

- Named Pipe between Parent & Child processes
  - Algorithm 10-4: pipe-nam.c (3)

```
isscgy@ubuntu:/mnt/os-2020$ gcc alg.10-4-pipe-nam.c
isscgy@ubuntu:/mnt/os-2020$ ./a.out /tmp/mypipe
Enter some text: Hello World!

                                        pipe read_end = Hello World!
Enter some text: Goodmorning

                                        pipe read_end = Goodmorning
Enter some text: end

                                        pipe read_end = end
isscgy@ubuntu:/mnt/os-2020$ ls -l /tmp/mypipe
ls: cannot access '/tmp/mypipe': No such file or directory
isscgy@ubuntu:/mnt/os-2020$
```

```
            close(fdr);
            unlink(fifoname);

            exit(EXIT_SUCCESS);
        }
```

Linux: Pipes

- Named Pipe between Parent & Child processes
  - Algorithm 10-4: pipe-nam.c (3)

```
isscgy@ubuntu:/mnt/os-2020$ gcc alg.10-4-pipe-nam.c
isscgy@ubuntu:/mnt/os-2020$ ./a.out /tmp/mypipe
Enter some text: Hello World!

                                        pipe read_end = Hello World!
Enter some text: Goodmorning

                                        pipe read_end = Goodmorning
Enter some text: end

                                        pipe read_end = end
isscgy@ubuntu:/mnt/os-2020$ ls -l /tmp/mypipe
ls: cannot access '/tmp/mypipe': No such file or directory
isscgy@ubuntu:/mnt/os-2020$
```

```
            close(fdr);
            unlink(fifoname);

            exit(EXIT_SUCCESS);
        }
```

Linux: Pipes

Named Pipes between Two Arbitrary Processes

Algorithm 10-5: pipe-nam-write1.c (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>

#define TEXT_SIZE 1024
```

```
/* read & write version */
int main(int argc, char *argv[])
{
    char fifoname[80], write_msg[TEXT_SIZE];
    int fdw;

    if(argc < 2) {
        printf("Usage: ./a.out pathname\n");
        return EXIT_FAILURE;
    }
    strcpy(fifoname, argv[1]);
    if(access(fifoname, F_OK) == -1) {
        if(mkfifo(fifoname, 0666) != 0) {
            perror("mkfifo()");
            exit(EXIT_FAILURE);
        }
        else
            printf("new fifo %s created ...\n", fifoname);
    }

    fdw = open(fifoname, O_RDWR); /* non-blocking send & receive */
//    fdw = open(fifoname, O_WRONLY);
      /* blocking send, wating for the receiving end ready */
//    fdw = open(fifoname, O_WRONLY | O_NONBLOCK);
      /* non-blocking send, return error if the receiving end not ready */
```

Linux: Pipes

- Named Pipes between Two Arbitrary Processes
  - Algorithm 10-5: pipe-nam-write1.c (2)

```
    if(fdw < 0) {
        perror("pipe write open()");
        exit(EXIT_FAILURE);
    }
    else {
        while (1) {
            printf("\nEnter some text: ");
            fgets(write_msg, TEXT_SIZE, stdin);
            write(fdw, write_msg, TEXT_SIZE); /* non-blocking send */
            if (strncmp(write_msg, "end", 3) == 0)
                break;
            sleep(1);
          }
    }

    close(fdw);
    exit(EXIT_SUCCESS);
}
```

Linux: Pipes

Named Pipes between Two Arbitrary Processes

Algorithm 10-6: pipe-nam-write2.c (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>

#define TEXT_SIZE 1024
```

```
/* write only & non-blocking send version */
int main(int argc, char *argv[])
{
    char fifoname[80], write_msg[TEXT_SIZE];
    int fdw;

    if(argc < 2) {
        printf("Usage: ./a.out pathname\n");
        return EXIT_FAILURE;
    }
    strcpy(fifoname, argv[1]);
    if(access(fifoname, F_OK) == -1) {
        if(mkfifo(fifoname, 0666) != 0) {
            perror("mkfifo()");
            exit(EXIT_FAILURE);
        }
        else
            printf("new fifo %s named pipe created\n", fifoname);
    }
```

Linux: Pipes

Named Pipes between Two Arbitrary Processes

Algorithm 10-6: pipe-nam-write2.c (2)

```
    int count = 10;
    while (count) {
        fdw = open(fifoname, O_WRONLY | O_NONBLOCK);
        /* non-blocking send, return error if the receiving end not ready */
        if(fdw < 0) {
            printf("waiting for receiver ... %d\n", count);
            sleep(1);
            /* do something, and query again, or exit(EXIT_FAILURE) when time out  */
            count--;
        }
        else
            break;
    }

    while (count) {
        printf("\nEnter some text: ");
        fgets(write_msg, TEXT_SIZE, stdin);
        write(fdw, write_msg, TEXT_SIZE); /* non-blocking write */
        if (strncmp(write_msg, "end", 3) == 0)
            break;
        sleep(1);
    }

    close(fdw);
    exit(EXIT_SUCCESS);
}
```

Linux: Pipes

Named Pipes between Two Arbitrary Processes

Algorithm 10-7: pipe-nam-read.c (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>

#define TEXT_SIZE 1024
```

```
/* blocking read version */
int main(int argc, char *argv[])
{
    char fifoname[80], read_msg[TEXT_SIZE];
    int fdr;

    if(argc < 2) {
        printf("Usage: ./a.out pathname\n");
        return EXIT_FAILURE;
    }
    strcpy(fifoname, argv[1]);
    if(access(fifoname, F_OK) == -1) {
        if (mkfifo(fifoname, 0666) != 0) {
            perror("mkfifo()");
            exit(EXIT_FAILURE);
        }
        else
            printf("new fifo %s named pipe created\n", fifoname);
    }
```

Linux: Pipes

- Named Pipes between Two Arbitrary Processes
  - Algorithm 10-7: pipe-nam-read.c (2)

```
    fdr = open(fifoname, O_RDONLY);  /* blocking read */
    if (fdr < 0) {
        perror("pipe read open()");
        exit(EXIT_FAILURE);
    }
    else {
        while (1) {
            read(fdr, read_msg, TEXT_SIZE);
            printf("\npipe read_end = %s",read_msg);
            if (strncmp(read_msg, "end", 3) == 0)
                break;
        }
    }
    close(fdr);
    exit(EXIT_SUCCESS);
}
```

Linux: Pipes

- Named Pipes between Two Arbitrary Processes
  - Terminal running write1.o

```
isscgy@ubuntu:/mnt/os-2020$ ./alg.10-5-pipe-nam-write1.o /tmp/myfifo

Enter some text: hello world

Enter some text: end
isscgy@ubuntu:/mnt/os-2020$
```

  - Terminal running read.o

```
isscgy@ubuntu:/mnt/os-2020$ ./alg.10-7-pipe-nam-read.o /tmp/myfifo

pipe read_end = hello world

pipe read_end = end
isscgy@ubuntu:/mnt/os-2020$
```

Linux: Pipes

- Named Pipes between Two Arbitrary Processes
  - Terminal running write2.o

```
isscgy@ubuntu:/mnt/os-2020$ ./alg.10-6-pipe-nam-write2.o /tmp/myfifo
waiting for receiver ... 10
waiting for receiver ... 9
waiting for receiver ... 8
waiting for receiver ... 7
waiting for receiver ... 6

Enter some text: hello world

Enter some text: end
isscgy@ubuntu:/mnt/os-2020$
```

  - Terminal running read.o

```
isscgy@ubuntu:/mnt/os-2020$ ./alg.10-7-pipe-nam-read.o /tmp/myfifo

pipe read_end = hello world

pipe read_end = end
isscgy@ubuntu:/mnt/os-2020$
```

Linux: Pipes

Named Pipes between Two Arbitrary Processes

Algorithm 10-8: pipe-nam-rdwr1.c (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/wait.h>

#define TEXT_SIZE 1024
```

```
/* establishing two named pipes for dialog between
   two arbitrary processes */
/* starting from terminal-1 with ./a.out pathname 1
   starting from terminal-2 with ./a.out pathname 2 */
/* two ordinary pipes are used to build connection
   between child (read pro) and parent (write pro) */
int main(int argc, char *argv[])
{
    char fifoname_1[80], fifoname_2[80];
    char write_msg[TEXT_SIZE], read_msg[TEXT_SIZE];
    int fdr, fdw, ret;
    pid_t pid;
    int pipefd1[2], pipefd2[2], flags; char msg_str[2];

    if(argc < 3) {
        printf("Usage: ./a.out pathname 1|2\n");
        return EXIT_FAILURE;
    }
    if(pipe(pipefd1) == -1) {
        perror("pipe()");
        exit(EXIT_FAILURE);
    }
    flags = fcntl(pipefd1[1], F_GETFL);
    fcntl(pipefd1[1], F_SETFL, flags | O_NONBLOCK);
    flags = fcntl(pipefd1[0], F_GETFL);
    fcntl(pipefd1[0], F_SETFL, flags | O_NONBLOCK);
```

Linux: Pipes

Named Pipes between Two Arbitrary Processes

Algorithm 10-8: pipe-nam-rdwr1.c (2)

```
if(pipe(pipefd2) == -1) {
    perror("pipe()");
    exit(EXIT_FAILURE);
}
flags = fcntl(pipefd2[1], F_GETFL);
fcntl(pipefd2[1], F_SETFL, flags | O_NONBLOCK);
flags = fcntl(pipefd2[0], F_GETFL);
fcntl(pipefd2[0], F_SETFL, flags | O_NONBLOCK);

strcpy(fifoname_1, argv[1]); strcat(fifoname_1,"-1");
strcpy(fifoname_2, argv[1]); strcat(fifoname_2,"-2");
if(access(fifoname_1, F_OK) == -1) {
    if((mkfifo(fifoname_1, 0666)) != 0) {
        perror("mkfifo()");
        exit(EXIT_FAILURE);
    }
    else printf("new fifo %s created ...\n", fifoname_1);
}
if(access(fifoname_2, F_OK) == -1) {
    if((mkfifo(fifoname_2, 0666)) != 0) {
        perror("mkfifo()");
        exit(EXIT_FAILURE);
    }
    else printf("new fifo %s created ...\n", fifoname_2);
}
```

Linux: Pipes

Named Pipes between Two Arbitrary Processes

Algorithm 10-8: pipe-nam-rdwr1.c (3)

```
printf("\n==== pipe write end ====          ==== pipe read end ====\n");
pid = fork();
if(pid < 0) {
    perror("fork()");
    exit(EXIT_SUCCESS);
}
if(pid == 0) {
    if(argv[2][0] == '1')
        fdr = open(fifoname_1, O_RDONLY | O_NONBLOCK);
    else fdr = open(fifoname_2, O_RDONLY | O_NONBLOCK);
    if(fdr < 0)
        perror("fdr open()");
    else
        while (1) {
            ret = read(fdr, read_msg, TEXT_SIZE); /* non-blocking read */
            if(ret > 0) {
                printf("\n%*.s%s", 40, " ", read_msg);
                if(strncmp(read_msg, "end", 3) == 0)
                    break;
            }
            ret = read(pipefd2[0], msg_str, 2);
            if(ret > 0 && msg_str[0] == '1')
                break;
        }
    write(pipefd1[1], "1", 2);
    exit(0);
```

Linux: Pipes

Named Pipes between Two Arbitrary Processes

Algorithm 10-8: pipe-nam-rdwr1.c (4)

```
    } else {
        if(argv[2][0] == '1')
            fdw = open(fifoname_2, O_RDWR);
        else fdw = open(fifoname_1, O_RDWR);
        if(fdw < 0)
            perror("fdw open()");
        else
            while (1) {
                printf("\n");
                fgets(write_msg, TEXT_SIZE, stdin);
                ret = write(fdw, write_msg, TEXT_SIZE); /* non-blocking write */
                if(ret <= 0)
                    break;
                if(strncmp(write_msg, "end", 3) == 0)
                    break;
                ret = read(pipefd1[0], msg_str, 2);
                if(ret > 0 && msg_str[0] == '1')
                    break;
            }
        write(pipefd2[1], "1", 2);
    }
    wait(0);
    close(fdr); close(fdw);
    close(pipefd1[1]); close(pipefd1[0]); close(pipefd2[1]); close(pipefd2[0]);
    exit(EXIT_SUCCESS);
}
```

Linux: Pipes

- Named Pipes between Two Arbitrary Processes
  - Terminal-1 running rdwr1.o

```
isscgy@ubuntu:/mnt/os-2020$ ./alg.10-8-pipe-nam-rdwr1.o /tmp/myfifo 1

==== pipe write end ====          ==== pipe read end ====

hello world


                                  goodmorning

                                  end

isscgy@ubuntu:/mnt/os-2020$
```

  - Terminal-2 running rdwr1.o

```
isscgy@ubuntu:/mnt/os-2020$ ./alg.10-8-pipe-nam-rdwr1.o /tmp/myfifo 2

==== pipe write end ====          ==== pipe read end ====


                                  hello world

goodmorning

end
isscgy@ubuntu:/mnt/os-2020$
```

Linux: Pipes

- Named Pipes between Two Arbitrary Processes
  - Terminal-1 running rdwr1.o

```
isscgy@ubuntu:/mnt/os-2020$ ./alg.10-8-pipe-nam-rdwr1.o /tmp/myfifo 1
==== pipe write end ====          ==== pipe read end ====
hello world

                                  goodmorning

                                  end

isscgy@ubuntu:/mnt/os-2020$
```

  - Terminal-2 running rdwr1.o

```
isscgy@ubuntu:/mnt/os-2020$ ./alg.10-8-pipe-nam-rdwr1.o /tmp/myfifo 2
==== pipe write end ====          ==== pipe read end ====

                                  hello world
goodmorning

end
isscgy@ubuntu:/mnt/os-2020$
```

Linux: Pipes

Named Pipes between Two Arbitrary Processes

Algorithm 10-8: pipe-nam-rdwr2.c (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/wait.h>

#define TEXT_SIZE 1024
```

```
/* establishing two named pipes for dialog between
   two processes */
/* starting from terminal-1 with ./a.out pathname 1
   starting from terminal-2 with ./a.out pathname 2 */
/* kill(, SIGKILL) is used for pro termination */

int main(int argc, char *argv[])
{
    char fifoname_1[80], fifoname_2[80];
    char write_msg[TEXT_SIZE], read_msg[TEXT_SIZE];
    int fdr, fdw, ret;
    pid_t pid;
    int pipefd1[2], pipefd2[2], flags; char msg_str[2];

    if(argc < 3) {
        printf("Usage: ./a.out pathname 1|2\n");
        return EXIT_FAILURE;
    }
```

Linux: Pipes

- Named Pipes between Two Arbitrary Processes
  - Algorithm 10-8: pipe-nam-rdwr2.c (2)

```
strcpy(fifoname_1, argv[1]); strcat(fifoname_1,"-1");
strcpy(fifoname_2, argv[1]); strcat(fifoname_2,"-2");
if(access(fifoname_1, F_OK) == -1) {
    if((mkfifo(fifoname_1, 0666)) != 0) {
        perror("mkfifo()");
        exit(EXIT_FAILURE);
    }
    else printf("new fifo %s created ...\n", fifoname_1);
}
if(access(fifoname_2, F_OK) == -1) {
    if((mkfifo(fifoname_2, 0666)) != 0) {
        perror("mkfifo()");
        exit(EXIT_FAILURE);
    }
    else printf("new fifo %s created ...\n", fifoname_2);
}
```

Linux: Pipes

Named Pipes between Two Arbitrary Processes

Algorithm 10-8: pipe-nam-rdwr2.c (3)

```
printf("\n==== pipe write end ====          ==== pipe read end ====\n");
pid = fork();
if(pid < 0) {
    perror("fork()");
    exit(EXIT_SUCCESS);
}
if(pid == 0) {
    if(argv[2][0] == '1')
        fdr = open(fifoname_1, O_RDONLY);
    else fdr = open(fifoname_2, O_RDONLY);
    if(fdr < 0)
        perror("fdr open()");
    else
        while (1) {
            ret = read(fdr, read_msg, TEXT_SIZE); /* blocking read */
            if(ret <= 0) /* if write-end error */
                break;
            printf("\n%*.s%s", 40, " ", read_msg);
            if(strncmp(read_msg, "end", 3) == 0)
                break;
        }
        kill(getppid(), SIGKILL);
        exit(0);
```

Linux: Pipes

- Named Pipes between Two Arbitrary Processes
  - Algorithm 10-8: pipe-nam-rdwr2.c (4)

```
    } else {
        if(argv[2][0] == '1')
            fdw = open(fifoname_2, O_RDWR);
        else fdw = open(fifoname_1, O_RDWR);
        if(fdw < 0)
            perror("fdw open()");
        else
            while (1) {
                printf("\n");
                fgets(write_msg, TEXT_SIZE, stdin);
                ret = write(fdw, write_msg, TEXT_SIZE); /* non-blocking write */
                if(ret <= 0)
                    break;
                if(strncmp(write_msg, "end", 3) == 0)
                    break;
            }
            kill(pid, SIGKILL);
    }
    wait(0);
    close(fdr);
    close(fdw);
    exit(EXIT_SUCCESS);
}
```

Linux: Pipes

- Named Pipes between Two Arbitrary Processes
  - Terminal-1 running rdwr2.o

```
isscgy@ubuntu:/mnt/os-2020$ ./alg.10-8-pipe-nam-rdwr2.o /tmp/myfifo 2
==== pipe write end ====          ==== pipe read end ====
hello world

                                  goodmorning

                                  end
Killed
isscgy@ubuntu:/mnt/os-2020$
```

  - Terminal-2 running rdwr2.o

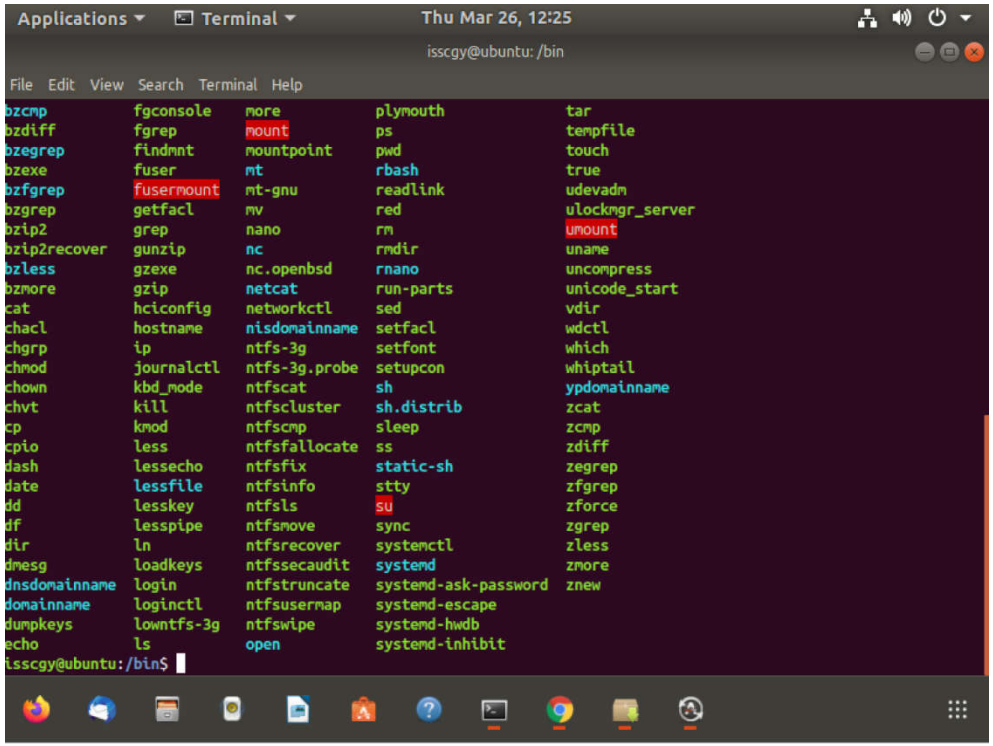
```
isscgy@ubuntu:/mnt/os-2020$ ./alg.10-8-pipe-nam-rdwr2.o /tmp/myfifo 1
==== pipe write end ====          ==== pipe read end ====

                                  hello world
goodmorning

end
isscgy@ubuntu:/mnt/os-2020$
```

Linux: Pipes

- Pipes in UNIX CLI
  - A pipe can be constructed on the UNIX command line using the | character. The complete command is

    ls | less

  - The commands ls and less are running as individual processes. The output of ls is delivered as the input to less. Result of ls:



Linux: Pipes

- Pipes in UNIX CLI
  - A pipe can be constructed on the UNIX command line using the | character. The complete command is

    ls | less

  - The ls serves as the producer, and its output is consumed by the less command. Result of ls | less: