

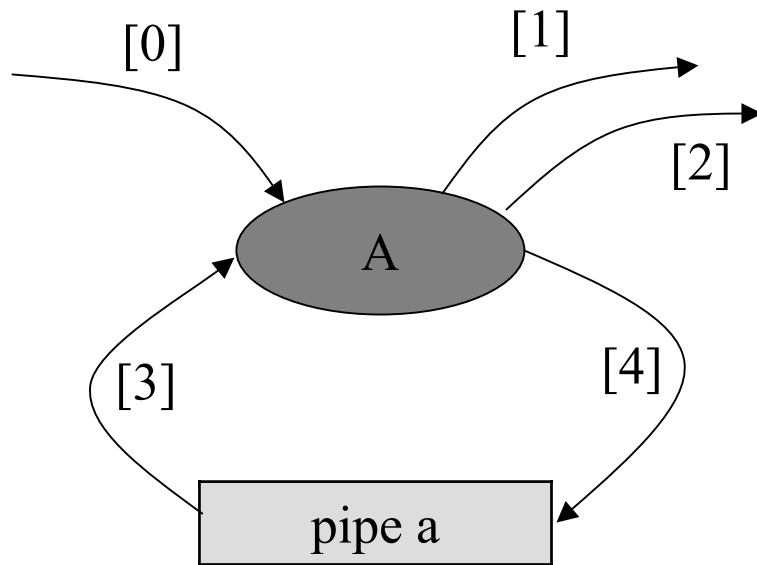
# Rings

This chapter demonstrates how processes can be formed into a ring using pipes for communication purposes.

# Forming a Ring of One Process

```
/* Example 4.1 */  
#include <unistd.h>  
int fd[2];  
  
pipe(fd);  
dup2(fd[0], STDIN_FILENO);  
dup2(fd[1], STDOUT_FILENO);  
close(fd[0]);  
close(fd[1]);
```

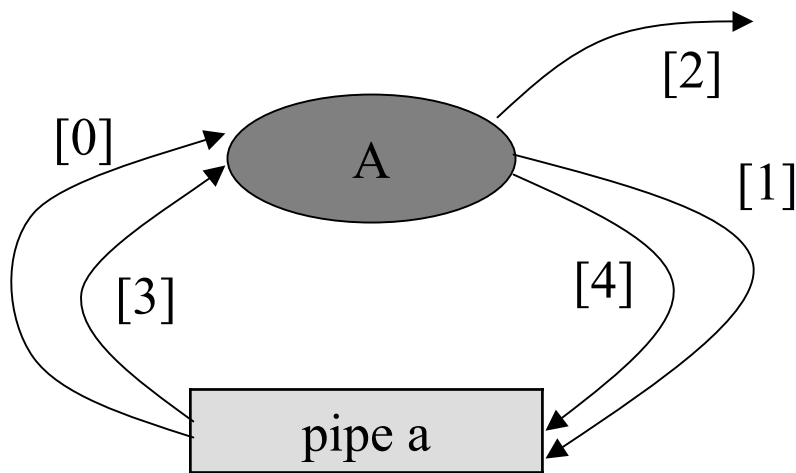
# One Process – Results of pipe



file descriptor table

[0]	standard input
[1]	standard output
[2]	standard error
[3]	pipe a read
[4]	pipe a write

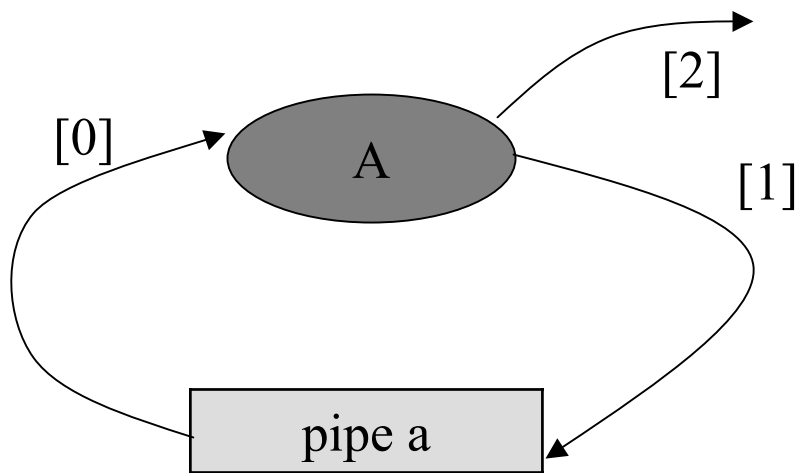
# One Process – Results after dup2s



file descriptor table

[0]	pipe a read
[1]	pipe a write
[2]	standard error
[3]	pipe a read
[4]	pipe a write

# One Process – Results after closes



file descriptor table

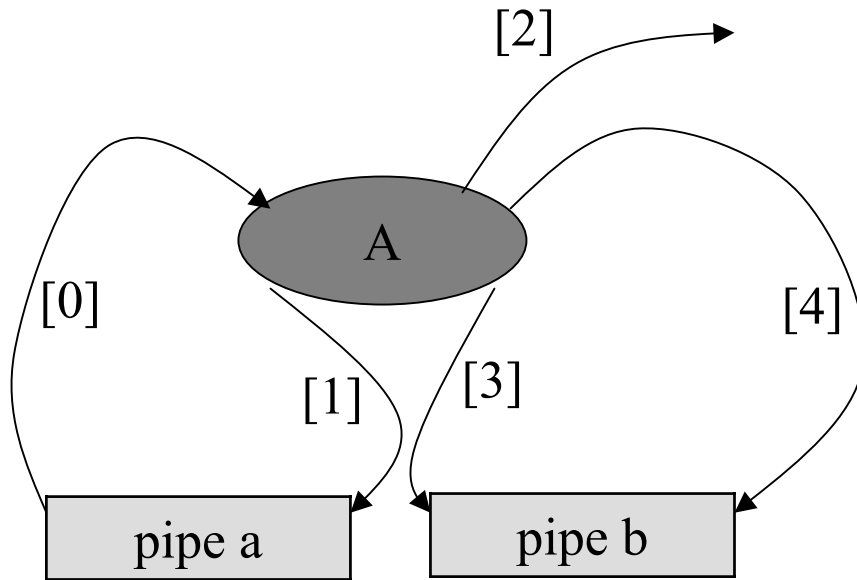
[0]	pipe a read
[1]	pipe a write
[2]	standard error

# Forming a Ring of Two Processes

```
/* Example 4.2 */
#include <unistd.h>
int fd[2];
pid_t haschild;
pipe(fd);
dup2(fd[0], STDIN_FILENO);
dup2(fd[1], STDOUT_FILENO);
close(fd[0]);
close(fd[1]);
pipe(fd);
if (haschild = fork())
    dup2(fd[1], STDOUT_FILENO); /* parent redirects std output
*/
else
    dup2(fd[0], STDIN_FILENO); /* child redirects std input */
close(fd[0]);
close(fd[1]);
```

\*\*\*Start with a ring of one process.\*\*\*

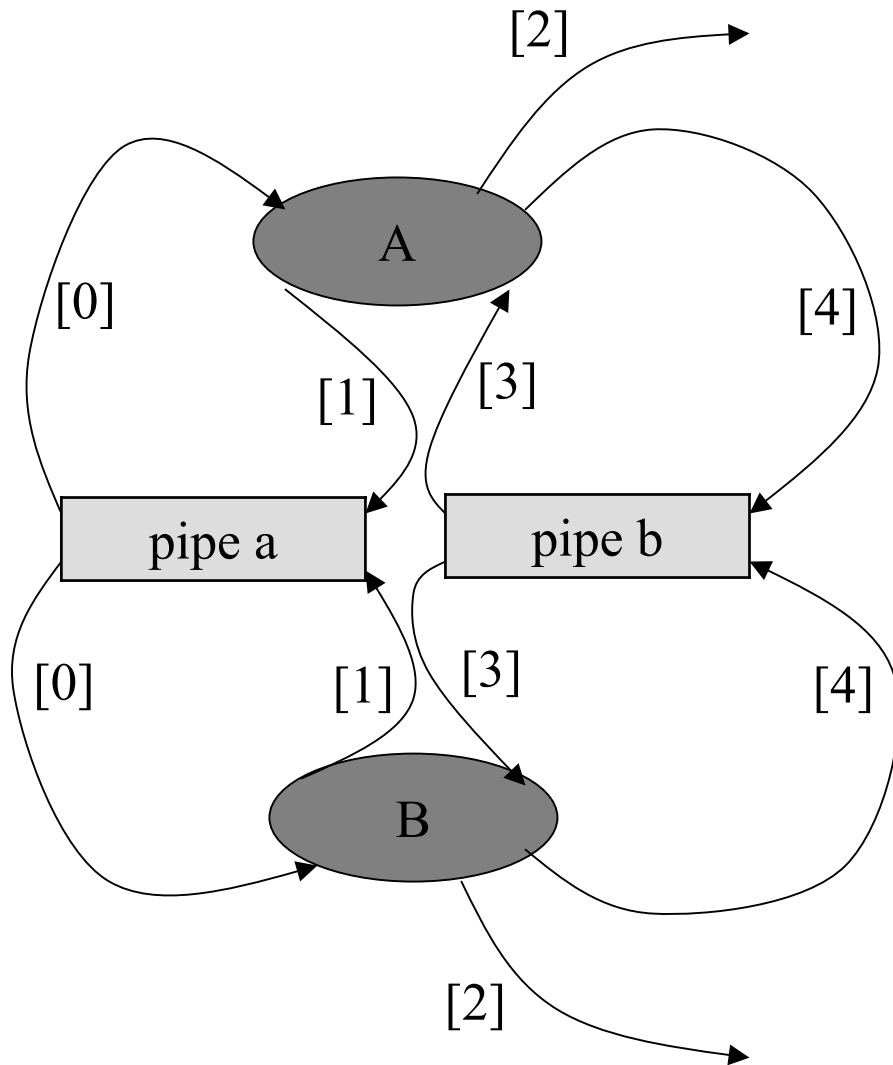
# Two Processes – After Second Pipe



Process A  
file descriptor table

[0]	pipe a read
[1]	pipe a write
[2]	standard error
[3]	pipe b read
[4]	pipe b write

# Two Processes – After fork



Process A

file descriptor table

[0] pipe a read

[1] pipe a write

[2] standard error

[3] pipe b read

[4] pipe b write

Process B

file descriptor table

[0] pipe a read

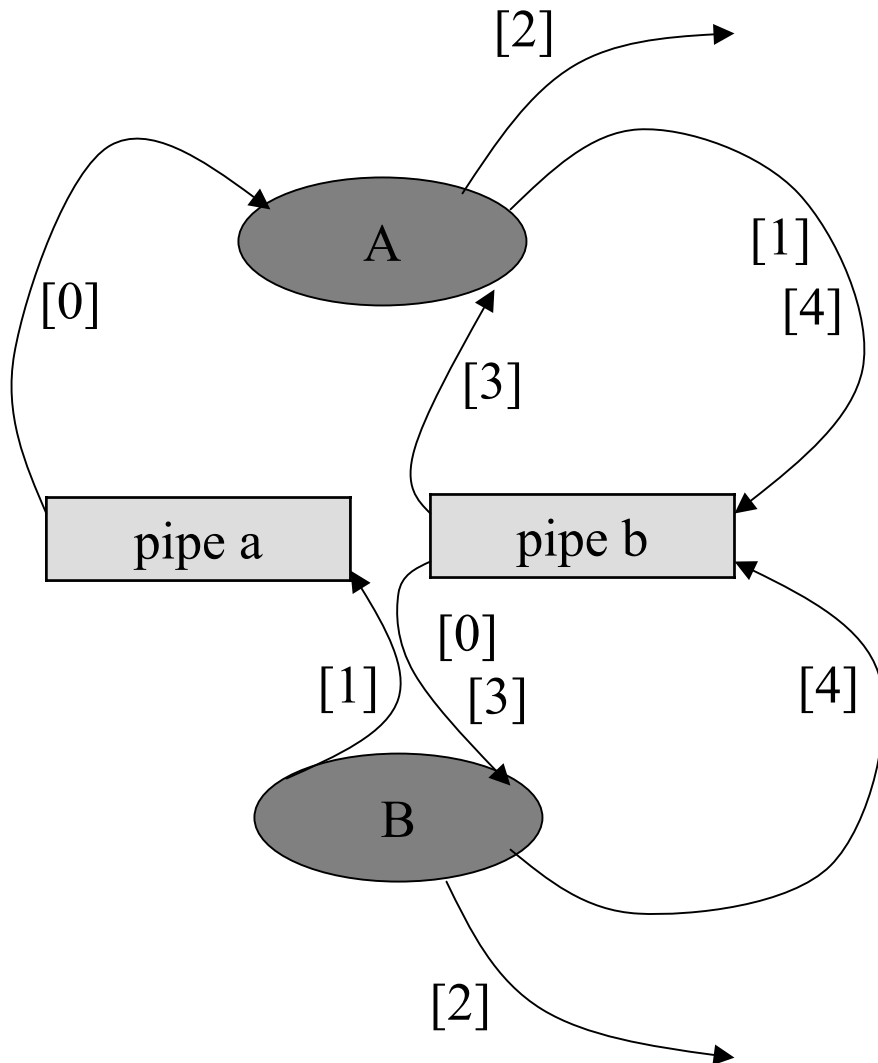
[1] pipe a write

[2] standard error

[3] pipe b read

[4] pipe write

# Two Processes – After Second dup2s



Process A

file descriptor table

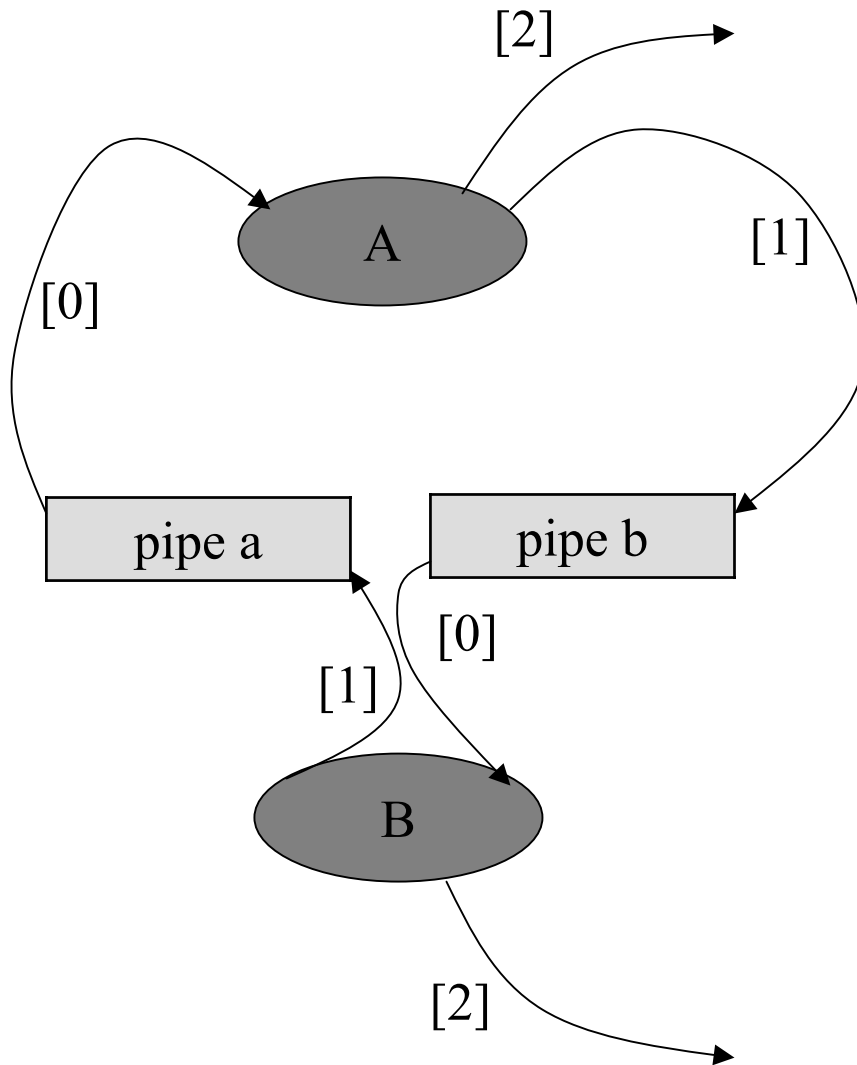
[0]	pipe a read
[1]	pipe b write
[2]	standard error
[3]	pipe b read
[4]	pipe b write

Process B

file descriptor table

[0]	pipe b read
[1]	pipe a write
[2]	standard error
[3]	pipe b read
[4]	pipe write

# Two Processes – After Second Closes



Process A

file descriptor table

[0] pipe a read

[1] pipe b write

[2] standard error

Process B

file descriptor table

[0] pipe b read

[1] pipe a write

[2] standard error

# Forming a Ring of N Processes

- Start with a ring of N-1 processes
- Have the last process in the N-1 ring perform a pipe
- Have the last process in the N-1 ring perform a fork
- Have the parent perform a dup2 of fd[1] with standard output
- Have the child perform a dup2 of fd[0] with standard input
- Close fd[0] and fd[1] for both processes

# Creating a Ring of N Processes – Part 1

```
/* Program 4.1 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
/* Sample C program for generating a unidirectional ring of processes. Invoke this program
with a command-line argument indicating the number of processes on the ring. Communication
is done via pipes that connect the standard output of a process to the standard input of
its successor on the ring. After the ring is created, each process identifies itself with
its process ID and the process ID of its parent. Each process then exits. */

void main(int argc, char *argv[ ])
{
    int i; /* number of this process (starting with 1) */
    int childpid; /* indicates process should spawn another */
    int nprocs; /* total number of processes in ring */
    int fd[2]; /* file descriptors returned by pipe */
    int error; /* return value from dup2 call */
    /* check command line for a valid number of processes to generate */
    if ( (argc != 2) || ((nprocs = atoi (argv[1])) <= 0) ) {
        fprintf (stderr, "Usage: %s nprocs\n", argv[0]);
        exit(1);
    }
    /* connect std input to std output via a pipe */
    if (pipe (fd) == -1) {
        perror("Could not create pipe");
        exit(1);
    }
    if ((dup2(fd[0], STDIN_FILENO) == -1) ||
        (dup2(fd[1], STDOUT_FILENO) == -1)) {
        perror("Could not dup pipes");
        exit(1);
    }
    if ((close(fd[0]) == -1) || (close(fd[1]) == -1)) {
        perror("Could not close extra descriptors");
        exit(1);
    }
}
```

# Creating a Ring of N Processes – Part 2

```
/* create the remaining processes with their connecting pipes */
for (i = 1; i < nprocs; i++) {
    if (pipe (fd) == -1) {
        fprintf(stderr, "Could not create pipe %d: %s\n",
            i, strerror(errno));
        exit(1);
    }
    if ((childpid = fork()) == -1) {
        fprintf(stderr, "Could not create child %d: %s\n",
            i, strerror(errno));
        exit(1);
    }
    if (childpid > 0) /* for parent process, reassign stdout */
        error = dup2(fd[1], STDOUT_FILENO);
    else
        error = dup2(fd[0], STDIN_FILENO);
    if (error == -1) {
        fprintf(stderr, "Could not dup pipes for iteration %d: %s\n",
            i, strerror(errno));
        exit(1);
    }
    if ((close(fd[0]) == -1) || (close(fd[1]) == -1)) {
        fprintf(stderr, "Could not close extra descriptors %d: %s\n",
            i, strerror(errno));
        exit(1);
    }
    if (childpid)
        break;
}

/* say hello to the world */
fprintf(stderr, "This is process %d with ID %d and parent id %d\n",
    i, (int) getpid(), (int) getppid());
exit (0);
} /* end of main program here */
```