Exercise 1: files, processes, pipes

Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

Exercise 1: Files, Processes, Pipes

Operating SystemsVU 2023W

Axel Brunnbauer, Florian Mihola, David Lung, Andreas Brandstätter, Peter Puschner

Technische Universität Wien
Computer Engineering
Cyber-Physical Systems

2023-10-19

Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

Content

File Handling

- File IO, Streams

Related Processes

- Create a process (fork)
- Load a new program into a process's memory (exec)
- Wait on a process's termination (wait)

IPC via Unnamed Pipes

- Communication between related processes

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

File Descriptor

- Non-negative integer

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

File Descriptor

- Non-negative integer
- Reference to an entry (= index) in the table of open files (file descriptor table) of the process

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

File Descriptor

- Non-negative integer
- Reference to an entry (= index) in the table of open files (file descriptor table) of the process
- Standard I/O file descriptors: already present at program start

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

File Descriptor

- Non-negative integer
- Reference to an entry (= index) in the table of open files (file descriptor table) of the process
- Standard I/O file descriptors: already present at program start

File desc.	Description	POSIX Name	stdio Stream
0	Standard input	STDIN_FILENO	stdin
1	Standard output	STDOUT_FILENO	stdout
2	Error output	STDERR_FILENO	stderr

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

File Descriptor

- Non-negative integer
- Reference to an entry (= index) in the table of open files (file descriptor table) of the process
- Standard I/O file descriptors: already present at program start

File desc.	Description	POSIX Name	stdio Stream
0	Standard input	STDIN_FILENO	stdin
1	Standard output	STDOUT_FILENO	stdout
2	Error output	STDERR_FILENO	stderr

- POSIX name is defined in `<unistd.h>`

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

int **open**(const char *pathname, int flags, mode_t mode)

Opens an existing file or creates a new file

Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

int **open**(const char *pathname, int flags, mode_t mode)

Opens an existing file or creates a new file

- pathname: path to the file

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

int **open**(const char *pathname, int flags, mode_t mode)

Opens an existing file or creates a new file

- pathname: path to the file
- flags: One of `O_RDONLY`, `O_WRONLY`, `O_RDWR`
  - Additional flags can be added (via bitwise OR):
    - `O_CREAT`: create the file if it does not exist
    - `O_EXCL`: fail if the file already exists

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

int **open**(const char *pathname, int flags, mode_t mode)

Opens an existing file or creates a new file

- pathname: path to the file
- flags: One of `O_RDONLY`, `O_WRONLY`, `O_RDWR`
  - Additional flags can be added (via bitwise OR):
    - `O_CREAT`: create the file if it does not exist
    - `O_EXCL`: fail if the file already exists
- mode: specifies the file mode bits to be applied when a new file is created

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

`int` **`open`**`(const char *pathname, int flags, mode_t mode)`

Opens an existing file or creates a new file

- pathname: path to the file
- flags: One of `O_RDONLY`, `O_WRONLY`, `O_RDWR`
  - Additional flags can be added (via bitwise OR):
    - `O_CREAT`: create the file if it does not exist
    - `O_EXCL`: fail if the file already exists
- mode: specifies the file mode bits to be applied when a new file is created
- Returns a file descriptor or -1 on error

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

int **open**(const char *pathname, int flags, mode_t mode)

Opens an existing file or creates a new file

- pathname: path to the file
- flags: One of `O_RDONLY`, `O_WRONLY`, `O_RDWR`
  - Additional flags can be added (via bitwise OR):
    - `O_CREAT`: create the file if it does not exist
    - `O_EXCL`: fail if the file already exists
- mode: specifies the file mode bits to be applied when a new file is created
- Returns a file descriptor or -1 on error

```c
int fd = open("~/data.txt", O_CREAT | O_EXCL | O_WRONLY);

if (fd < 0) {
    fprintf(stderr, "open failed: %s\n", strerror(errno));
    exit(EXIT_FAILURE);
}
```

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

ssize_t **read**(int fd, void *buf, size_t count)

Read up to count bytes from a file

Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

ssize_t **read**(int fd, void *buf, size_t count)

Read up to count bytes from a file

- fd: file descriptor

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

ssize_t **read**(int fd, void *buf, size_t count)

Read up to count bytes from a file

- fd: file descriptor
- buf: buffer to be filled with the read data

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

ssize_t **read**(int fd, void *buf, size_t count)

Read up to count bytes from a file

- fd: file descriptor
- buf: buffer to be filled with the read data
- count: size of buffer (max number of bytes to read)

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

`ssize_t` **`read`**`(int fd, void *buf, size_t count)`

Read up to count bytes from a file

- fd: file descriptor
- buf: buffer to be filled with the read data
- count: size of buffer (max number of bytes to read)
- Returns number of bytes effectively read or -1 on error

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

ssize_t **read**(int fd, void *buf, size_t count)

Read up to count bytes from a file

- fd: file descriptor
- buf: buffer to be filled with the read data
- count: size of buffer (max number of bytes to read)
- Returns number of bytes effectively read or -1 on error

```
char buffer[80];

for (int pos = 0; pos < sizeof(buffer); ) {
    int numread = read(fd, buffer+pos, sizeof(buffer)-pos);

    if (numread < 0) {
        // error
    } else
        pos += numread;
}
```

Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

ssize_t **write**(int fd, void *buf, size_t count)
: Write up to count bytes to a file

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

ssize_t **write**(int fd, void *buf, size_t count)

Write up to count bytes to a file

- fd: file descriptor

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

ssize_t **write**(int fd, void *buf, size_t count)

Write up to count bytes to a file

- fd: file descriptor
- buf: buffer with the data to be written

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

ssize_t **write**(int fd, void *buf, size_t count)

Write up to count bytes to a file

- fd: file descriptor
- buf: buffer with the data to be written
- count: size of buffer (max number of bytes to write)

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

ssize_t **write**(int fd, void *buf, size_t count)

Write up to count bytes to a file

- fd: file descriptor
- buf: buffer with the data to be written
- count: size of buffer (max number of bytes to write)
- Returns the number of bytes effectively written or -1

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

ssize_t write(int fd, void *buf, size_t count)

Write up to count bytes to a file

- fd: file descriptor
- buf: buffer with the data to be written
- count: size of buffer (max number of bytes to write)
- Returns the number of bytes effectively written or -1

```
char buffer[80] = "Data to be written";

for (int pos = 0; pos < sizeof(buffer); ) {
    int n = write(fd, buffer+pos, sizeof(buffer)-pos);

    if (n < 0) {
        // error
    } else
        pos += n;
}
```

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

int **close**(int fd)
: Close a file

Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

int **close**(int fd)

Close a file

- fd: file descriptor

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

int **close**(int fd)

Close a file

- fd: file descriptor
- Returns 0 on success and -1 on error

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

System calls for file access (see man pages chapter 2)

int **close**(int fd)

Close a file

- fd: file descriptor
- Returns 0 on success and -1 on error

```
if (close(fd) < 0) {
    fprintf(stderr, "close failed: %s\n", strerror(errno));
    exit(EXIT_FAILURE);
}
```

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

EINTR

- read() and write() can be interrupted by a signal

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

EINTR

- read() and write() can be interrupted by a signal
- In this case they return -1 and set errno to EINTR

Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

Unix File I/O
EINTR

- read() and write() can be interrupted by a signal
- In this case they return -1 and set errno to EINTR
- No bytes are read or written

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

EINTR

- read() and write() can be interrupted by a signal
- In this case they return -1 and set errno to EINTR
- No bytes are read or written
- If this happens, just retry

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

EINTR

- read() and write() can be interrupted by a signal
- In this case they return -1 and set errno to EINTR
- No bytes are read or written
- If this happens, just retry

read() without checking for EINTR:

```
char buffer[80];
int pos, numread;

for (pos = 0; pos < sizeof(buffer); ) {
    numread = read(fd, buffer + pos, sizeof(buffer) - pos);

    if (numread < 0) {
        // error

    } else
        pos += numread;
}
```

Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

Unix File I/O

EINTR

- read() and write() can be interrupted by a signal
- In this case they return -1 and set errno to EINTR
- No bytes are read or written
- If this happens, just retry

read() with checking for EINTR:

```c
char buffer[80];
int pos, numread;

for (pos = 0; pos < sizeof(buffer); ) {
    numread = read(fd, buffer + pos, sizeof(buffer) - pos);

    if (numread < 0) {
        if (errno != EINTR)
            // other error than EINTR
    } else
        pos += numread;
}
```

Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

Unix File I/O

EINTR

- read() and write() can be interrupted by a signal
- In this case they return -1 and set errno to EINTR
- No bytes are read or written
- If this happens, just retry

write() without checking for EINTR:

```
char buffer[80] = "Data to be written";
int pos, numwrit;

for (pos = 0; pos < sizeof(buffer); ) {
    numwrit = write(fd, buffer + pos, sizeof(buffer) - pos);

    if (numwrit < 0) {
        // error

    } else
        pos += numwrit;
}
```

Exercise 1: files, processes, pipes
Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unix File I/O

EINTR

- read() and write() can be interrupted by a signal
- In this case they return -1 and set errno to EINTR
- No bytes are read or written
- If this happens, just retry

write() with checking for EINTR:

```
char buffer[80] = "Data to be written";
int pos, numwrit;

for (pos = 0; pos < sizeof(buffer); ) {
    numwrit = write(fd, buffer + pos, sizeof(buffer) - pos);

    if (numwrit < 0) {
        if (errno != EINTR)
            // other error than EINTR
    } else
        pos += numwrit;
}
```

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Stream I/O in C

The functionality descends from a "portable I/O package" written by Mike Lesk at Bell Labs in the early 1970s. (Source: Wikipedia)

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Stream I/O in C

The functionality descends from a "portable I/O package" written by Mike Lesk at Bell Labs in the early 1970s. (Source: Wikipedia)

- Standard I/O library (portability)

```
#include <stdio.h>
```

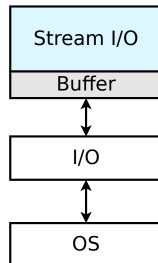
Stream I/O in C

The functionality descends from a "portable I/O package" written by Mike Lesk at Bell Labs in the early 1970s. (Source: Wikipedia)

- Standard I/O library (portability)

```
#include <stdio.h>
```

- Buffered layer on top of the Unix I/O



Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

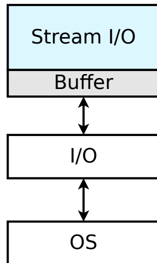
Stream I/O in C

The functionality descends from a “portable I/O package” written by Mike Lesk at Bell Labs in the early 1970s. (Source: Wikipedia)

- Standard I/O library (portability)

```
#include <stdio.h>
```

- Buffered layer on top of the Unix I/O
- Stream data type: `FILE`, includes i.a. file descriptor, pointer to buffer, current position, EOF and error flags



Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

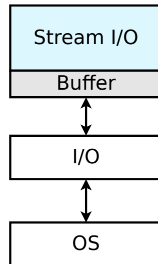
Stream I/O in C

The functionality descends from a "portable I/O package" written by Mike Lesk at Bell Labs in the early 1970s. (Source: Wikipedia)

- Standard I/O library (portability)

```
#include <stdio.h>
```

- Buffered layer on top of the Unix I/O
- Stream data type: `FILE`, includes i.a. file descriptor, pointer to buffer, current position, EOF and error flags
- Predefined streams `stdin`, `stdout`, `stderr`



Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

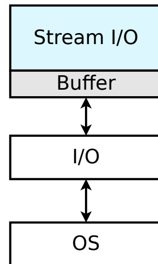
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Stream I/O in C

The functionality descends from a "portable I/O package" written by Mike Lesk at Bell Labs in the early 1970s. (Source: Wikipedia)

- Standard I/O library (portability)

```
#include <stdio.h>
```



- Buffered layer on top of the Unix I/O
- Stream data type: `FILE`, includes i.a. file descriptor, pointer to buffer, current position, EOF and error flags
- Predefined streams `stdin`, `stdout`, `stderr`
- Convention: functions start with "f": `fopen(3)`, `fdopen(3)`, `fwrite(3)`, `fprintf(3)`, ...

Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

fopen(3)

FILE ***fopen**(const char *path, const char *mode)

fopen(3)

FILE ***fopen**(const char *path, const char *mode)

- The file at `path` is opened, and associated with the returned stream

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

fopen(3)

FILE ***fopen**(const char *path, const char *mode)

- The file at `path` is opened, and associated with the returned stream
- Different I/O modes:
  - `"r"` : read-only

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

fopen(3)

FILE ***fopen**(const char *path, const char *mode)

- The file at `path` is opened, and associated with the returned stream
- Different I/O modes:
  - `"r"` : read-only
  - `"w"` : write-only (truncate to zero length first)

Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

fopen(3)

FILE ***fopen**(const char *path, const char *mode)

- The file at `path` is opened, and associated with the returned stream
- Different I/O modes:
  - `"r"` : read-only
  - `"w"` : write-only (truncate to zero length first)
  - `"a"` : append-only

fopen(3)

FILE ***fopen**(const char *path, const char *mode)

- The file at `path` is opened, and associated with the returned stream
- Different I/O modes:
  - `"r"` : read-only
  - `"w"` : write-only (truncate to zero length first)
  - `"a"` : append-only
  - `"r+"` / `"w+"` / `"a+"` : read and write (update mode)
    Read from beginning / truncate to zero length / writing at EOF

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

fopen(3)

FILE ***fopen**(const char *path, const char *mode)

- The file at `path` is opened, and associated with the returned stream
- Different I/O modes:
  - `"r"` : read-only
  - `"w"` : write-only (truncate to zero length first)
  - `"a"` : append-only
  - `"r+"` / `"w+"` / `"a+"` : read and write (update mode)
    Read from beginning / truncate to zero length / writing at EOF
- Returns NULL on failure (errno)

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

fopen(3)

FILE ***fopen**(const char *path, const char *mode)

- The file at `path` is opened, and associated with the returned stream
- Different I/O modes:
  - `"r"` : read-only
  - `"w"` : write-only (truncate to zero length first)
  - `"a"` : append-only
  - `"r+"` / `"w+"` / `"a+"` : read and write (update mode)
    Read from beginning / truncate to zero length / writing at EOF
- Returns NULL on failure (errno)

```
FILE *input = fopen("data.txt", "r");

if (input == NULL) {
    fprintf(stderr, "fopen failed: %s\n", strerror(errno));
    exit(EXIT_FAILURE);
}
```

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

fdopen(3)

FILE ***fdopen**(int fd, const char *mode)

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

fdopen(3)

FILE ***fdopen**(int fd, const char *mode)

- Associates a stream with a file descriptor

Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

fdopen(3)

FILE ***fdopen**(int fd, const char *mode)

- Associates a stream with a file descriptor
  - `fd`: file descriptor

Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

fdopen(3)

FILE ***fdopen**(int fd, const char *mode)

- Associates a stream with a file descriptor
  - `fd`: file descriptor
  - `mode`: I/O mode

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

fdopen(3)

FILE ***fdopen**(int fd, const char *mode)

- Associates a stream with a file descriptor
  - `fd`: file descriptor
  - `mode`: I/O mode
- Returns NULL on failure (errno)

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

fdopen(3)

FILE ***fdopen**(int fd, const char *mode)

- Associates a stream with a file descriptor
  - `fd`: file descriptor
  - `mode`: I/O mode
- Returns NULL on failure (errno)

```
int fd = open(...);
if (fd < 0) {
    // error
}

FILE *f = fdopen(fd, "r");
if (f == NULL) {
    // error
}
```

Reading and Writing

Function	Description
fread	Reads n elements, each s bytes long
fgets	Reads a line (up to `'\n'`)
fgetc	Reads a character
fwrite	Writes n elements, each s bytes long
fputs	Writes a C-string
fputc	Writes a character
fprintf	Formatted printing
fseek	Set the file position indicator
Since POSIX.1-2008:	
getline	Reads a line into a dynamically allocated buffer

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Stream Status

int **ferror**(FILE *stream)

- ferror tests the error indicator of the stream (0 = error flag not set).

Stream Status

int **ferror**(FILE *stream)

- ferror tests the error indicator of the stream (0 = error flag not set).

int **feof**(FILE *stream)

- feof tests the end-of-file indicator of the stream (e.g. functions fgets and fgetc set this flag upon reaching the end of file)

Stream Status

int **ferror**(FILE *stream)

- ferror tests the error indicator of the stream (0 = error flag not set).

int **feof**(FILE *stream)

- feof tests the end-of-file indicator of the stream (e.g. functions fgets and fgetc set this flag upon reaching the end of file)

int **clearerr**(FILE *stream)

- clearerr resets error and end-of-file indicators

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Stream Status

int **ferror**(FILE *stream)

- ferror tests the error indicator of the stream (0 = error flag not set).

int **feof**(FILE *stream)

- feof tests the end-of-file indicator of the stream (e.g. functions fgets and fgetc set this flag upon reaching the end of file)

int **clearerr**(FILE *stream)

- clearerr resets error and end-of-file indicators

int **fileno**(FILE *stream)

- fileno returns the file descriptor of a stream

Stream Status

int **ferror**(FILE *stream)

- ferror tests the error indicator of the stream (0 = error flag not set).

int **feof**(FILE *stream)

- feof tests the end-of-file indicator of the stream (e.g. functions fgets and fgetc set this flag upon reaching the end of file)

int **clearerr**(FILE *stream)

- clearerr resets error and end-of-file indicators

int **fileno**(FILE *stream)

- fileno returns the file descriptor of a stream

e.g. fileno(stdout) -> 1

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

fflush(3), fclose(3)

int **fflush**(FILE *stream)

- fflush enforces writing of buffered data

fflush(3), fclose(3)

int **fflush**(FILE *stream)

- fflush enforces writing of buffered data

int **fclose**(FILE *stream)

- fclose calls fflush and closes the stream and the associated file descriptor.

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

fflush(3), fclose(3)

int **fflush**(FILE *stream)

- fflush enforces writing of buffered data

int **fclose**(FILE *stream)

- fclose calls fflush and closes the stream and the associated file descriptor.

Return 0 on success, EOF on failure (errno)

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Stream I/O Examples

Read and write files

Read the content of an input file line by line and write it to an output file

```
char buffer[1024];
FILE *in, *out;

if ((in = fopen("input.txt", "r")) == NULL)
    // fopen failed

if ((out = fopen("output.txt", "w")) == NULL)
    // fopen failed

while (fgets(buffer, sizeof(buffer), in) != NULL) {
    if (fputs(buffer, out) == EOF)
        // fputs failed
}

if (ferror(in))
    // fgets failed

fclose(in);
fclose(out);
```

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Processes

Why should we create processes?

- Divide up a task
  - Simpler application design
  - Greater concurrency

Processes

Why should we create processes?

- Divide up a task
  - Simpler application design
  - Greater concurrency

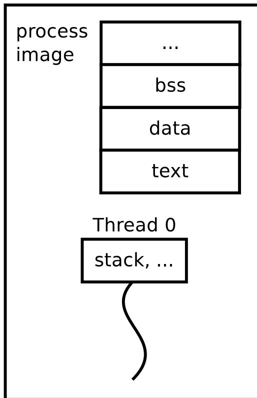
Example

A server listens to client requests. The server process starts a new process to handle each request and continues to listen for further connections.
The server can handle several client requests simultaneously.

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Process vs. Thread

fork(2) vs. pthreads(7)



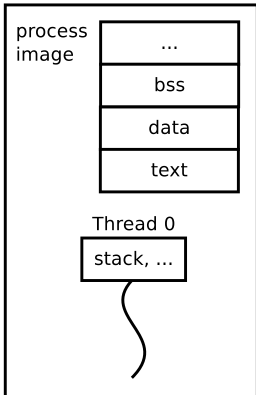
Files
- Unix File I/O
- Stream I/O

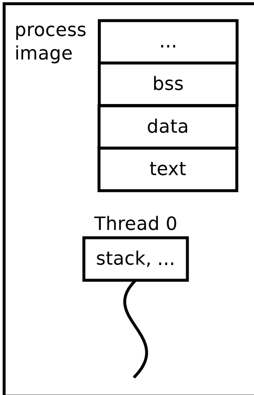
Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

Process vs. Thread

fork(2) vs. pthreads(7)

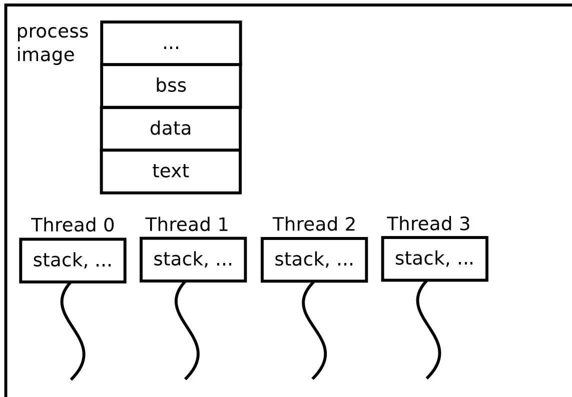




Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Process vs. Thread

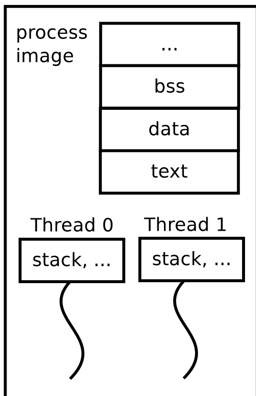
fork(2) vs. pthreads(7)

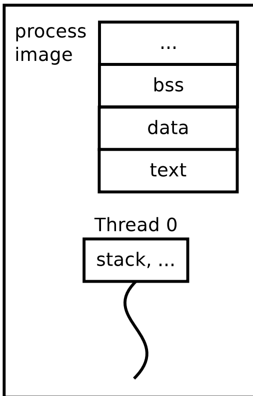


Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Process vs. Thread

fork(2) vs. pthreads(7)





Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

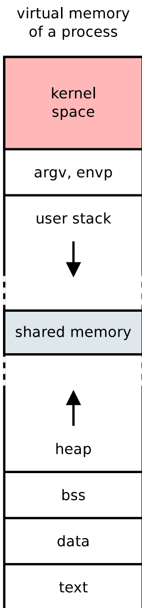
Process Hierarchy

- Every process has a parent process
- Exception: init process (`init`, `systemd`)
- Every process has a unique ID (`pid_t`)
- Show process hierarchy: `pstree(1)`

```
systemd-+-ModemManager---2*[{ModemManager}]
        |-NetworkManager-+-dhclient
        |                `-2*[{NetworkManager}]
        |-abrt-dbus---{abrt-dbus}
        |-2*[abrt-watch-log]
        |-abrtd
        |-acpid
        |-agetty
        |-alsactl
        |-atd
        |-auditd-+-audispd-+-sedispatch
        |        |         `-{audispd}
        |        `-{auditd}
        |-automount---7*[{automount}]
        |-avahi-daemon---avahi-daemon
        |-chronyd
        |-colord---2*[{colord}]
        |-crond
        |-cupsd
        |-dbus-daemon
        |-dnsmasq---dnsmasq
        |-firewalld---{firewalld}
        .
        .
```

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Memory Layout of a Process



Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

Properties of a Process in Linux

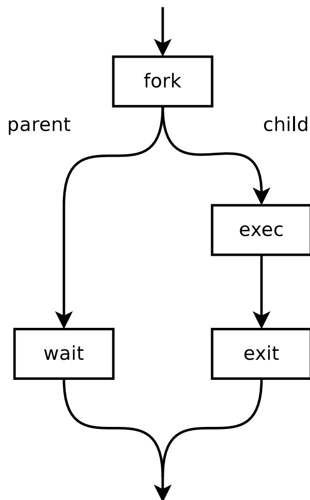
State	Running, waiting, ...
Scheduling	Priority, CPU time, ...
Identification	PID, owner, group, ...
Memory Management	Pointer to MMU information
Signals	Mask, pending
Process Relations	Parents, siblings

- Show process info: `cat /proc/<pid>/status`
- See `struct task_struct` in `sched.h`

Interface

fork / exec / exit / wait

- `fork(2)` – creates a process (copies the process image)
- `exec(3)` – loads a program (replaces the process image of a process with a new one)
- `exit(3)` – exits a process
- `wait(2)` – awaits the exit of child processes

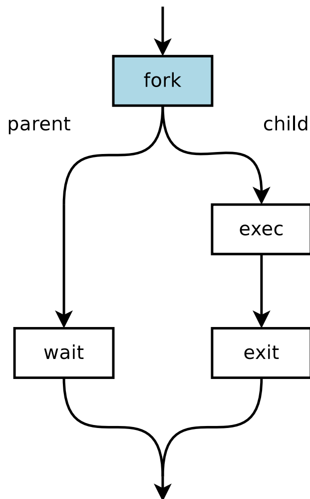


Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Process Creation

fork

- Creates a new process
- New process is an identical copy of the calling process – except PID, pending signals, ...
- Calling process is the parent of the created process, the child – processes are related
- Both processes run parallel and execute the same program (from the `fork` call on)



Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Process Creation

fork

- Create the process

```
#include <unistd.h>

pid_t fork(void);
```

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Process Creation

fork

- Create the process

```
#include <unistd.h>

pid_t fork(void);
```

- Distinguish between parent and child via return value of `fork`

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Process Creation

fork

- Create the process

```
#include <unistd.h>

pid_t fork(void);
```

- Distinguish between parent and child via return value of `fork`
  - -1 On error

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Process Creation

fork

- Create the process

```
#include <unistd.h>

pid_t fork(void);
```

- Distinguish between parent and child via return value of `fork`
  - -1 On error
  - 0 In the child process

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Process Creation

fork

- Create the process

```
#include <unistd.h>

pid_t fork(void);
```

- Distinguish between parent and child via return value of `fork`
  - -1 On error
  - 0 In the child process
  - >0 In the parent process

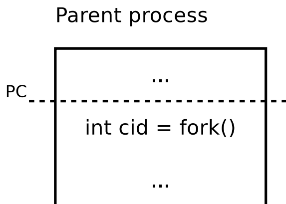
Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

Process Creation

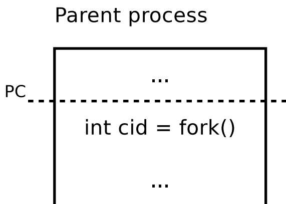
Before `fork()`



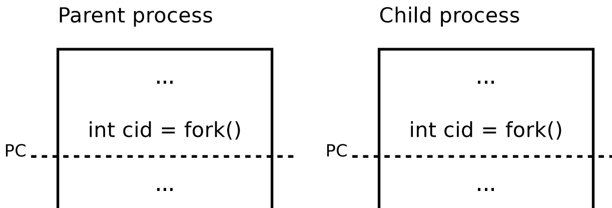
Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Process Creation

Before `fork()`



After `fork()`



Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Process Creation

Example

```
pid_t pid = fork();

switch (pid) {
  case -1:
    fprintf(stderr, "Cannot fork!\n");
    exit(EXIT_FAILURE);

  case 0:
    // child tasks
    ...
    break;
  default:
    // parent tasks
    ...
    break;
}
```

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Process Creation

Child

Child inherits from parent:

- Opened files (common access!)
- File buffers
- Signal handling
- Current values of variables

Process Creation

Child

Child inherits from parent:

- Opened files (common access!)
- File buffers
- Signal handling
- Current values of variables

But:

- Variables are local to process (no influence)
- Signal handling can be re-configured
- Communication (IPC) via pipes, sockets, shared memory, ...

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

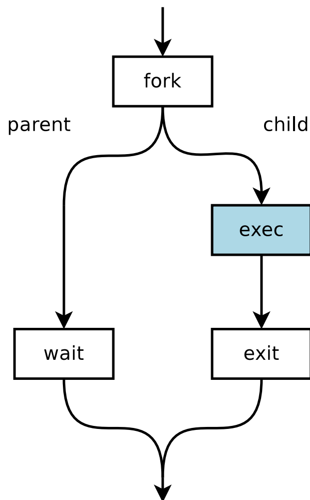
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Program Execution

exec

- Load a new program into a process's memory
- Executes another program
- In the same process (PID remains the same)



Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Program Execution

exec Family[1]

```
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);

int execle(const char *path, const char *arg, ...,
           char *const envp[]);

int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);

int fexecve(int fd, char *const argv[],
            char *const envp[]);
```

[1] Frontend of `execve(2)`

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Program Execution

exec Family

- `execl□` – variable number of arguments
- `execv□` – arguments via array
- `exec□p` – searching the environment variable $PATH for the program specified
- `exec□e` – environment[2] can be changed
- `fexecve` – accepts file descriptor (instead of path)

[2] FYI: environ(7)

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Program Execution

exec Family

- `execl□` – variable number of arguments
- `execv□` – arguments via array
- `exec□p` – searching the environment variable `$PATH` for the program specified
- `exec□e` – environment[2] can be changed
- `fexecve` – accepts file descriptor (instead of path)

Note Argument Passing!

- 1st argument is the program's name (`argv[0]`)!
- Last argument must be a `NULL` pointer!

[2] FYI: environ(7)

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Program Execution

Example: execv(), execvp()

```c
#include <unistd.h>


char *cmd[] = { "ls", "-l", (char *) 0 };

execv("/bin/ls", cmd);
// or:
// execvp("ls", cmd);

fprintf(stderr, "Cannot exec!\n");
exit(EXIT_FAILURE);
```

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Program Execution

Example: execl(), execlp()

```
#include <unistd.h>


execl("/bin/ls", "ls", "-l", NULL);
// or:
// execlp("ls", "ls", "-l", NULL);

fprintf(stderr, "Cannot exec!\n");
exit(EXIT_FAILURE);
```

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Program Execution

Example: execl(), execlp()

```
#include <unistd.h>


execl("/bin/ls", "ls", "-l", NULL);
// or:
// execlp("ls", "ls", "-l", NULL);

fprintf(stderr, "Cannot exec!\n");
exit(EXIT_FAILURE);
```

Attention - this is not working:

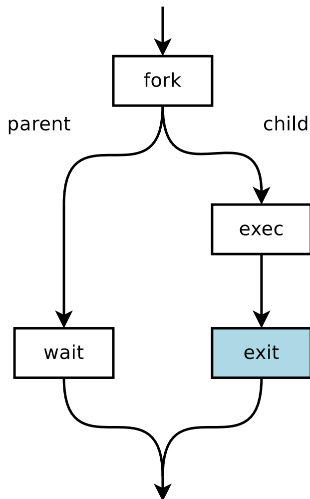
```
execl("/bin/ls", "ls -l", NULL);

int a = 1;
execl("myprog", "myprog", "-a", a, NULL);
    // e.g., use a char-buffer and snprintf(3)
```

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Process Termination

exit

- Terminates a process (normally)
- Termination status can be read by parents
- Actions performed by `exit()`
  - Flush and close stdio stream buffers
  - Close all open files
  - Delete temporary files (created by `tmpfile(3)`)
  - Call exit handlers (`atexit(3)`)



Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Process Termination

exit

- Terminate a process normally

```c
#include <stdlib.h>

void exit(int status);
```

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Process Termination

exit

- Terminate a process normally

```c
#include <stdlib.h>

void exit(int status);
```

- Status: 8 bit (0-255)
- By convention
  - `exit(EXIT_SUCCESS)` – process completed successfully
  - `exit(EXIT_FAILURE)` – error occurred

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Process Termination

exit

- Terminate a process normally

```
#include <stdlib.h>

void exit(int status);
```

- Status: 8 bit (0-255)
- By convention
  - `exit(EXIT_SUCCESS)` – process completed successfully
  - `exit(EXIT_FAILURE)` – error occurred
- More return values
  - BSD: sysexits.h
  - `http://tldp.org/LDP/abs/html/exitcodes.html`

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

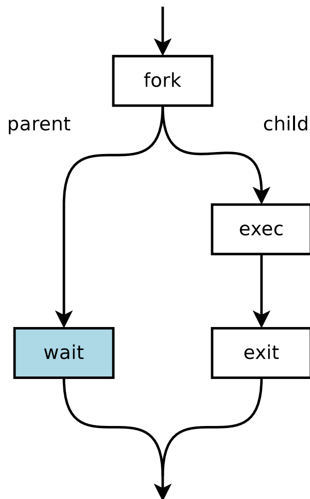
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Waiting on a Child Process

wait

- Wait until a child process terminates
- Returns the PID and status of the terminated child



Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Waiting on a Child Process

wait

- Wait for a child to terminate

```
#include <sys/wait.h>

pid_t wait(int *status);
```

[3] ≠ busy waiting

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Waiting on a Child Process

wait

- Wait for a child to terminate

```
#include <sys/wait.h>

pid_t wait(int *status);
```

- `wait()` blocks[3] until a child terminates or on error

[3] ≠ busy waiting

Exercise 1: files, processes, pipes
Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Waiting on a Child Process

wait

- Wait for a child to terminate

```c
#include <sys/wait.h>

pid_t wait(int *status);
```

- `wait()` blocks[3] until a child terminates or on error
- Return value
  - PID of the terminated child
  - -1 on error (→ `errno`, e.g., `ECHILD`)

[3] ≠ busy waiting

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Waiting on a Child Process

wait

- Wait for a child to terminate

```
#include <sys/wait.h>

pid_t wait(int *status);
```

- `wait()` blocks[3] until a child terminates or on error
- Return value
  - PID of the terminated child
  - -1 on error (→ `errno`, e.g., `ECHILD`)
- Status includes exit value and signal information
  - `WIFEXITED(status)`, `WEXITSTATUS(status)`
  - `WIFSIGNALED(status)`, `WTERMSIG(status)`
  - See `wait(2)`

[3] ≠ busy waiting

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Waiting on a Child Process

Zombies and Orphans

- UNIX: Terminated processes remain in the process table
- No more space in process table → no new process can be started!
- After `wait()` the child process is removed from the process table

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Waiting on a Child Process

Zombies and Orphans

- UNIX: Terminated processes remain in the process table
- No more space in process table → no new process can be started!
- After `wait()` the child process is removed from the process table

Zombie Child terminates, but parent didn't call `wait` yet
- State of the child is set to "zombie"
- Child remains in process table until parent calls `wait`

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Waiting on a Child Process

Zombies and Orphans

- UNIX: Terminated processes remain in the process table
- No more space in process table → no new process can be started!
- After `wait()` the child process is removed from the process table

Zombie Child terminates, but parent didn't call `wait` yet

- State of the child is set to "zombie"
- Child remains in process table until parent calls `wait`

Orphan Parent terminates before child

- Child becomes an orphan and is inherited to the init process
- When an orphan terminates, the init process removes the entry in the process table

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Waiting on a Child Process

Example

```
#include <sys/wait.h>

int status;
pid_t child_pid, pid;
...
while ((pid = wait(&status)) != child_pid)
{
  if (errno == EINTR) continue;
  if (pid == -1) {
    fprintf(stderr, "Cannot wait!\n");
    exit(EXIT_FAILURE);
  }
}

if (WEXITSTATUS(status) == EXIT_SUCCESS) {
  ...
```

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Waiting on a Child Process

waitpid

- Wait on a specific child process

```
#include <sys/wait.h>

pid_t waitpid(pid_t pid, int *status, int options);
```

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Waiting on a Child Process

waitpid

- Wait on a specific child process

```
#include <sys/wait.h>

pid_t waitpid(pid_t pid, int *status, int options);
```

- Examples

```
waitpid(cid, &status, 0);
      // waits on a child process with PID 'cid'

waitpid(-1, &status, 0);
      // equivalent to wait

waitpid(-1, &status, WNOHANG);
      // does not block
```

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Notification

on Termination of a Child

If parent should not block

- Synchronous
  - `waitpid(-1, &status, WNOHANG)`
  - Returns exit status when a child terminates
  - Repeating calls → polling

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Notification

on Termination of a Child

If parent should not block

- Synchronous
  - `waitpid(-1, &status, WNOHANG)`
  - Returns exit status when a child terminates
  - Repeating calls → polling
- Asynchronous
  - Signal `SIGCHLD` is sent to the parent process whenever one of its child processes terminates
  - Catch by installing a signal handler (`sigaction`)
  - Call `wait` in the signal handler

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Pitfalls

```c
int main(int argc, char **argv)
{
    fprintf(stdout, "Hello");

    (void) fork();
    return 0;
}
```

Output: `"HelloHello"`

Why?

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Pitfalls

```
int main(int argc, char **argv)
{
    fprintf(stdout, "Hello");
    fflush(stdout);
    (void) fork();
    return 0;
}
```

Output: `"Hello"`

→ for all opened streams

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Debugging

gdb

- Before `fork` is executed:
  `set follow-fork-mode [child|parent]`

Example

```
$ gdb -tui ./forktest
(gdb) break main
(gdb) set follow-fork-mode child
(gdb) run
(gdb) next
(gdb) :
(gdb) continue
(gdb) quit
```

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Inter-Process Communication

Recall

So far:

- Signals (e.g., to synchronise between parent and child)
  → see Development in C I

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Inter-Process Communication

Recall

So far:

- Signals (e.g., to synchronise between parent and child)
  → see Development in C I

New:

- Pipes

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Inter-Process Communication

Recall

So far:

- Signals (e.g., to synchronise between parent and child)
  → see Development in C I

New:

- Pipes

Next lectures:

- Shared Memory
- Sockets

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Pipes

Overview

(Unnamed) Pipe

$=$ unidirectional data channel

$=$ enables communication between related processes

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Pipes

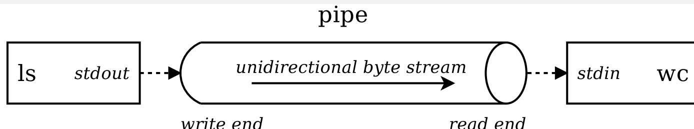
Overview

(Unnamed) Pipe

= unidirectional data channel
= enables communication between related processes

- Example

```
$ ls | wc -l
```



Exercise 1: files, processes, pipes
Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Pipes

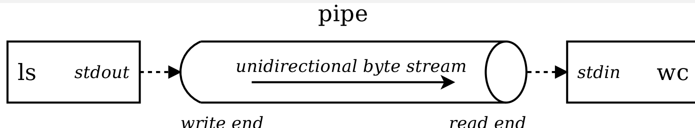
Overview

(Unnamed) Pipe

= unidirectional data channel
= enables communication between related processes

- Example

```
$ ls | wc -l
```



- Access to read and write end of the pipe via file descriptors
- Pipe is an unidirectional byte stream
- Buffered
- Implicit synchronisation

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Pipes

Create

- Create a pipe

```
#include <unistd.h>

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

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Pipes

Create

- Create a pipe

```
#include <unistd.h>

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

- File descriptors of read and write end are returned in specified integer array `pipefd`
  - `pipefd[0]` – read end
  - `pipefd[1]` – write end

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Pipes

Create

- Create a pipe

```
#include <unistd.h>

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

- File descriptors of read and write end are returned in specified integer array `pipefd`
  - `pipefd[0]` – read end
  - `pipefd[1]` – write end
- Close unused ends
- Use read/write end via stream-IO (`fdopen`, etc.)

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Pipes

Create

- Create a pipe

```c
#include <unistd.h>

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

- File descriptors of read and write end are returned in specified integer array `pipefd`
  - `pipefd[0]` – read end
  - `pipefd[1]` – write end
- Close unused ends
- Use read/write end via stream-IO (`fdopen`, etc.)
- A child process inherits the pipe → common access

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

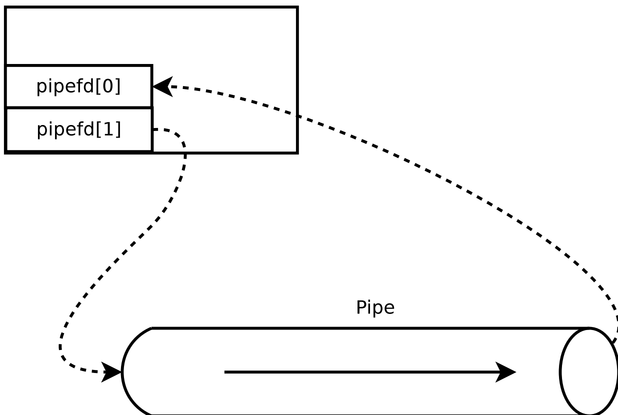
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unnamed Pipes

Illustration

`pipe`;



Files
- Unix File I/O
- Stream I/O

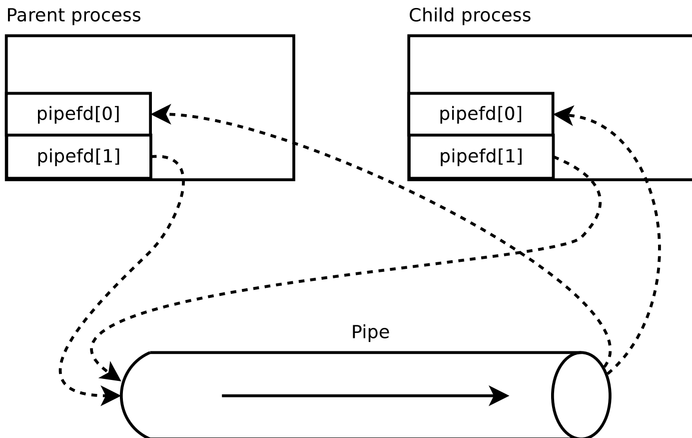
Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

Unnamed Pipes

Illustration

`pipe`; `fork`;



Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

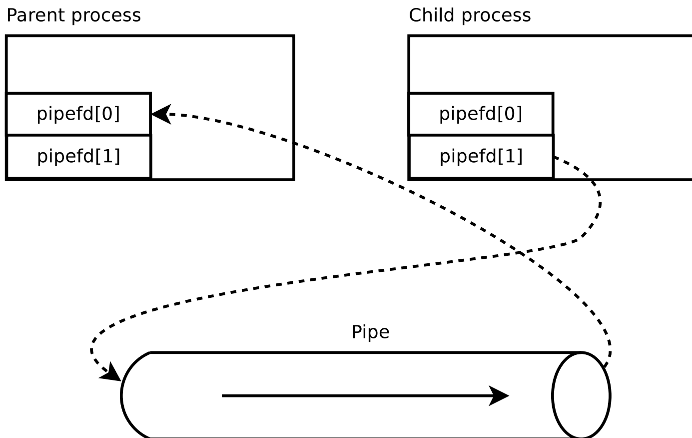
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unnamed Pipes

Illustration

`pipe`; `fork`; close unused ends;



Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unnamed Pipes

Implicit Synchronisation

- `read` blocks on empty pipe
- `write` blocks on full pipe

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unnamed Pipes

Implicit Synchronisation

- `read` blocks on empty pipe
- `write` blocks on full pipe

- `read` indicates end-of-file if all write ends are closed (return value 0)
- `write` creates signal `SIGPIPE` if all read ends are closed (if signal ignored/handled: `write` fails with errno `EPIPE`)

Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

Unnamed Pipes

Implicit Synchronisation

- `read` blocks on empty pipe
- `write` blocks on full pipe

- `read` indicates end-of-file if all write ends are closed (return value 0)
- `write` creates signal `SIGPIPE` if all read ends are closed (if signal ignored/handled: `write` fails with errno `EPIPE`)

Therefore…

… close unused ends, to get this behaviour (end-of-file and `SIGPIPE/EPIPE`).

Unnamed Pipes

Implicit Synchronisation

- `read` blocks on empty pipe
- `write` blocks on full pipe

- `read` indicates end-of-file if all write ends are closed (return value 0)
- `write` creates signal `SIGPIPE` if all read ends are closed (if signal ignored/handled: `write` fails with errno `EPIPE`)

Therefore...

... close unused ends, to get this behaviour (end-of-file and `SIGPIPE`/`EPIPE`).

Besides, the kernel removes pipes with all ends closed.

Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

Unnamed Pipes

What about named pipes?

- Unnamed pipes
  - `|`
  - `pipe(2)`

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Unnamed Pipes

What about named pipes?

- Unnamed pipes
  - `|`
  - `pipe(2)`
- Named pipes
  - `mkfifo(1)`, `mknod(2)`
  - Usage similar to files.
  - (Will not be dealt with any further throughout this course.)

Redirection of stdin/stdout

Why?

- Main application: pipes
- Example: shell redirection of `stdin` and `stdout`

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Redirection of stdin/stdout

Why?

- Main application: pipes
- Example: shell redirection of `stdin` and `stdout`

Scenario:

- A process may be forked or not
  → uses standard IO

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Redirection of stdin/stdout

Why?

- Main application: pipes
- Example: shell redirection of `stdin` and `stdout`

Scenario:

- A process may be forked or not
  → uses standard IO
- A parent process forks and executes another program

Redirection of stdin/stdout

Why?

- Main application: pipes
- Example: shell redirection of `stdin` and `stdout`

Scenario:

- A process may be forked or not
  → uses standard IO
- A parent process forks and executes another program
- Parent usually wants to use the child's output
  → redirect stdin (file descriptor 0, `STDIN_FILENO`) and/or stdout (file descriptor 1, `STDOUT_FILENO`) in new process

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Redirection of stdin/stdout

Approach

- Close file descriptors for standard I/O (`stdin`, `stdout`)
- Duplicate opened file descriptor (e.g., a pipe's end) to the closed one

```
#include <unistd.h>

int dup(int oldfd);
int dup2(int oldfd, int newfd);
```

- Close duplicated file descriptor

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Redirection of stdin/stdout

dup / dup2

- `dup(oldfd)` duplicates file descriptor `oldfd`
  - New file descriptor uses smallest unused ID = entry in file descriptor table
  - Duplicated file descriptor points to the same open file description (equal file offset, status flags) → see `open(2)`

Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Redirection of stdin/stdout

dup / dup2

- `dup(oldfd)` duplicates file descriptor `oldfd`
  - New file descriptor uses smallest unused ID
    = entry in file descriptor table
  - Duplicated file descriptor points to the same open file description (equal file offset, status flags) → see `open(2)`
- `dup2(oldfd, newfd)` duplicates `oldfd`
  - New file descriptor uses ID `newfd`
  - (Implicitly) closes the file descriptor `newfd` (if necessary)
  - `newfd` points to the same open file description like `oldfd`

Exercise 1: files, processes, pipes

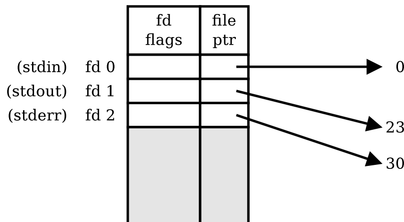
Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Redirection of stdin/stdout

Example: redirect stdout to opened file

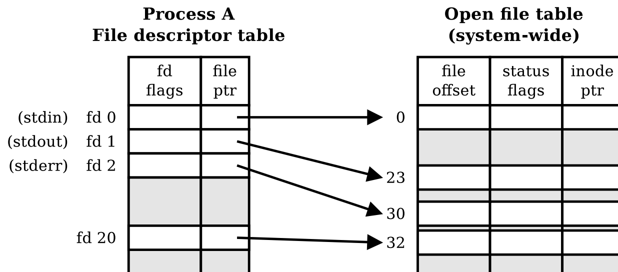


Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Redirection of stdin/stdout

Example: redirect stdout to opened file

open file;



Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

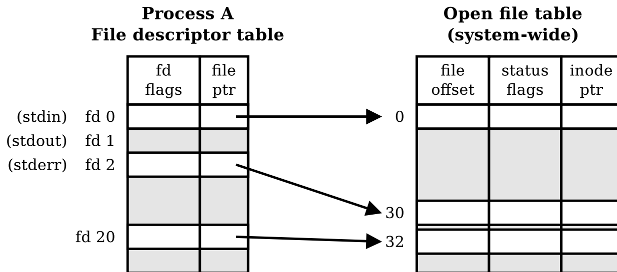
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Redirection of stdin/stdout

Example: redirect stdout to opened file

open file; close stdout;

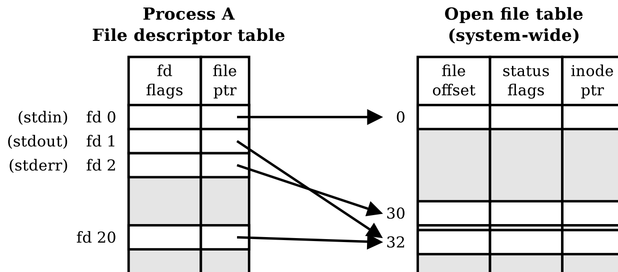


Exercise 1: files, processes, pipes
Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Redirection of stdin/stdout

Example: redirect stdout to opened file

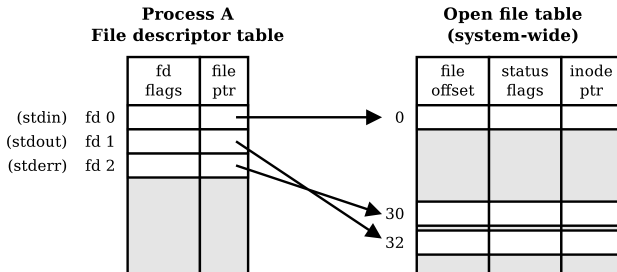
open file; close stdout; dup;



Redirection of stdin/stdout

Example: redirect stdout to opened file

open file; close stdout; dup; close file;



Exercise 1: files, processes, pipes

Files
Unix File I/O
Stream I/O

Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging

IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Redirection of stdin/stdout

Example: redirect stdout to log.txt

```
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>

int fd;

// TODO error handling!

fd = open("log.txt", O_WRONLY | O_CREAT);

dup2(fd,                // old descriptor
     STDOUT_FILENO);   // new descriptor

close(fd);

execlp("ls", "ls", NULL);
```

Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

Redirection of stdin/stdout

Example: redirect stdin to pipe

```
// TODO error handling!

int pipefd[2];
pipe(pipefd);             // create pipe

pid_t pid = fork();
switch(pid) {
  :
  case 0: // child counting lines from parent
    close(pipefd[1]);   // close unused write end

    dup2(pipefd[0],     // old descriptor - read end
         STDIN_FILENO); // new descriptor

    close(pipefd[0]);

    execlp("wc", "wc", "-l", NULL);
    // should not reach this line
  :
```

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Pitfalls

- Pipes are unidirectional
- Bidirectional: two pipes, but ...
  - Erroneous synchronisation (deadlock, e.g., both processes `read` from empty pipe)
- Synchronisation & Buffer
  - Use `fflush()`
  - Configure buffer (`setbuf(3)`, `setvbuf(3)`)

Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

Tips for the Exercise

- Try to parallel the functionality of your program (as much as possible)

Example

DO NOT: The parent first reads all input from a file to an array. It then sends the data within one burst to the child. The child processes the data and outputs the result.
INSTEAD DO: The parent reads line-by-line from a file. Each line is sent to the client immediately. Reading and processing of the lines happens in parallel.

Files
- Unix File I/O
- Stream I/O

Related Processes
- Process Properties
- Interface
- Process Creation
- Program Execution
- Process Termination
- Waiting on a Child Process
- Pitfalls
- Debugging

IPC
- Pipes
- Redirection of stdin/stdout
- Pitfalls

Tips for the Exercise

- Communicate over pipes (do not exploit inherited memory areas)

Example

DO NOT: The parent reads a file and saves its content into an array and forks a child. The child processes the data from the array.
INSTEAD DO: The parent communicates the data from the file over a pipe.

- However, you may pass options/flags/settings to the child (process). For example, use inherited variable `argv` to set arguments when using `exec`.

Exercise 1

- 1A: Implement a simple Unix tool
  - Become acquainted with the C language
  - Argument handling
  - Learn to use Makefiles

Files
Unix File I/O
Stream I/O
Related Processes
Process Properties
Interface
Process Creation
Program Execution
Process Termination
Waiting on a Child Process
Pitfalls
Debugging
IPC
Pipes
Redirection of stdin/stdout
Pitfalls

Exercise 1

- 1A: Implement a simple Unix tool
  - Become acquainted with the C language
  - Argument handling
  - Learn to use Makefiles
- 1B: Multiple communicating processes
  - fork/exec/wait
    - Start further programs
  - Unnamed Pipes
    - Communication between related processes
    - Redirection of stdin/stdout

Material

- Michael Kerrisk: A Linux and UNIX System Programming Handbook, No Starch Press, 2010.
- man pages: fork(2), exec(3), execve(2), exit(3), wait(3), pipe(2), dup(2)
- gdb - Debugging Forks:
  `https://sourceware.org/gdb/onlinedocs/gdb/Forks.html`