Project2 Supplementary

NP TAs

Outline

- Specs Reminder
  - Difference between server2 and server3
  - User pipe for server2 and server3
- Implementation
  - Handle Function Failures !!
    - Handle signal example

Specs Reminder

3 Servers

- np_simple (Single user)
  - Project1
  - **Concurrent connection-oriented**
- np_single_proc (Multiple users)
  - Project1 + **User pipe** + **4 Functions** + Broadcast message
  - **Single-process concurrent**
- np_multi_proc (Multiple users)
  - Project1 + **User pipe** + **4 Functions** + Broadcast message
  - **Concurrent connection-oriented** + **FIFO** + Shared memory

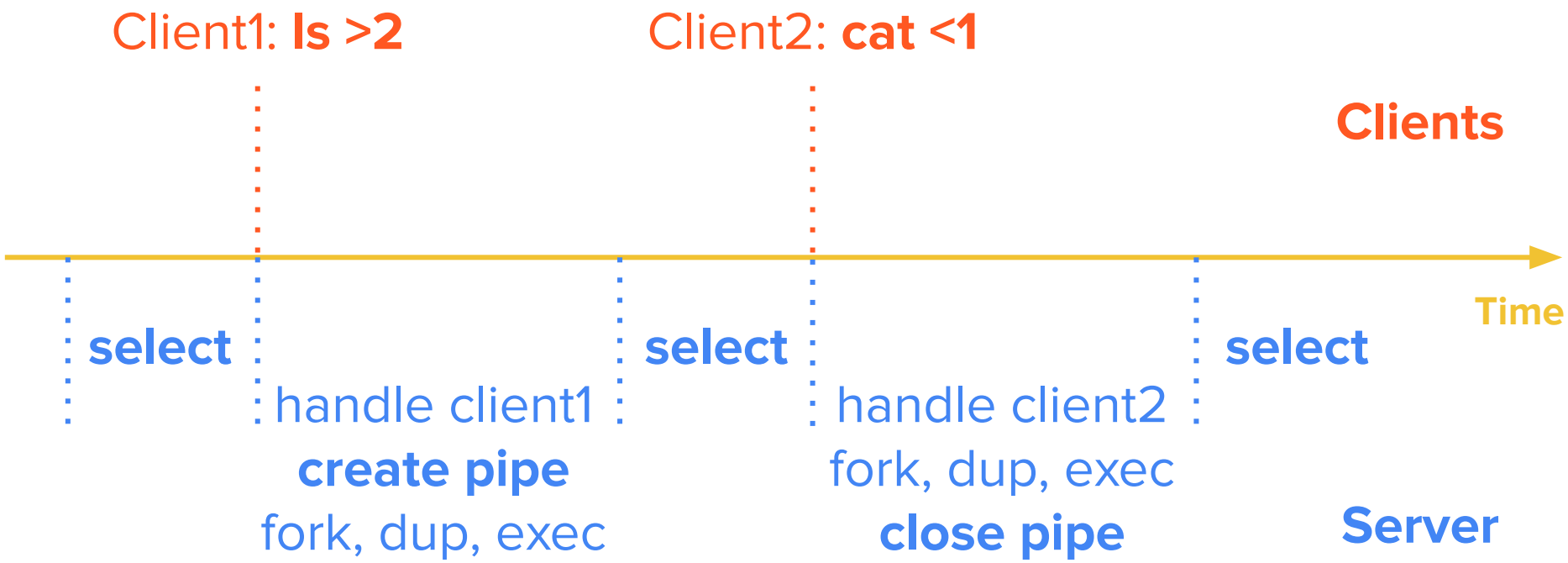
Difference between Server2 and Server3

- Server2 (np_single_proc)
  - **Single-process concurrent**
  - Use **pipe** to implement user pipe
  - Use socket to send messages directly.
- Server3 (np_multi_proc)
  - **Concurrent connection-oriented**
  - Use **FIFO** to implement user pipe
  - Use **shared memory** to save **clients infos** and **messages**.

Server2 (np_single_proc)

- **Single-process concurrent** (use **select**)
- Use **pipe** to implement user pipe
  - **DO NOT** use FIFO or temporary files
- Use socket to send messages directly
- Maintain environment variables for every user

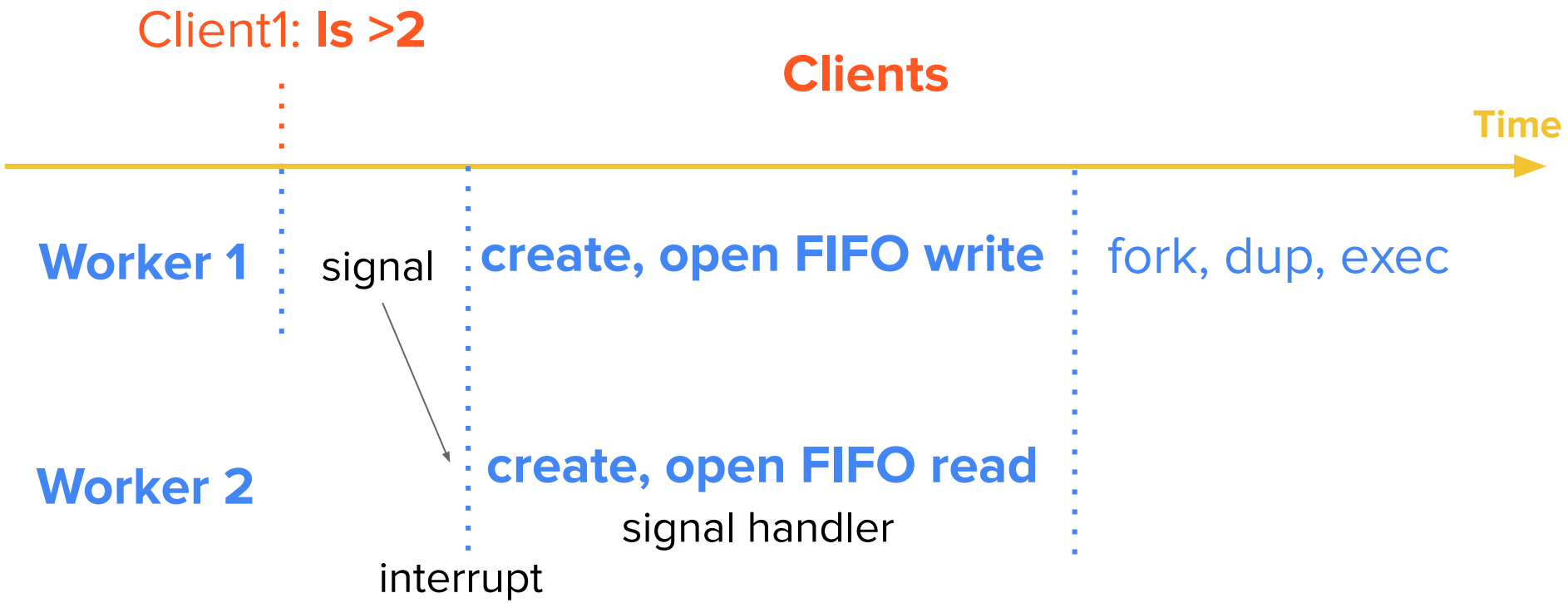
Server2 (np_single_proc) - User Pipe



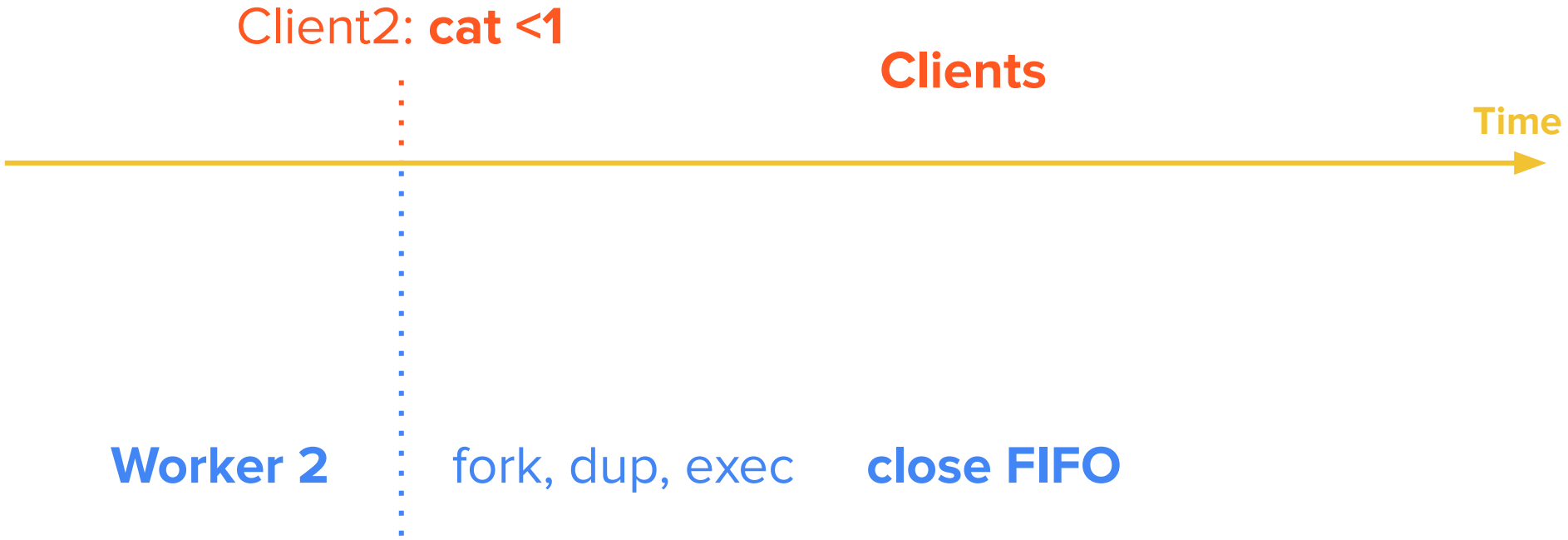
Server3 (np_multi_proc)

- **Concurrent connection-oriented**
- Use **FIFO** to implement user pipe
- Use **shared memory** to save **clients infos** and **messages**
- Handle signal
- Server3 will be terminated by **SIGINT** (Ctrl-C)
  - Receive SIGINT ➔ Clean up shared memory ➔ exit

Server3 (np_multi_proc) - User Pipe send



Server3 (np_multi_proc) - User Pipe recv



Implementation

Handle Function Failures !!

- **Fork** may failed
- **Create pipe** may failed
- **Select** may failed
- **Read** may failed
- See **man** for more details
  - 去除神秘力量

Select May Failed

```
if (select(maxfd+1, &read_set, NULL, NULL, NULL) < 0) {
    // may be interrupted by signal or other errors
    // handle error
}
for (fd = 0; fd < maxfd; ++fd) {
    if (FD_ISSET(fd, &read_set)) {
        //handle fd
    }
}
```

Read May Failed

```
if (read(cli_fd, buf, BUF_SIZE) < 0) {

    // may be interrupted by signal or other errors
    // handle error
}
```

Don't Send Additional '\0' Through Socket

- char str[] = "Hello"
  - write(fd, buf, **sizeof**(str)) sizeof(str) is 6
  - write(fd, buf, **strlen**(str)) OK
- std::string str = "Hello";
  - write(fd, str.c_str(), str.length()) OK

Handle Signal Example

```
void signal_handle(int signo) {
  if (SIGINT == signo) {
    // handle signal
  }
}

struct sigaction act;
act.sa_handler = signal_handler;
sigaction(SIGINT, &act, 0);
```

Linux IPC

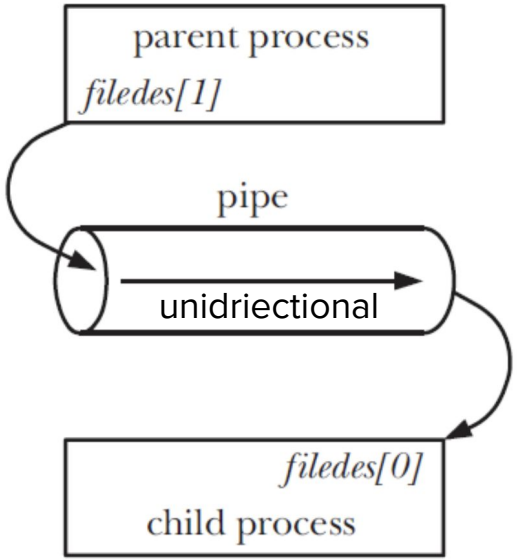
Linux IPC

- communication
  - data transfer
    - pipe
    - FIFO
    - stream socket
  - shared memory
    - SysV shared memory
- synchronization
  - semaphore
- signals

- Data transfer: user space ➜ kernel ➜ user space
- Shared memory: single copy in user space

pipe

- Pipe == byte stream buffer in kernel
  - Sequential (can't lseek())
  - Multiple readers/writers difficult
- Unidirectional
  - Write end + read end
- Pipes are anonymous
  - No name in file system



Closing unused file descriptors

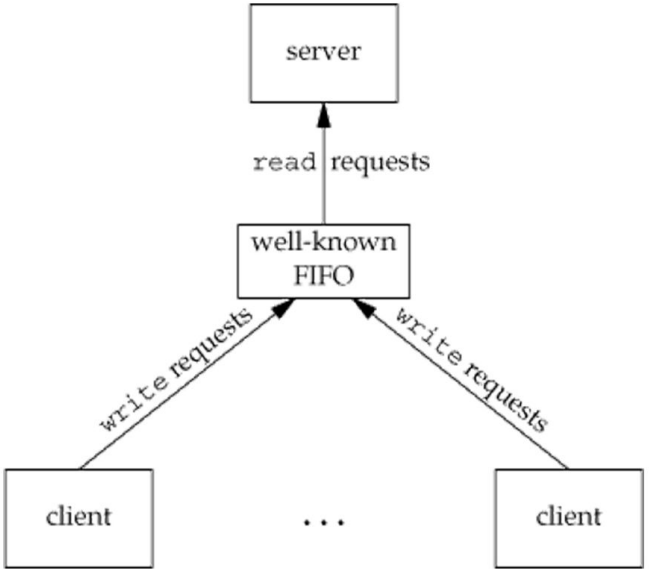
- Parent and child must close unused descriptors
  - Necessary for correct use of pipes!
- close() write end
  - read() returns 0 (EOF)
- close() read end
  - write() fails with EPIPE error + SIGPIPE signal

I/O on pipes

- read() blocks if pipe is empty
- write() blocks if pipe is full
- Writes <= PIPE_BUF guaranteed to be atomic
  - Multiple writers > PIPE_BUF may be interleaved
  - POSIX: PIPE_BUF at least 512B
  - Linux: PIPE_BUF is 4096B
- Can use dup2() to connect filters via a pipe

FIFO (named pipe)

- (Anonymous) pipes can only be used by related processes
- FIFOs == pipe with name in file system
- Any process can open and use FIFO
- I/O is the same as for pipes



SysV Shared memory

- Share memory between unrelated process, without creating file in (traditional) filesystem
  - Don’t need to create a file
  - Avoid file I/O overhead
- Commands: ipcs

```
[user@nplinux1 ~]$ ipcs

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
```

Sockets

- “A socket is endpoint of communication...”
  - ... you need two of them
- Bidirectional