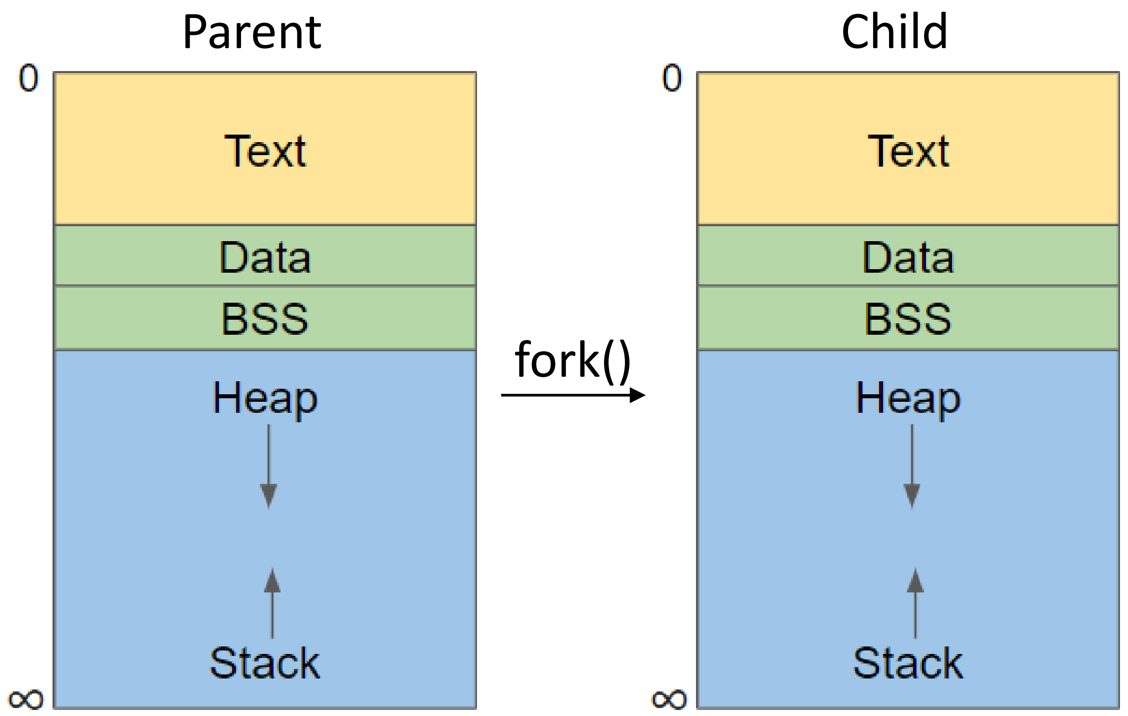
CS3841 – Design of Operating Systems
Inter-Process Communication

Problem

- Parent and child process do not share address spaces

Question

- How can parent and child communicate?

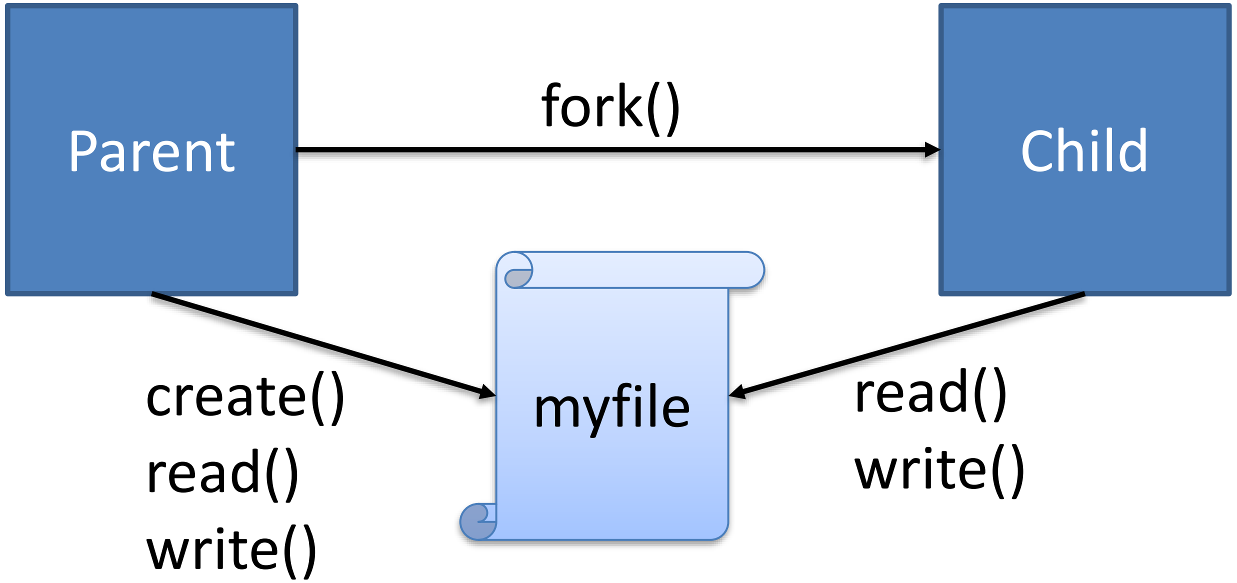


Inter-Process Communication

- Mechanism provided by the operating system from one process to communicate with another
- How?
  - Shared file
  - Pipes - classic
    - Anonymous
    - Named
  - Message Passing
  - Signals
  - Shared memory
  - Memory-mapped file
  - Sockets
    - IP
    - Unix Domain

File System

- Memory is NOT shared between parent and child
- Open file descriptors ARE shared
- Shared file



File System

- Shared named file
  - File descriptors are shared
  - File position pointers are also shared

- Parent opens file
  - file position pointer is at location 0
- Child writes “HELLO” to file –
  - File position pointers is at location 5
- Parent reads from the file
  - Parent gets no data returned

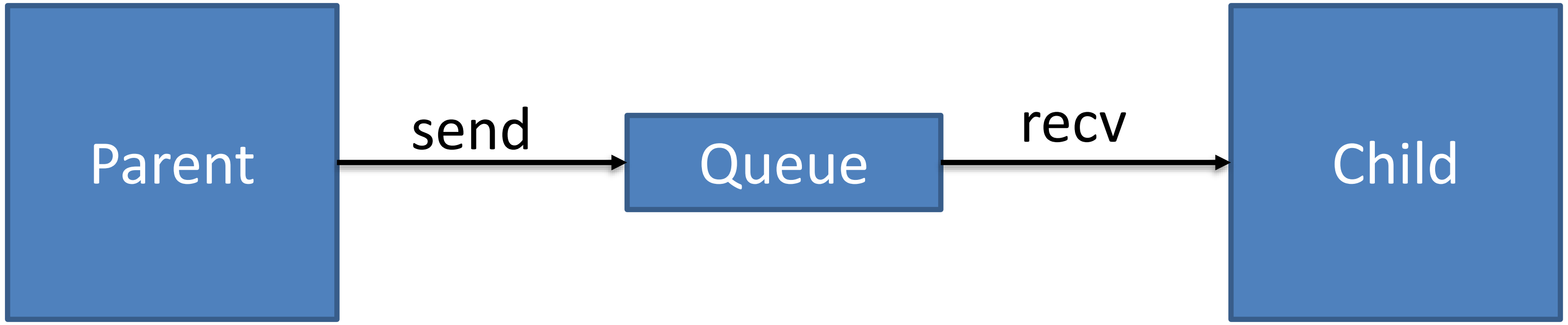
- Parent opens file
  - file position pointer is at location 0
- Child writes “HELLO” to file –
  - File position pointers is at location 5
- Parent seeks to beginning of file
  - File position pointers is at location 0
- Parent reads from the file
  - Parent gets “HELLO”

File System – Pipes

- Provides an anonymous communication mechanism between parent and child
- pipe() system call
- Creates 2 file descriptors – one for each end
- Kernel maintains position pointers
- Named – FIFO (First in First out)

Message Passing

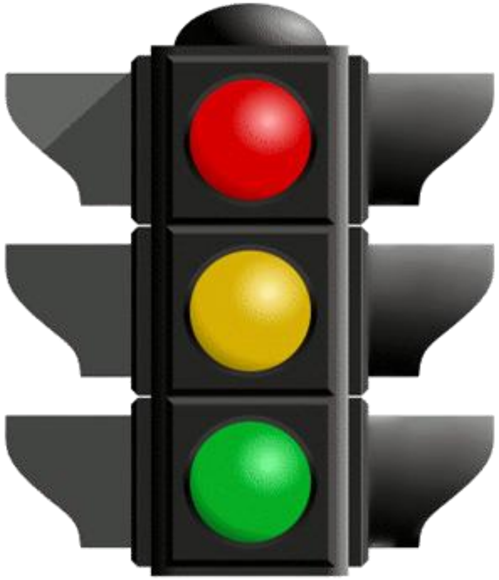
- Messages are sent to and received from queues
- Can contain arbitrary binary data (int, struct, etc.)



Message Passing

- Lots of options on how to send and receive
- Synchronous or asynchronous (blocking or non-blocking)?
  - Blocking send – Sending process is blocked until the receiving process reads message
  - Non-blocking send – sending process sends message and continues
  - Blocking receive – receiver blocks until a message is available
  - Non-blocking receive – receiver retrieves either a valid message or a null message
- Message queues live beyond the life of the processes that use them

Signals



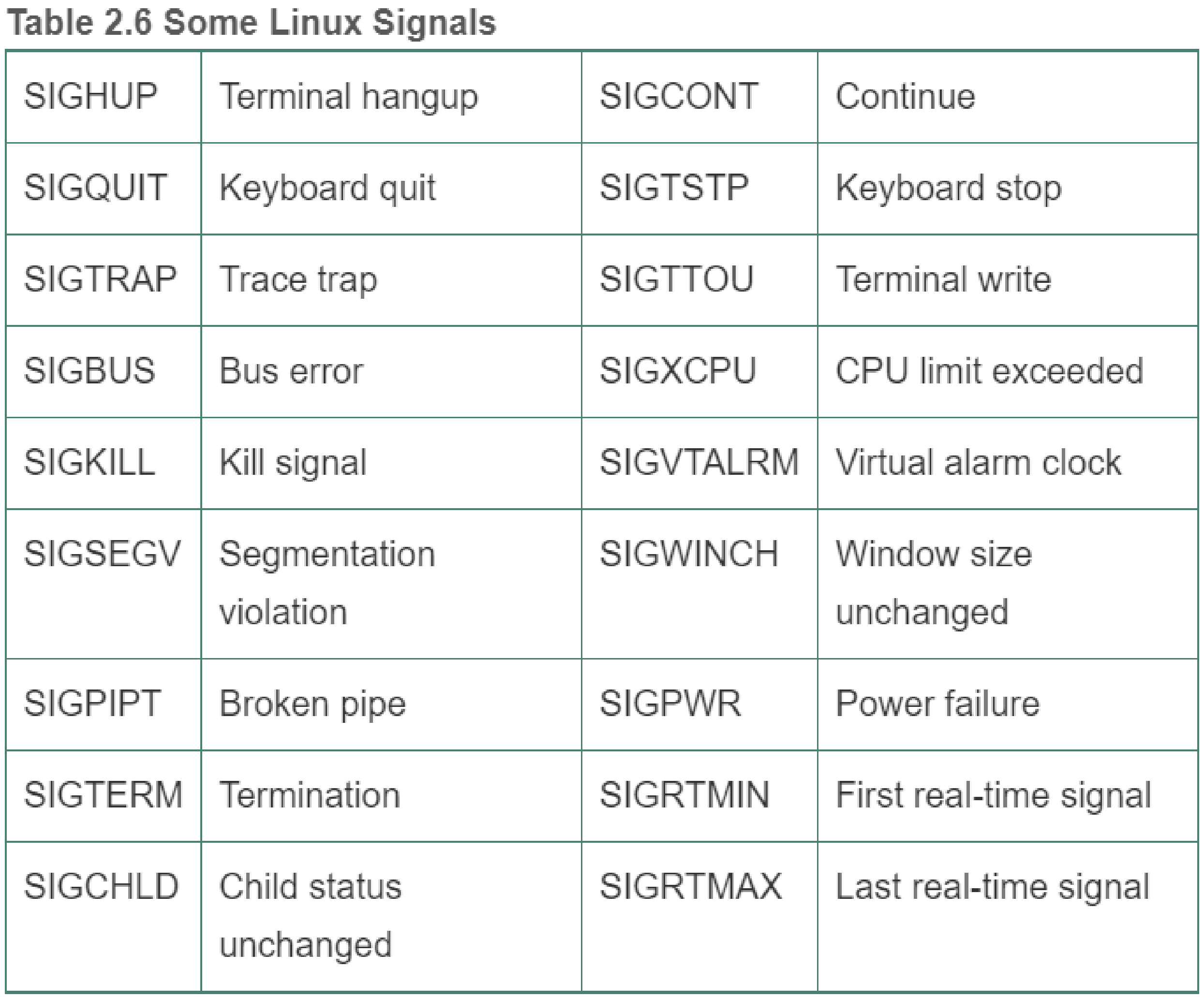
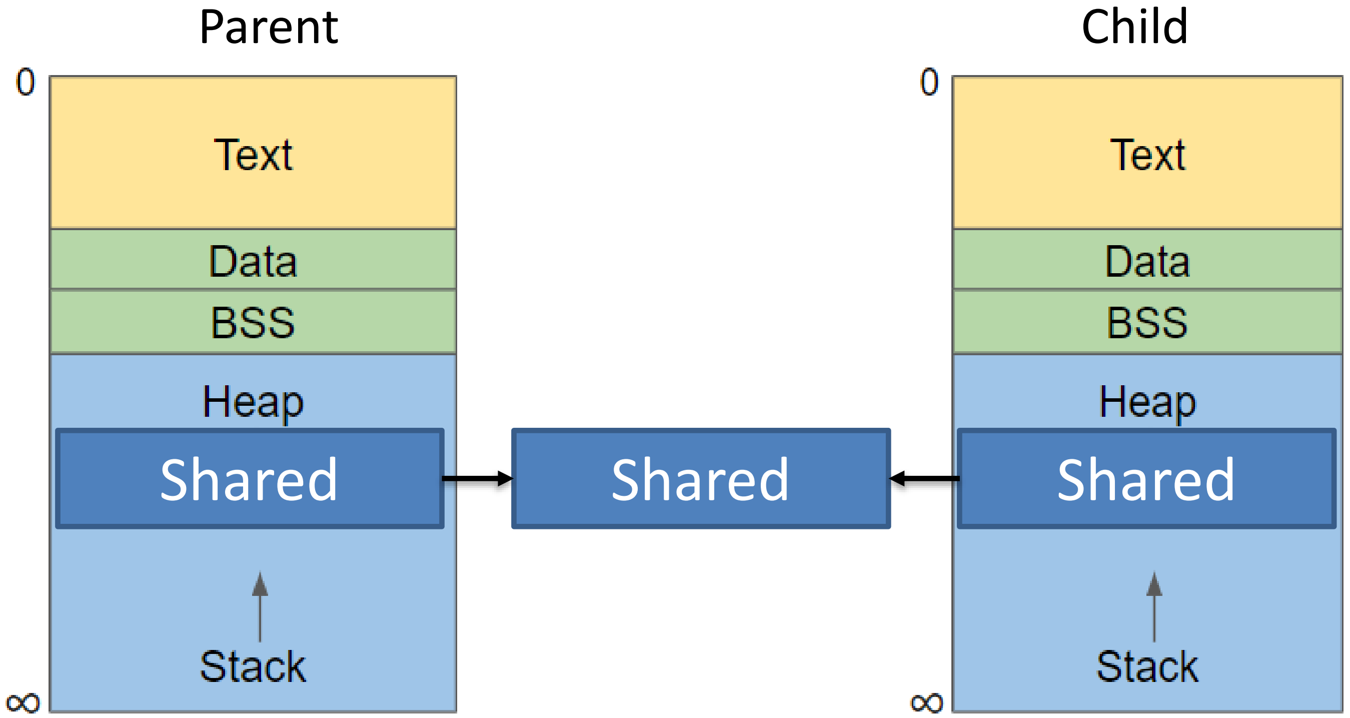


Table 2.6 Some Linux Signals

SIGHUP	Terminal hangup	SIGCONT	Continue
SIGQUIT	Keyboard quit	SIGTSTP	Keyboard stop
SIGTRAP	Trace trap	SIGTTOU	Terminal write
SIGBUS	Bus error	SIGXCPU	CPU limit exceeded
SIGKILL	Kill signal	SIGVTALRM	Virtual alarm clock
SIGSEGV	Segmentation violation	SIGWINCH	Window size unchanged
SIGPIPT	Broken pipe	SIGPWR	Power failure
SIGTERM	Termination	SIGRTMIN	First real-time signal
SIGCHLD	Child status unchanged	SIGRTMAX	Last real-time signal

Shared Memory

- Memory is NOT shared between parent and child by default
- Shared Memory – special memory map between parent and child



Shared Memory

- Pros
  - – Can share variables between parent and child - treat like global data
- Cons
  - – Takes up address space from other variables
    - How much shared memory should we allocate?
  - – Synchronization
    - What happens if multiple processes try to write to the same variable at the same time?