دانشگاه علم و صنعت ایران

Iran University of Science & Technology

IUST

# Introduction to
# Inter-Process Communication

# Zombie and Orphan processes

# Orphan Process

- A process whose parent process no more exists Fetch Instruction at PC
- either finished or terminated without waiting for its child
- the orphan process is soon adopted by init process, once its parent process dies.

# Orphan Process

```c
#include<stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    // Create a child process
    int pid = fork();

    if (pid > 0)
        printf("in parent process");

    // Note that pid is 0 in child process
    // and negative if fork() fails
    else if (pid == 0)
    {
        sleep(30);
        printf("in child process");
    }

    return 0;
}
```

# Zombie Process

- When a process ends, all of the memory and resources associated with it are deallocated so they can be used by other processes.

# Zombie Process

- However, the process's entry in the process table remains.
- The parent is sent a SIGCHLD signal indicating that a child has died; the handler for this signal will typically execute the wait system call, which reads the exit status and removes the zombie.
- The zombie's process ID and entry in the process table can then be reused.
- However, if a parent ignores the SIGCHLD, the zombie will be left in the process table.

# Zombie Process

```
// A C program to demonstrate Zombie Process.
// Child becomes Zombie as parent is sleeping
// when child process exits.
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    // Fork returns process id
    // in parent process
    pid_t child_pid = fork();

    // Parent process
    if (child_pid > 0)
        sleep(50);

    // Child process
    else
        exit(0);

    return 0;
}
```

# What does a zombie look like?

- normal (no zombie)

```
$ ps
```

```
PID TTY          TIME CMD
1074 pts/2    00:00:00 bash
1280 pts/2    00:00:00 parentTest.exe
1281 pts/2    00:00:00 childTest.exe
1283 pts/2    00:00:00 ps
```

# What does a zombie look like?

- abnormal (zombie)

```
$ ps
```

```
PID TTY         TIME CMD
1074 pts/2    00:00:00 bash
1280 pts/2    00:00:00 parentTest.exe
1281 pts/2    00:00:00 childTest.exe <defunct>
1288 pts/2    00:00:00 ps
```

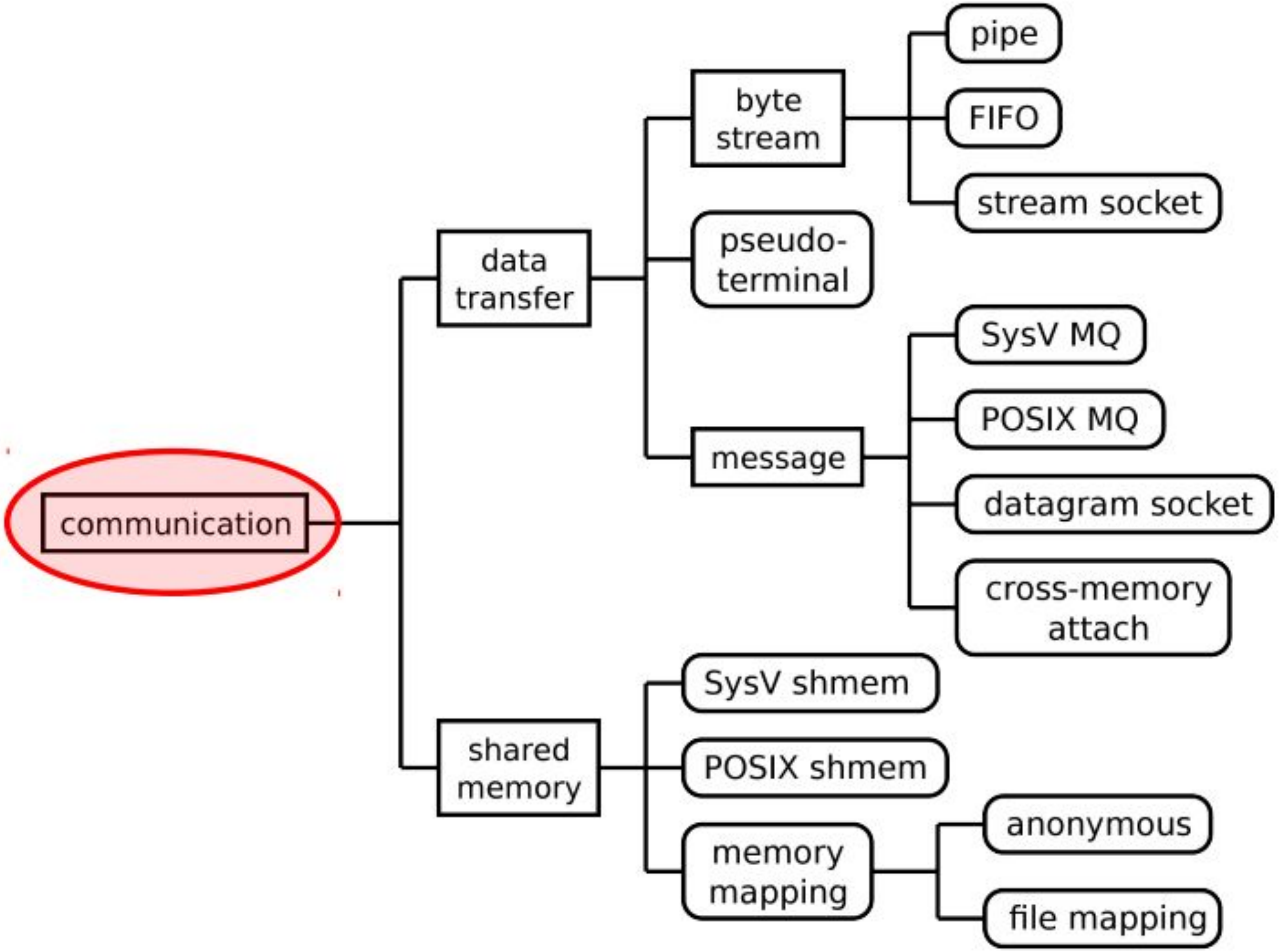
# What does a zombie look like?

```
$ ps -l
```

```
Warning: /boot/System.map has an incorrect kernel version.
  F S   UID   PID  PPID  C PRI  NI ADDR    SZ  WCHAN TTY          TIME CMD
000 S   561  1074  1073  0  76   0    -   628 11a418 pts/2    00:00:00 bash
000 S   561  1301  1074  0  70   0    -   436 11f22b pts/2    00:00:00 parentTes
004 Z   561  1302  1301  0  70   0    -     0 119ffb pts/2    00:00:00 childTest
000 R   561  1320  1074  0  77   0    -   646      - pts/2    00:00:00 ps
```

# Inter-Process Communication

# Inter-Process Communication



# Inter-Process Communication

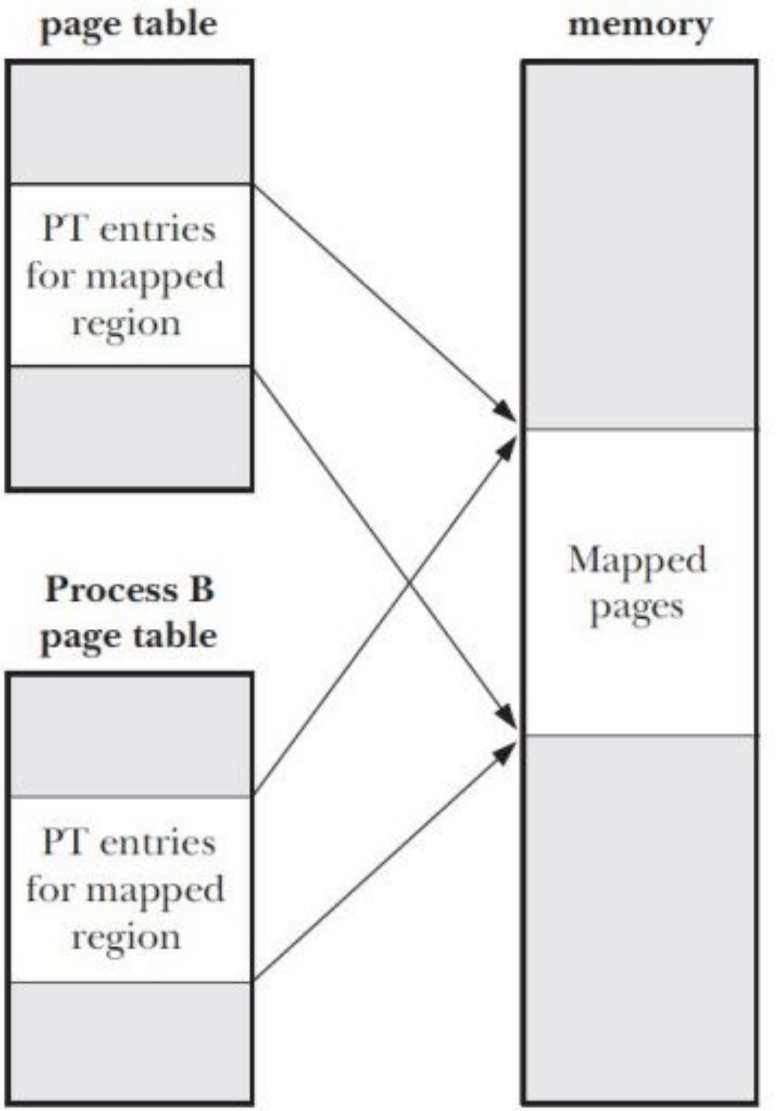
- Shared Memory
- Pipe

# Shared Memory

- Processes share same physical pages of memory
- Communication == copy data to memory Efficient;
- Data transfer: user space ==> kernel ⇒ user space
- Shared memory: single copy in user space

# Shared Memory

- Processes share same
- Physical pages of memory



page table

PT entries for mapped region

Process B page table

PT entries for mapped region

memory

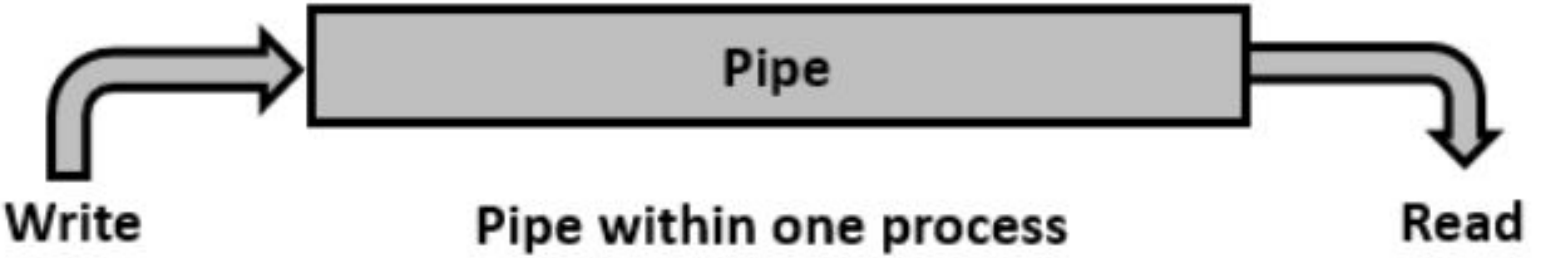
Mapped pages

# mmap() syscall

- addr = mmap(daddr, len, prot, flags, fd, offset);
- daddr – choose where to place mapping; Best to use NULL, to let kernel choose
- len – size of mapping
- prot – memory protections (read, write, exec)
- flags – control behavior of call: MAP_SHARED, MAP_ANONYMOUS
- fd – file descriptor for file mappings
- offset – starting offset for mapping from file
- addr – returns address used for mapping

# Pipe

- standard output from one process becomes the standard input of the other process
- Pipe == byte stream buffer in kernel
- Pipe is one-way communication only
- It opens a pipe, which is an area of main memory that is treated as a "virtual file".



# Pipe

- Step 1 – Create pipe1 for the parent process
- Step 2 – Create pipe2 for the child process
- Step 3 – Close the unwanted ends
- Step 4 – Parent process to write a message and child process to read
- Step 5 – Child process to write a message and parent process to read