**Lab 5 Inter Process communication**

GOAL: We will explore how 2 running processes can exchange information

Definitions:

-Pipe: an operating system storage unit for inter-process communication, usually is a file on the disk, that is written to kernel reserved storage.

* Pipe has 2 Operations
  + read – Retrieves/reads X number of bytes written to the pipe, and copies it to main memory
  + write – Puts/Writes x number of bytes to the pipe, copying it from main memory to the pipe
    - both use a buffer in main memory to work with the data, for either reading or writing.
* A Pipe is uni-directional (can only communicate in one direction),
* To prevent data corruption and data continuity problems, pipes can only be used once

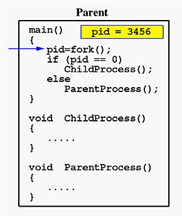
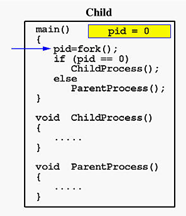
-File Descriptor: A integer array that stores information about the pipe,

* When accessing a pipe a FD is used to reference it.
* A file descriptor is an integer array of size 2,
* fd[0] represents the read end of the pipe, fd[1] represents the write end of the pipe

-Fork: An OS system function in which a “Parent” process creates a subprocess known as the “Child” process, which is a copy of the parent, but is allocated separate working memory.

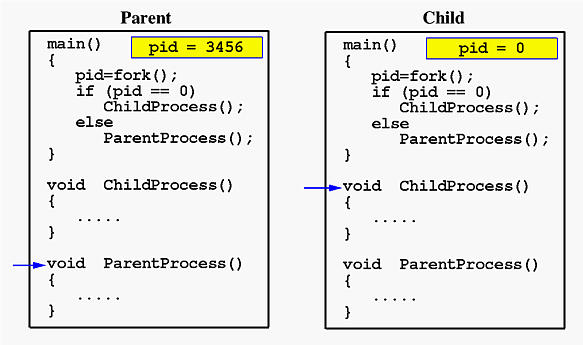
* These 2 processes now run in parallel
* Child process is a sub process of the parent if the parent is terminated so is the child.
* If the parent process reaches the end of its instructions, it is blocked, i.e, it must wait for the child to finish before it can finish

Code example:

The kernel ,Creates a new process and copies all code and variable from the calling(parent) process.

-Each process will take different paths in the code depending on if it is the parent or the child.



*Source:https://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/create.html*

C function you will use:

All functions are included in “unistd.h”

This library is a linux/unix only library.

* pipe(int \* fd); , where fd is an array of size 2, for the file descriptor
* fork();

int pid=fork();

on successful fork() in the parent, this function returns the child process id number, therefore pid = int>0,

the child process also gets a return value from this function, the child gets process id 0, therefore pid = 0, you can use this to determine which process is which.

* read(int \* fd, void \* buffer, size\_t count);
  + fd - is your file descriptor you created,
  + buffer - is a variable of any data type,
  + count is the number of bytes your are reading or writing, char is 1 byte int is 4 bytes.
* write(int \* fd, void \* buffer, size\_t count);
* close(fd[0] or fd[1]), closes read or write end of the pipe, always close the ends you are not using before you read or write.

Example flow with fd1

-Parent write pipe0:

Close(fd1[0])

Write(…)

Close(fd1[1])

-Child read pipe0:

Close(fd1[1])

Read(…)

Close(fd1[0])