Chapter 8.3) fork Function

An existing process can create a new one by calling the fork function

#include <unistd.h>

pid\_t fork(void);

Returns 0 in child, process ID of child in parent, -1 on error

* New process is called the child process
* fork () is called once but returns twice; 0 and PID
* PID of child is returned in parent because a process can have more than one child and there is no function that allows a parent to have the PIDs of its children
* 0 is returned in child because a child can only have a single parent and can always use getppid to get the its parent’s PID.
* PID of 0 is reserved for kernel so it’s not possible for 0 to be a child’s PID.
* Child is a copy of its parent in terms of data space, heap and stack
* Memory not shared, copied.
* Text segment is shared between child and parent
* fork () can fail if (a) there already too many processes are already in the system or (b) the total number of processes for this real user ID exceeds the system’s limit.
* Two uses of fork ()
  + When a process wants to execute each section of the code at the same time using parent and child
  + When the program wants to execute a different program

File sharing

* All open file descriptors in a parent are duplicated in the child
* For every open file descriptor, the parent and child share a file table entry
* Parent and child share the same file offset
* Handling of file descriptors after a fork ()
  + Parent waits for the child to complete
    - Parent doesn’t do anything; file offsets update accordingly when child terminates
  + Parent and child go their own ways
    - Parent and child close the descriptors that they don’t need; neither interferes with each other’s open descriptors

8.10) exec Functions

* exec replaces a process with the new program that called exec
* new program starts at its main
* process ID is not replaced, process is replaced. Not created
* seven different exec functions: execl, execv, execle, execve, execlp, execvp, fexecve
  + first 4 takes a pathname argument
  + next 2 takes a filename argument
    - filename containing a slash = pathname
    - file is searched for in directories specified by the PATH environment variable
    - filename is used when executable file is found but is not machine executable, file is seen as a shell script
  + last one takes a file descriptor argument
    - relies on caller to find the correct file using the file descriptor
* all seven exec functions return -1 on error and nothing on success
* In many UNIX system implementation, execve is a system call within the kernel; other 6 are library functions
  + execlp and execvp process the PATH environment variable to look for the filename
  + Fexecve converts using /proc the file descriptor into a pathname for execve to execute
* p means function takes a filename argument and uses the PATH environment variable to find the executable file
* l means the function takes a list of arguments and is mutually exclusive
* v means the function takes an argv[] vector
* e means the function takes an envp[] array instead of using the current environment

A picture containing scatter chart

Description automatically generated