



HOLBERTON SCHOOL

SIMPLE SHELL PROJECT

PID (Process Identifier): A PID is a unique number given to each running program on a computer. It helps the system track and manage different processes, like giving each running program its own “name tag”.

PPID (Parent Process Identifier): A PPID is the number (ID) of the parent process that started another process. It shows which process created it. For example, if a program starts another program, the first one is the “parent”, and its ID the PPID of the second one.

EXECVE: Is a system call that runs a new program in place of the current one. It replaces the current process with new program, using the given file, arguments, and environment variables. Once “execve” runs, the old program is gone, and only the new one continues.

FORK: Is a command that creates a new process. The new process is called the “child”, and the original one is the “parent”. Both processes run the same program, starting from where “fork” was called.

WAIT: Is a system call that pauses a parent process until one of its child processes finishes. It is used to clean up “zombie” processes and retrieve the exit status of the child.

STRCSFN: Is a C function that counts how many characters in a string (“str1”) come before any character from another string (“str2”). It stops counting when it finds a match.

STRTOK: Is a C function that splits a string into parts (tokens) based on delimiters. It modifies the string and must be called repeatedly to get all tokens.

BREAK: Is a control flow keyword used to immediately terminate the execution of a loop or switch statement.

CONTINUE: Is a control flow keyword used in loops to skip the rest of the current iteration and move to the next one.

ECHO \$\$: “The echo \$\$” command in Linux is used to display the Process ID (PID) of the current shell.

ACCESS: In the context of a simple shell, “access” is a system call used to check the accessibility of a file. It determines whether the calling process can access a specified file in a certain way.

CHDIR: Is a system call in Unix and Unix-like operating systems used to change the current working directory of the calling process.

CLOSE: Is a system call used in Unix and Unix-like operating systems to close a file descriptor, thereby terminating access to the associated file.

CLOSEDIR: Is a system call used in Unix and Unix-like operating systems to close a directory stream that was previously opened by "opendir".

EXIT: Is a standard library function in C is used to terminate a program and return a status code to the operating system.

_EXIT: Is a system call in Unix and Unix-like operating systems used to terminate a process immediately without performing cleanup operations that are typically handled by the standard exit function.

FFLUSH: Is a function in C used to flush the output buffer of a stream, ensuring that all data in the buffer is written to the underlying file or device.

FREE: free is a standard library function in C used to deallocate memory that was previously allocated dynamically using functions like "malloc", "calloc", or "realloc".

GETCWD: Is a standard library function in C used to obtain the current working directory of the calling process.

GETLINE: Is a standard library function in C used to read an entire line from a specified input stream, dynamically allocating memory for the line as needed.

GETPID: Is a system call in Unix and Unix-like operating systems that returns the process ID (PID) of the calling process.

ISATTY: Is a function in C that checks whether a given file descriptor refers to a terminal device.



KILL: Is a system call in Unix and Unix-like operating systems used to send signals to processes. It can be used to terminate processes, among other functionalities.

MALLOC: Is a standard library function in C used for dynamic memory allocation. It allocates a specified amount of memory during the execution of a program.

OPEN: Is a system call in Unix and Unix-like operating systems used to open or create a file and obtain a file descriptor that can be used for subsequent file operations.

PERROR: Is a standard library function in C used to print a descriptive error message to the standard error stream ("stderr"). It provides a way to display the last error that occurred, as indicated by the global variable "errno".

PRINTF, FPRINTF, VF PRINTF, SPRINTF: These functions are part of the C standard library and are used for formatted output. They allow developers to display or store data in a specific format, making it easier to present information clearly.



READ: Is a system call in Unix and Unix-like operating systems used to read data from a file descriptor into a buffer.

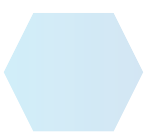

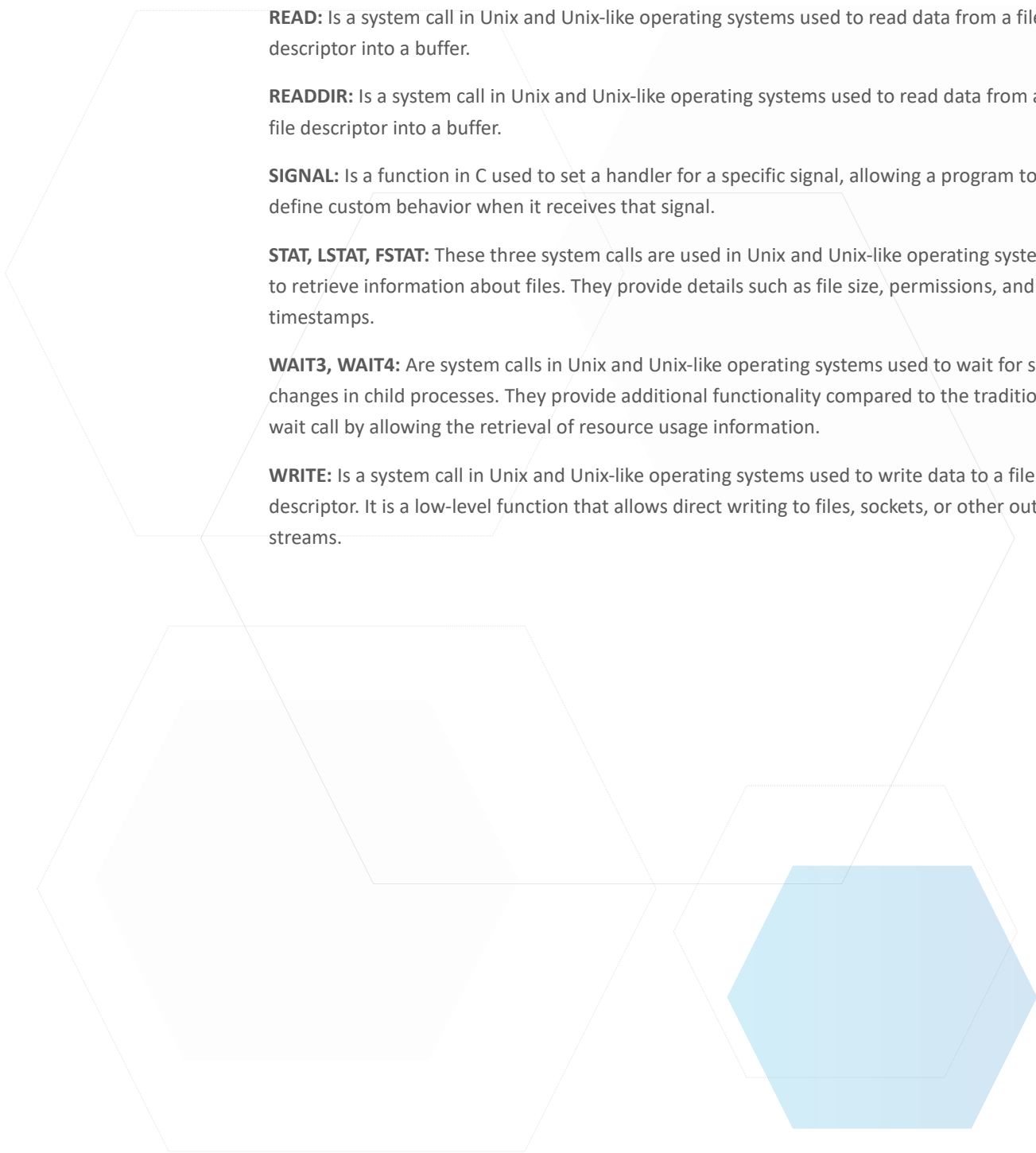
READDIR: Is a system call in Unix and Unix-like operating systems used to read data from a file descriptor into a buffer.

SIGNAL: Is a function in C used to set a handler for a specific signal, allowing a program to define custom behavior when it receives that signal.

STAT, LSTAT, FSTAT: These three system calls are used in Unix and Unix-like operating systems to retrieve information about files. They provide details such as file size, permissions, and timestamps.

WAIT3, WAIT4: Are system calls in Unix and Unix-like operating systems used to wait for state changes in child processes. They provide additional functionality compared to the traditional wait call by allowing the retrieval of resource usage information.

WRITE: Is a system call in Unix and Unix-like operating systems used to write data to a file descriptor. It is a low-level function that allows direct writing to files, sockets, or other output streams.



Example Code: PID with fork()

Here's an example of C code that illustrates the use of Process IDs (PIDs) using the `getpid()` and `fork()` functions. This program creates a child process and displays the PIDs of both the parent and child processes.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
    pid_t pid = fork(); // Create a new process
    if (pid < 0)
    {
        // Error occurred during fork
        perror("Error during fork");
        exit(EXIT_FAILURE);
    }
    else if (pid == 0)
    {
        // Code executed by the child process
        printf("I am the child process. My PID is %ld and my parent's PID is %ld.\n", (long)getpid(),
            (long)getppid());
    } else
    {
        // Code executed by the parent process
        printf("I am the parent process. My PID is %ld and my child's PID is %ld.\n", (long)getpid(),
            (long)pid);
    }
    return( EXIT_SUCCESS);
}
```

Example Code: Getting PPID in C

Here's an example code in C that demonstrates how to retrieve the Parent Process ID (PPID) of a specific process. This example creates a child process and then prints both the PID of the child and its PPID.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
    pid_t pid = fork(); // Create a new child process
    if (pid < 0)
    {
        // Error occurred during fork
        perror("Error during fork");
        exit(EXIT_FAILURE);
    } else if (pid == 0)
    {
        // Code executed by the child process
        pid_t parent_pid = getppid(); // Get the parent process ID
        printf("Child Process: My PID is %ld and my Parent PID is %ld.\n", (long)getpid(),
            (long)parent_pid);
    }
    else
    {
        // Code executed by the parent process
        printf("Parent Process: My PID is %ld and I created a child with PID %ld.\n", (long)getpid(),
            (long)pid);
        // Optionally wait for the child to finish
        wait(NULL); // Wait for the child process to terminate
    }
    return (EXIT_SUCCESS);
}
```

Shell Script Example

file ./hello.sh:

```
#!/usr/bin/tcsh -f
echo Hello World

% chmod +x ./hello.sh
% ./hello.sh
```

Shell Script features

- Variables for storing data
- Decision-making control (e.g. if and case statements)
- Looping abilities (e.g. for and while loops)
- Function calls for modularity
- Any UNIX command:
 - file manipulation: cat, cp, mv, ls, wc, tr, ...
 - utilities: grep, sed, awk, ...
- comments: lines starting with "#"