Kohl Johnson

Activity 2

06/15/2024

# Processes

Part 2

A screenshot of a computer

Description automatically generated

**Theory of Operation:** The fork() function is a system call that creates a new process via duplicating the calling process. The new process is called the child, and the original is called the parent. They run in different memory spaces and have the same content (depending on how you code it).

Part 3

A screenshot of a computer

Description automatically generated

**Theory of Operation:** The posix\_spawn() function creates a new process from the specified process image. The new process is constructed from a regular executable file called the new process image file. Upon successful completion, the function returns the process ID of the child process to the parent process.

Waitpid() is used to wait until a specific process finishes, based on a process ID. It can be used to wait for any group of child processes.

# Signals

Code Screenshot

A computer screen with a black screen

Description automatically generated

Output Screenshot

A screenshot of a computer

Description automatically generated

**Theory of Operation:** The system call is used to create processes, it takes no arguments and returns a process ID. These new process become child processes of the calling process and both the parent and child will execute the next instruction(s) following the call.

# Threads

Output Screenshot

A computer screen with a purple screen

Description automatically generated

Code ScreenshotA computer screen with a black screen

Description automatically generated

**Theory of Operation:** POSIX threads (also known as pthreads) are single processes that can contain multiple threads. All of which are executing the same program, share the same memory, but each thread has its own stack (automatic variables).

# Bad Bank Example

Code Screenshot

A computer screen with a black screen

Description automatically generated

Output Screenshot

A computer screen with a black screen

Description automatically generated

**Theory of Operation:** The program did not give the correct result because both threads were attempting to change the global BANK\_BALANCE at the same time. While the result is “close” to 2 million, it still is not correct.

# Mutex

Code Screenshot

A computer screen with a black screen

Description automatically generated

Output Screenshot

A computer screen with a black screen

Description automatically generated

**Theory of Operation:** The program now behaves correctly because weve implemented the “critical section” of mutual exclusion which keeps the two processes from changing the BANK\_BALANCE global var at the same time.

# Semaphores

Code Screenshot

A computer screen with a black screen

Description automatically generated

Output Screenshot

A computer screen with a black screen

Description automatically generated

**Theory of Operation:** As stated above, we keep the two threads from entering their critical section at the same time, which prevents the errors we had with original bad bank code.