Lab 3: Signal Handling and Process Control

In this lab you will be given two pieces of code, bulk\_scheduler.c and gambler.c. You will also receive PCB.h which holds a structure definition that we will use throughout the exercise.

bulk\_scheduler’s job is to coordinate and run several instances of gambler sequentially. You will notice that bulk\_scheduler is a skeleton. There is an outline for the program but much of the ‘meat’ is missing.

Go through the code and look for lines marked with // TODO:

Completing these tasks help you master the craft of systems building.

It may even give you the experience you needed to put the ghost of MCP to rest.

Functions:

The following notes are intended complement the TODOs in the file.

void InitializePCBs()

This function should fill out ProccessControlBlock as found in PCB.h.

void FreePCBs()

Reset any variables that and free up any memory that you allocated.

Void sigusr2\_handler(int sugnum)

This function sets Exit to 1, but many processes may try to call this at the same time. Maybe sure to block concurrent access to shared variables.

void WaitAndExecOrCleanup(struct ProcessControlBlock \*prgm)

Run execvp using the process control block you are given. Save the result.

Free up your PCB and get out

void LaunchProcess(struct ProcessControlBlock \*launch)

Fork a child, then find out of you’re a child or not. Then call WaitAndExecOrCleanup if it’s your job to do so.

void LaunchAllProcesses()

Subscribe to sigusr1, then unsubscribe after the job is done.

void settimer(int sec, int microsec)

Set a timer. See gnu documentation for details about the itimerval struct.

We just need to initialize this structure and set it with the time values that were passed in.

Don’t forget to ‘set’ it after you’re done initializing things.

<https://www.gnu.org/software/libc/manual/html_node/Setting-an-Alarm.html>

void stoptimer()

Set up a timer and set all of its values to 0. Then set it.

void sigalarm\_handler(int signum)

What is the appropriate signal to send for each case that may be? Remember, kill(proc\_id, SIGNAL) means send a SIGNAL to proc\_id.

void sigchild\_handler(int signum)

We’ve received a call from the child process.

What should we do now? Remember that if we are executing this function, we must block other processes from calling it.

Hint: waitpid()will be useful here

Signal Documentation:

The following are brief descriptions of some of the signals you will need to complete this exercise. Refer to the documentation for more detail.

sigaddset()

Add a signal to the set of signals your program knows to look out for.

<http://pubs.opengroup.org/onlinepubs/7908799/xsh/sigaddset.html>

signal()

“I’m ready to receive a signal”

<http://man7.org/linux/man-pages/man2/signal.2.html>

Kill()

Send a signal

<http://man7.org/linux/man-pages/man2/kill.2.html>

SIGUSR1, SIGUSR2

General purpose signals that we will be using to coordinate process execution

<https://www.gnu.org/software/libc/manual/html_node/Miscellaneous-Signals.html>

SIGALRM

Set up an alarm to fire after some period of time. When it fires do something

<https://linux.die.net/man/2/alarm>

sigprocmask()

Use this with SIG\_BLOCK and SIG\_UNBLOCK to control when and where signals can be called. *i.e.*, if you are currently handling a signal you will want to call sigprockmask with SIG\_BLOCK on the signal you are handling. When you are done handling the signal, use the function with SIG\_UNBLOCK.

<https://linux.die.net/man/2/sigprocmask>

fork()

Create a child process.

<http://man7.org/linux/man-pages/man2/fork.2.html>

execvp()

Run a command.

<https://linux.die.net/man/3/execvp>

waitpid()

Waits for a process to change state.

Pass with WNOHANG to return immediately if no child has exited.

<https://linux.die.net/man/2/waitpid>