CS39002: Operating Systems Lab
Spring 2017

Assignment 6
Due: April 14th, 8 AM (Not PM)

In this assignment you will continue to work with the Pintos OS by adding some basic signal handling calls to Pintos.

Firstly, you will need to keep track of parent and children of a thread. The parent thread of a thread X is the thread that called thread_create to create X. You will also add a function called setlifetime(X) that will set the maximum lifetime of a process to X time ticks. See below what the maximum lifetimes is used for.

We will support 5 signals with names SIG_CHLD, SIG_KILL, SIG_CPU and SIG_UBLOCK, and SIG_US. Each of them has a default handler (CHLD_hanlder, KILL_handler, CPU_handler, UNBLOCK_handler, and USR_handler). The default handler is called with one parameter which is the id of the thread that sent the signal. The signals work as follows:

- SIG_CHLD is sent to a thread X by the OS when any of its children terminates. If X has already terminated when one of its children dies, no signal is sent. The handler for SIG_CHLD updates the count of children which has died so far and prints both the total number of children created by X (till the signal is delivered) and the number of children still alive.
- SIG_KILL can be sent by one thread to another. The handler simply terminates the receiving thread and prints a message. However, this signal can only be delivered by a thread to a descendant thread (children, grandchildren, …), otherwise the signal is not sent.
- SIG_CPU signal is sent to a thread by the OS if a predefined maximum lifetime of a thread is exceeded. The lifetime of a thread includes both its running and waiting times. The handler prints the maximum lifetime set and terminates the thread.
- SIG_UNBLOCK signal can be sent by any thread to any other thread. The handler unblocks the receiving thread if it is blocked. If the receiving thread is already unblocked, no action is taken.
- SIG_USER can be sent by any thread to any other thread. The default handler just prints which thread sent the signal to which thread and returns.

With the above signals, you will have to implement the following functions similar to signals in Linux, except that process ids will be replaced by thread ids.

- signal() (since we do not support user-defined signal handlers, the only options supported for the second argument are SIG_IGN and SIG_DFL with same meaning)
- kill() (should be valid only for SIG_KILL and SIG_UNBLOCK subject to the conditions stated, all other cases are error cases. Returns 0 on success, -1 on error)
- sigemptyset(), sigfillset(), sigaddset(), sigdelset(), sigprocmask() (all of them have the same meaning as in Linux, with the same meaning of masking signals. The mask is 4 LSB bits of an unsigned short. Masking SIG_KILL has no effect.).

Signals can be raised at any time (subject to the conditions mentioned in the description of the signals). All signals other than SIG_UNBLOCK are delivered to a thread when the thread is about to run again (next context switch where the thread is chosen for running). For these signals, after choosing the next thread to run, the signal handler is first run and then the actual switching to the receiving thread is done (note that all handlers simply require access to OS data structures only, no particular user level data of the receiving thread needs to be accessed). SIG_UNBLOCK signal is delivered to the receiving thread when the next context switch happens irrespective of which thread is chosen to run. In this case, after choosing the next thread to run from the ready queue,, the signal handler is run first (which changes some queues and PCB fields for the receiving thread) and then the usual context switch happens. All signals are queued till delivery. Multiple same signals to the same thread are queued up as a single signal and delivered to the receiving thread only once (with the last thread id taken as the sending thread).

Create a proper signal.h and signal.c file to add the functions and add it to the Pintos code base. Submit the files signal.c, signal.h, and any other Pintos files you change. No new files should be added. Also submit a small pdf file describing (i) which files you changed, (ii) what data structures you added and in which file, and (ii) your design. The pdf should be max 2-3 pages long.