PiBox - Bug #712
Fix thread management in all apps

26 Jun 2019 10:27 - Hammel

Status:	Closed	**Start date:**	26 Jun 2019
Priority:	Immediate	**Due date:**	
Assignee:	Hammel	**% Done:**	100%
Category:	04 - Applications	**Estimated time:**	0.00 hour
Target version:	1.1.0 - Upgrades		
Severity:	01 - Critical		

Description

I have lousy thread management in my apps. The current design for all apps is this.

1. Start all threads
2. each thread spins looking for work
    1. if no work, short delay and spin again
3. spin in main thread till exit

This has worked fine up till now but it sucks. The threads are polling for work and spinning wildly if there isn't any. There are artificial delay loops to slow them down, but it's sucking the life out of the Pis, especially the Pi Zero. I disable some of the threads on Xeon builds but that's a hack. I need better thread synchronization.
What should really happen is this.

1. Start event queue thread waiting on network or other I/O via select() or similar.
2. Inbound message placed on appropriate queue.
3. start thread for that queue is called.

In the other thread.

1. If started, return (prevents multiple thread starts)
2. Run the queue until exhausted.
3. Exit the thread.

The start-thread call uses a mutex on a variable that can tell when the thread has exited.

There should never be more than one thread started when the app starts and that thread should be waiting on kernel events (network or other I/O) so that it's not polling and chewing up CPU time.

In other words: apps should be idle unless there is actual work to do!
Apps to review and fix:

1. piboxd
2. appmgr
3. videofe
4. musicfe
5. jarvis
6. picam
7. pidialer
8. pipics
9. pixm

Note: Everything that uses the piboxTouchProcessor() should be reviewed but mostly the piboxTouchProcessor() should be checked to make sure it's event driven and not polling. I don't think msgProcessor.c or queueProcessor.c in libpibox are being used yet but they should be checked too.

Related issues:

Blocked by VideoFE - Bug # 714: omxplayer FIFO control is broken	**Closed**	**15 Sep 2019**
Blocked by piboxd - Bug # 721: Missing resolv.conf causes MT_NET SET_IPV4 to ...	**Closed**	**20 Oct 2019**

Associated revisions

Revision 6ee79aa4 - 13 Sep 2019 14:34 - Hammel

RM #712: Reduce the size of the psplash image so it fits on all supported screens. Remove white pixels on corners of progress bar.

Revision 6ee79aa4 - 13 Sep 2019 14:34 - Hammel

RM #712: Reduce the size of the psplash image so it fits on all supported screens. Remove white pixels on corners of progress bar.

Revision 1659960d - 13 Sep 2019 14:36 - Hammel

RM #712: Use pthread_cond_wait/signal to avoid spinning thread functions.

Revision a31b5941 - 13 Sep 2019 16:32 - Hammel

RM #712: Fix pthread use to include conditional variables to avoid spinning in thread functions and have better synchronization.

Revision bef33259 - 13 Sep 2019 16:34 - Hammel

RM #712: Fix touchProcessor to use pthread conditional variables for better synchronization.

Revision f848f82f - 13 Sep 2019 16:35 - Hammel

RM #712: Make use of updated touchProcessor support for better pthread synchronization.

Revision 187c28c5 - 14 Sep 2019 10:35 - Hammel

RM #712: Add missing graphics for updated logo and progress bar.

Revision 187c28c5 - 14 Sep 2019 10:35 - Hammel

RM #712: Add missing graphics for updated logo and progress bar.

Revision c32e912d - 15 Sep 2019 14:14 - Hammel

RM #712: Use custom cfg for kiosk=1 mode to disable use of -o both, which is causing studder on the kiosk display.

Revision 8d8f8cc5 - 18 Sep 2019 12:00 - Hammel

RM #712: Add updated videofe.cfg and dbuscontrol.sh to autokiosk build.

Revision f1c63bd7 - 19 Sep 2019 09:59 - Hammel

RM #712: Add "rpi1" as a HW type to mkinstall.sh so the kernel gets copied to the /boot directory with the proper filename.

Revision 672abbc8 - 27 Oct 2019 10:56 - Hammel

RM #712: Switch from condition variables to semaphores for inbound network messages to avoid losing messages on race conditions.

Revision be1aab08 - 27 Oct 2019 12:11 - Hammel

RM #712: Switch from condition variables to semaphores for inbound network messages to avoid losing messages on race conditions.

History

#1 - 26 Jun 2019 15:22 - Hammel

- Status changed from New to In Progress

- % Done changed from 0 to 10

See: https://gist.github.com/rtv/4989304

See: https://www.geeksforgeeks.org/condition-wait-signal-multi-threading/

An alternative it to use pthread signals. We start each consumer thread as before but in each loop we wait on a signal

pthread_cond_wait(&cond, &mutex)

When we need to signal the thread from the producer (whoever is putting work on the queue)

pthread_cond_signal(&cond)

We need the mutex to set the signal and around the _wait() but the wait actually releases the mutex while it waits. Thread loops exhaust the queue before waiting on another signal, but producers always signal when they have work to do.

Q: do all signals have to be consumed by an equivalent wait?
A: I don't think so. They are async and immediately consumed, if anyone is waiting on them.

#2 - 25 Jul 2019 16:57 - Hammel

The first one to work on is the touch screen library. It's used by multiple apps. It's also the hardest one because it's a wrapper around another library (tslib) and the other library is doing the read. Currently I've configured the tslib to do non-blocking reads. So this thread is always busy. The library can be set to use blocking read()'s. So the problem is how to interrupt that read and catch it. This is because we need a way to wake the thread before shutting it down otherwise the thread in the blocked I/O will hang the app.

A blocking read can be interrupted in a thread with SIGINT. The read() will fail, which the touch library catches and then loops back around to try again. If we first call a thread function that resets the loop variable (causing it to exit on the next loop) then the SIGINT will cause the next loop to exit the thread.

Actually, we don't currently capture the fail correctly (from the perspective of blocking reads) so that's part of what has to be changed here.

A SIGINT will be propagated to all threads of a process. This isn't so bad as long as all threads handle blocked I/O the same way (non-blocking I/O would be caught with the existing shutdown() calls).

To test this I need to replace libpibox on the target. Apps dynamically link against that so I just need to update it to verify that the change still works.

I also need a way to measure if this change has any affect on an app. That's an interesting problem. I'm not sure the best way to do that yet.

#3 - 26 Jul 2019 15:33 - Hammel

This article discusses the use of SIGINT to break out of blocking reads and verifies that my intended design for the touch library should work. There is a very small time period where we'd have to send multiple SIGINT's to break out of the read() but we could simply do that three times in a row to make sure we weren't getting caught in that time period.

So the changes for the touch processor library (in libpibox):

1. Add a callback that a signal handler can call that sets ts_server=0 to disable the thread loop. This is wrapped in a mutex.
2. Change the thread loop to be while(1) because we want to use the mutex to read the loop variable.
3. Use a mutex to get a copy of the loop variable and test it, breaking out of thread loop if disabled.
4. Change call to ts_setup() to use a blocking read.
5. Add missing ts_close() to end of thread loop.
6. Update, as necessary, the tests of return value from ts_read() in thread loop to deal with possible EINTR.

#4 - 26 Jul 2019 15:36 - Hammel

- % Done changed from 10 to 20

Initial test with launcher seemed to work fine. I then updated pidialer to see if it would work correctly and it did.

So I think I have an improved thread processing capability in libpibox's touch processor. It actually does seem just a tad faster. The code is changed in the following (but not yet pushed):

1. libpibox
2. launcher
3. pidialer
4. psplash

The latter I did because the oversized logo has been driving me crazy for a long time and I just thought now was as good a time as any to fix it up.

Next up is to fix up appmgr, then piboxd. Fixing those two up should have major improvements since I know both have 1 or more threads (none of which are the touch processor).

#5 - 28 Jul 2019 10:49 - Hammel

appmgr:

- queueProcessor: needs pthreads_cond_wait/pthreads_cond_signal
- waiter: needs pthreads_cond_wait/pthreads_cond_signal

piboxd:

- smb: needs pthreads_cond_wait/pthreads_cond_signal to wake up smbProcessor instead of using smbEnable.
- regProcessor: blocks in recvfrom() but has artificial delay on EAGAIN/EWOULDBLOCK that could probably be removed.
- queueProcessor: needs pthreads_cond_wait/pthreads_cond_signal.
- timer: use alarm (like smb) to wake up cleanup processing.

#6 - 01 Aug 2019 21:16 - Hammel

appmgr is now working with mutex/cond updates.

On to piboxd.

#7 - 11 Sep 2019 20:47 - Hammel

piboxd has

- queueProcessor: needs pthreads_cond_wait/pthreads_cond_signal.
- timer: use alarm (like smb) to wake up cleanup processing.

These still need to be done.

- smb: needs pthreads_cond_wait/pthreads_cond_signal to wake up smbProcessor instead of using smbEnable.
- regProcessor: blocks in recvfrom() but has artificial delay on EAGAIN/EWOULDBLOCK that could probably be removed.

Once I get these done I need to build all the updated apps and test them all at once on the media server.
After that, if all goes well, I can rebuild/deply:

1. Media Center
    1. PiBox Server
    2. PiBox Player
    3. RPi 2 Server
2. Dev Platform
3. Kiosk test platform
4. Autokiosk (on kiosk test platform)

Then I need to test the Ironman project with the updates.
Finally, I can test Xeon with the updates and return to work on that project.

#8 - 13 Sep 2019 16:27 - Hammel

- % Done changed from 20 to 60

piboxd updates are complete and have been tested on RPi2 server with the media system. However the minor change to regProcessor can only be tested with Ironman. Since the change is very minor I'm going to go ahead and push now to make future testing easier with the metabuild.

appmgr and psplash changes have been pushed. Those changes really only affect the media system and if they work there it's safe to bet they work on other systems.

Dev platform is already tested because it only required the updates for the last build refresh (RM #687) and it must have worked or I couldn't have tested the above items.

Next: regenerate the system image with proper packages to test the kiosk and autokiosk.
After: make an ironman build and test it. It should be able to communicate with the IoT device and properly support network config via the web interface.
Finally: push all changes and test with xeon.

Note that none of the other apps (listed in the original description) require updates for thread management.

#9 - 15 Sep 2019 14:05 - Hammel

kiosk is not working properly. omxplayer is stuttering through videos. I tried running it manually from xterm and got the same problem. A little googling and I found this solution:
https://retropie.org.uk/forum/topic/17107/bug-solved-omxplayer-on-stretch-based-builds-stutters-on-composite

On the kiosk, where audio is played only out of the audio port (not HDMI) the use of -o both is causing the studdering of playback. Removing the -o both fixes the problem.

Removing the -o both option doesn't affect audio output to the audio port.

So the fix is to create a custom videofe.cfg that addresses the omxplayer command line issue and then pick it up if KIOSK=1 in the opkg/Makefile.am when building the opkg.

#10 - 15 Sep 2019 14:25 - Hammel

That's fixed but it's clear that the changes to the touchscreen library are not working. No touches are processed and I can't exit the app once it's started.
More debug is needed. I'll need to add a network adapter and enable debug on videofe to see if touches are being processed.

#11 - 15 Sep 2019 17:13 - Hammel

I don't think this is a touchscreen problem. Manually running omxplayer with the FIFO input works to start the player but no other commands are accepted via the FIFO after that by the player. Need to check if this is a known bug or if I will need to revert to the previous version in order to use the touchscreen.

A little searching and I found this bug report from Aug 6, 2019 which suggests using something like dbuscontrol.sh to control omxplayer. I'll have to switch to that in videofe, although using the C interface would be even better. I can find no solution to the broken FIFO problem as of yet.

#12 - 15 Sep 2019 20:19 - Hammel

The dbuscontrol.sh script doesn't work. It wants some files in /tmp/ that are created by something other than omxplayer, probably something in Raspbian. But it turns out those aren't really necessary because the files and their contents aren't really used by the commands in the script.

What is necessary is having the display set for X11. dbus apparently doesn't work without it, at least on PiBox. If I set DISPLAY=unix:0.0 and then run the following command, it will kill omxplayer using dbus:

```
dbus-send --print-reply=literal --session --dest=org.mpris.MediaPlayer2.omxplayer /org/mpris/MediaPlayer2
org.mpris.MediaPlayer2.Player.Action int32:15
```

So I can export DISPLAY when running the command or possibly set it in the environment before I run the external command (so it's picked up in the child environment). Interestingly, the script has options for a lot of omxplayer features managed via the dbus interface. So I can do things like pause or jump forward, back using the touchscreen regions. But first, just get exit to work with dbus.

Update: I've moved the FIFO problem to RM #714 and will track it's completion there. This issue is now blocked by that one.

#13 - 17 Sep 2019 21:27 - Hammel

- % Done changed from 60 to 70

RM #714 is completed and the kiosk appears to be working fine now.
~~I need to test autokiosk mode next.~~
~~Then test RPi1 media player.~~

After: make an ironman build and test it. It should be able to communicate with the IoT device and properly support network config via the web interface.
Finally: push all changes and test with xeon.

#14 - 20 Oct 2019 15:51 - Hammel

This can't be completed until I fix the bugs uncovered with testing ironman: RM #721, RM #722, RM #723.

#15 - 24 Oct 2019 22:33 - Hammel

The problem with ironman is that piboxd's changes for thread management don't work. Instead of using a mutex to notify queueProcessor() I need to use a semaphore (or a mutex that does the same thing). And I should only read one message per notification - aka a 1-to-1 semaphore post to message process. The way I implemented it there was a chance (that actually occurs with Ironman) that an inbound message could cause a notification while the last notification was not completely processed, meaning that a message could be queued for processing but the processor has missed the notification.

This should fix RM #721 once I fix it.

#16 - 27 Oct 2019 12:29 - Hammel

- % Done changed from 70 to 80

Ironman is working again. PiSensors has a minor bug but it's not build refresh related.

Now on to Xeon.

#17 - 30 Oct 2019 12:06 - Hammel

- Status changed from In Progress to Closed

- % Done changed from 80 to 100

Xeon is just plain broken and I need to restart the project to make sure the hardware is actually working right (like the charger is charging the battery). So thread management support there is kinda moot until the project gets rebooted.

Therefore I'm closing this issue as complete and will address thread issues in Xeon later.