

PA1 – Jetson's Audio Playback

Student Information

Integrity Policy: All university integrity and class syllabus policies have been followed. I have neither given, nor received, nor have I tolerated others' use of unauthorized aid.

I understand and followed these policies: Yes No

Name:

Date:

Submission Details

Final **Changelist** number:

Verified build: Yes No

Required Configurations:

Part A: YouTube Link:

Part B: YouTube Link:

Part B: PDF perforce loc:

Additional Discussion (What did you learn):

Verify Builds

- Follow the Piazza procedure on submission
 - Verify your submission compiles and works at the changelist number.
- Verify that only MINIMUM files are submitted
 - No – Generated files
 - *.pdb, *.suo, *.sdf, *.user, *.obj, *.exe, *.log, *.pdb, *.db
 - Anything that is generated by the compiler should not be included
 - No – Generated directories
 - /Debug, /Release, /Log, /ipch, /.vs
- Typical files project files that are required
 - *.sln, *.suo,
 - *.vcxproj, *.vcxproj.filters, *.vcxproj.user
 - *.cpp, *.h
 - CleanMe.bat

Standard Rules

Submit multiple times to Perforce

- Submit your work as you go to perforce several times (at least 5)
 - As soon as you get something working, submit to perforce
 - Have reasonable check-in comments
 - Seriously, I'm checking

Write all programs in cross-platform C++

- Optimize for execution speed and robustness
- Working code doesn't mean full credit

Submission Report

- Fill out the submission Report
 - No report, no grade

Code and project needs to compile and run

- Make sure that your program compiles and runs
 - Warning level 4
 - NO Warnings or ERRORS
 - Your code should be squeaky clean.
 - Code needs to work "as-is".
 - No modifications to files or deleting files necessary to compile or run.

- All your code must compile from perforce with no modifications.
 - Otherwise it's a 0, no exceptions

Project needs to run to completion

- If it crashes for any reason...
 - It will not be graded and you get a 0

Leave Project Settings

- Do NOT change the project or warning level
 - Any changing of level or suppression of warnings is an integrity issue

Leaking Memory

- If the program leaks memory
 - There is a deduction of 20% of grade
- If a class creates an object using new/malloc
 - It is responsible for its deletion
- Any **MEMORY** dynamically allocated that isn't freed up is **LEAKING**
 - Leaking is **HORRIBLE**, so you lose points

No Debug code or files disabled

- Make sure the program is returned to the original state
 - If you added debug code, please return to original state
- If you disabled file, you need to re-enable the files
 - All files must be active to get credit.
 - Better to lose points for unit tests than to disable and lose all points
- Disable your debug printing otherwise you will lose points

Due Dates

- See Piazza for due date and time
 - Submit program perforce in your student directory assignment supplied.
- Fill out your this **Submission Report** and commit to perforce
 - **ONLY** use Adobe Reader to fill out form, all others will be rejected.
 - Fill out the form and discussion for full credit
- Assignment See Piazza for **Date and Time**
 - Due in two weeks before 8th week class session on Thursday.
 - Grading synchronized to this time
 - ----- ABSOLUTELY NO EXTENSIONS -----
 - Date cannot be moved

Goals

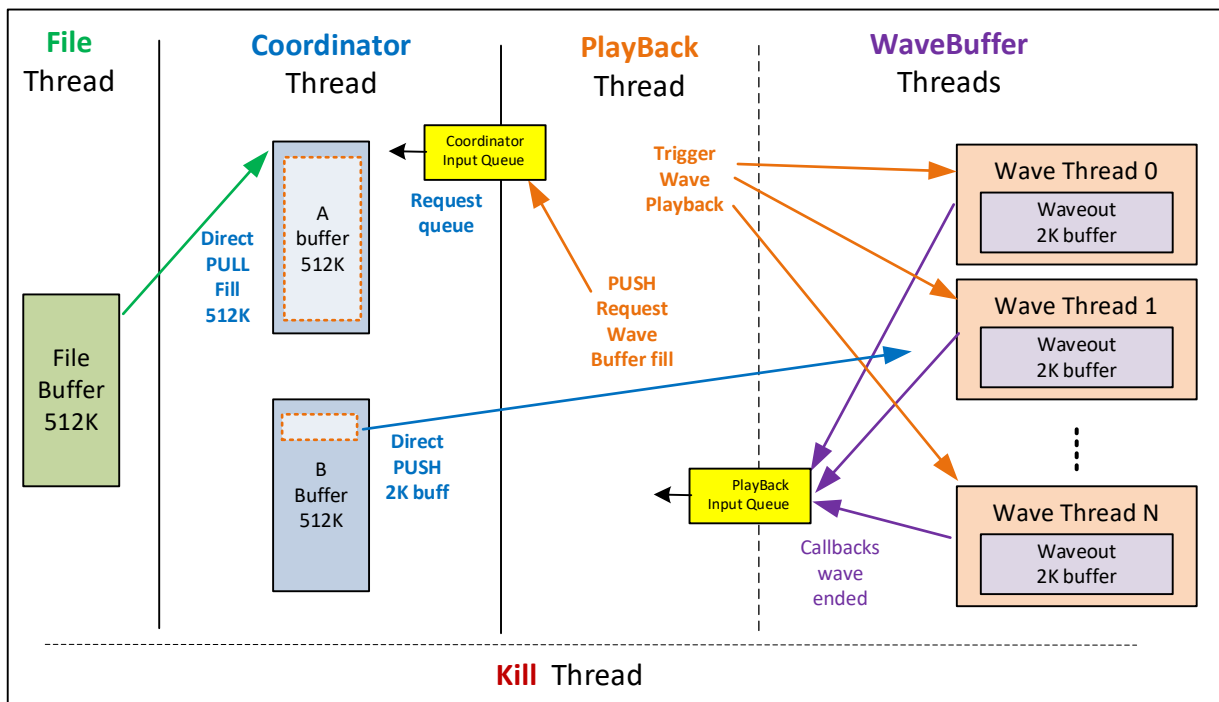
- Required:
 - Simplified Threading program - Using C++ 11 threading model
 - Program using basic multithreading primitives
 - Several fixed threads, Mutexes, Callbacks, Critical Sections
 - Material from Chapter 1-4 of book - only
 - No Atomics or advanced features allowed
 - Must follow the spirit of the problem
 - **Part-A** 10% - Grade
 - Do working in /student/**PA1_A**
 - File <-> Coordinator Thread Simulation
 - Video Demo - max 10 min
 - Design / code show and tell
 - Demo of the code working with prints
 - Show off the File <-> Coordinator requirements
 - Submit PA1 pdf with links (leave Part_B empty)
 - **Part-B** 20% - Grade
 - Do work in /student/**PA1_B**
 - Complete application working
 - Video Demo max 10 min
 - Code review...
 - no Music playing..
 - just design / code show and tell
 - show the code show off all the design requirements
 - Working Code
 - Will be compiled and tested
 - Checking for pops/clicks and clean shutdown in Debug/Release
 - 4-5 page pdf write-up
 - Submit PA1 pdf with links (leave Part_A section empty)

Assignments

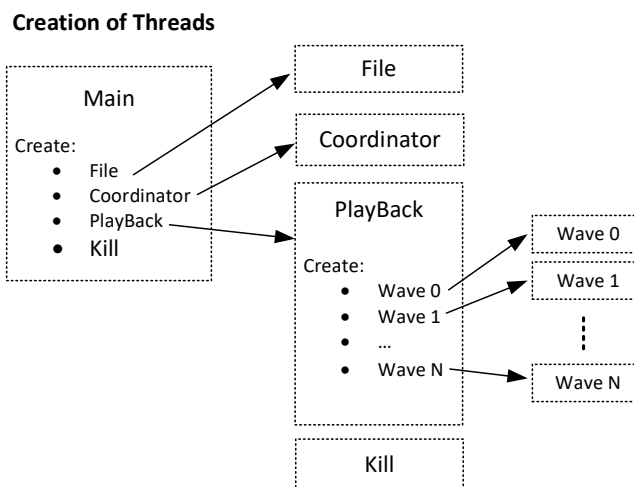
1. Summary

- a. Playback the Jetson's theme song using 5 large threads {**Main**, **File**, **Coordinator**, **PlayBack**, **Kill**} with 20 subordinate **WaveBuffer** threads plus the main thread.
 - i. 25 threads in total (Can be more)
 - **Main** Thread
 - a. All threads "context" classes created in **Main** thread
 - b. They are launched from main
 - i. Invoking the thread begin as a method call
 - **File**, **Coordinator**, **PlayBack**, **Kill**
 - 20 - **WaveBuffer** threads
 - ii. Once created they stay alive for the duration of the application
 - (until termination of the application)
 - **NO - KEY PRESS** allowed... should terminate at end of song
- b. Source wave data is raw wave data
 - i. Headerless, just raw data
 - ii. Broken into 23 separate files
 - wave_0.wav - wave_22.wav
 - Sizes of files vary between 512K to 128K bytes
 - iii. Sample rate: 22050, PCM, 16 bit, stereo wav file format
 - iv. Hard coding of names permitted, but not the file size
 - There is a pattern... easy to generate the names
- c. Goal:
 - i. Play the stereo sound track without any clicks or pops
 - ii. Should be seamless, playback should sound as if it plays
 - iii. Only sleep allowed is in the File Thread
 - iv. No spin locks... use the proper (async, condition variables, future, or promises)
- d. Communication
 - i. **Unlimited** use:
 - Mutex, Condition_Variables, Async, Future, Promise, Locks
 - anything from Chapter 1-4
 - ii. Only **one** timing sleep related allowed in the whole application
 - *sleep_for()*, *wait_for()*, *wait_until()* – for 200ms in **File** Thread ONLY
 - No extra sleep related allowed anywhere
 - a. If you need extra sleep your design is wrong - lose 50% of trade
 - iii. **Restricted** use (2 input Circular Queues)
 - **Coordinator** Thread: **Input Queue**
 - a. Request in order coming in **PlayBack** Thread

- b. Cannot be used for anything else... only request from the PlayBack thread
 - **PlayBack Thread: Input Queue**
 - a. Request coming in from WaveBuffer threads (20 of them)
 - b. Cannot be used for anything else... only request from the WaveBuffer threads through corresponding WaveBuffer callback functionality.



2. Thread creation spawning hierarchy



3. **File** thread
 - a. Single Buffer - 512K
 - b. Only one dynamic buffer allocation.
 - i. At beginning, then everything is out of that buffer
 - c. Some type of flag, living in the thread to communicate that it is filled
 - d. Need a mutex for access
 - i. Will sleep for **200 ms**, then check flag
 - e. Refill buffer with SYNCHRONOUS file loads using FileSlow library
 - i. FileSlow::Open(), Read(), Seek(), Tell(), Close()
 - ii. You cannot use any other file system
4. **Coordinator** Thread
 - a. Double buffer
 - i. A/B buffer or Front/Back buffer
 - b. Two separate buffers each 512K, dynamically allocated in beginning once
 - c. Two major roles of the coordinator thread:
 - i. Pull data mode
 - Retrieves data from the **File** thread, places in the A/B buffer
 - ii. Push data mode
 - Copies data from A/B to the wave buffer thread
 - Initiated by a trigger from the **PlayBack** thread
 - d. There is an input Circular Queue to the **Coordinator** thread
 - i. Only the **PlayBack** thread can feed this queue
5. **PlayBack** thread
 - a. General Description:
 - i. This thread keeps music playing between all of its wave buffers
 - ii. When a **WaveBuffer** thread is done playing
 - It sends a callback to the **PlayBack** thread
 - iii. This thread then points to the next Waveout buffer and continues playing
 - b. **PlayBack** thread cycles indefinitely between all the Waveout buffers in a continuous circular buffer fashion
 - i. Playing music without pops or clicks
 - c. Creates 20 **WaveBuffer** threads
 - i. Created in beginning
 - ii. Need to stay alive for the duration of the program
 - d. **PlayBack** thread communication
 - i. PlayBack gets message from waveout buffer threads
 - A callback from the waveout buffer player
 - Feeds the input CircularQueue to the **PlayBack** thread
 - a. Only the **WaveBuffer** callback can feed this queue
 - ii. **PlayBack** doesn't do the loading – it delegates
 - **Coordinator** thread refills the wave buffer threads

- **PlayBack** communicates to the **Coordinator** which buffer needs filling
 - a. It signals to coordinator to do the dirty business
 - e. In order to play sounds cleanly
 - i. We may need to have the wave buffers (20 of them) in individual threads
 - 2K buffers each
 - 20 separate (waveout) **WaveBuffer** threads
 - These threads will be created once, and then reused
6. Paper
- a. 4-5 page pdf paper
 - b. Necessary items to cover:
 - i. Description of the application
 - ii. Thread creation process
 - Who creates the threads
 - Names you use in code
 - Each thread responsibilities
 - iii. Communication between different threads
 - What is signaling, callbacks, mutexes, synchronization operations
 - iv. Complete Data movement from Reading to playback
 - Follow the data through the whole process to the actual playing
 - Diagrams please in your discussion
 - v. Challenges you had and what you learned
7. Videos
- a. See above for Part_A and Part_B video code/design discussion

Validation

Simple checklist to make sure that everything is submitted correctly

- Program compiles and runs without crashing?
 - Program warning free?
 - Memory leak free and clean closing
 - Severe deduction for leaking and shutting down properly
 - No Additional sleeping mechanism
 - Make sure program build without errors or level warnings
 - Project should be able to run without crashing
 - In Debug/Release
- Did you write your pdf file?
- Did you do the Videos? Part_A and Part_B

Hints

Most assignments will have hints in a section like this.

- Focus on small problems...(prototype for learning)
 - Focus on waveout playback: Single thread
 - Next add several buffers
 - Next separate into several threads
 - File thread
 - Mutex coordination with xxx ms sleep
 - Coordinator thread
 - A/B buffer switching
 - Loading
 - Queuing requests from playback thread
 - Playback thread
 - Triggering wave out buffer/threads
 - Callbacks
 - Circular indexing
 - Communicating to Coordinator thread
 - Synchronization primitives

Troubleshooting

- Baby steps
 - You'll be in trouble if you don't
- This is so slow and painful, takes forever to get working.
 - You cannot escape the agony of this part
 - Just do it.
- Hard to debug -> use `Debug::out()` for this project
- Have you BEEN studying and doing experiments over the last several weeks?
 - If not... this assignment is virtually impossible
 - If you have... PIECE of CAKE