THEATRE DESIGN & TECHNOLOGY MAGAZINE

1993 SUMMER ISSUE - SOUND COLUMN

MULTITASKING: DON'T TRY THIS AT HOME! (...or at the Office)

By Charlie Richmond

I recently went out of town for 12 days and returned to find about 125 new messages on "Computers in the Theatre" in an E-mail discussion forum I regularly attend. Since this subject is admittedly dear to my heart I plunged in, eager to read others' opinions. Discourse ran all the way from the old "Mac vs. IBM" warhorse to TD scheduling software, box office, and ticket programs. No discussions, however, focused on computers involved with the actual production or running the show.

This is easy to understand since the most familiar computer platforms would make extremely poor show or media controllers. To see "multitasking" defined, just watch any stage manager. The average computer user, however, has only recently become familiar with multitasking, however. When computers help run a show, everyone has to feel completely comfortable - with live control still ultimately in charge of everything. Only genuine preemtive multitasking software makes this possible.

Operating Systems and Multitasking

Now, new versions of old operating systems (or "OS" - the software which makes computers run) are touting "multitasking" - often in an abbreviated form allowing the user to easily access any one of several software applications. This way, various programs can be used concurrently (but not necessarily simultaneously) and applications can be chosen quickly. Popular examples are: MS-DOS 6.0, Windows 3.1 and Macintosh System 7.1.

Terms which signify such incomplete implementation - "message driven," "time slice" and "task swap" - are sometimes used with or instead of "multitasking" (see OS comparison chart). These are quite different from true (preemptive) multitasking where each program has its own set of tasks - each with its own priorities - and where the OS can actually make the decision what program and which task needs to be serviced based on a variety of needs.

These lesser OS's use "cooperative" multitasking - lacking the speed and realtime response that normally comes with a well designed preemtive OS. Almost all software written for older OS's, from which these newer variants have sprung, predates this cooperative functionality and therefore has no inherent multitasking ability. In this case, the newer OS must guess the needs of the software it is running and switch functions cooperatively - but not necessarily efficiently. By comparison, preemptive multitasking allows tasks to be serviced by the OS according to their needs, with or without human interaction.

More complex OS's incorporate preemptive multitasking via

multithreading - most effective in an expensive multiprocessor environment, in which individual "threads" or "parallel lightweight processes" are assigned to separate processors. When multithreading with a single processor, extra overhead for additional scheduling and context switching is incurred compared to a non-multithreading OS. Speed and realtime response suffers yet again. Popular examples are: Windows NT, OS/2 2.1, NeXTstep and UNIX.

In preemptive multitasking, any task may "forbid" other tasks to preempt it until it "permits" such preemption again. Also, each task is assigned a "task priority" value which indicates to the OS how important that task is to the program using it. Each application program uses any number of tasks, and many programs can be active simultaneously. Since each task is a separate entity, multiple programs can use a single task instead of multiple identical tasks. Sophisticated multitasking OS's have powerful messaging capabilities allowing these "library" tasks to be instantly accessible through standard system calls.

True preemptive multitasking OS's are hardly the most popular ones, even though they've been around for years. And, vice-versa, the least powerful multitaskers have become the most popular OS's. Why? Almost any programmer knows:

1. Few programs inherently require multitasking (show control is a notable exception)

2. Multitasking OS's have complex rules which must be followed even though multiple tasks may not be needed.

3. A more powerful multitasking OS uses more complex rules. Programs are harder to write as the OS becomes more complex.

4. Most application programs are developed for simpler OS's first because they:

A. are easier to write
B. are faster to write
C. can be marketed sooner and (potentially) cheaper
D. can be made reliable more easily
E. do not require more complex OS's

5. The average computer user sees no need to consider any but a simple OS because:

A. almost all applications are available
B. most applications appear there first
C. applications should be reliable and cheap
D. it's the most popular and best supported
E. return to 4) above and repeat until Microsoft wins

Multitasking Benefits

Perhaps the most popular application for a multitasking OS is the action video game. Most games, however, use multitasking sparingly since they usually accept only limited user input via

the mouse or joystick. This input is controlled by the game to a limited range of predictable moves at any time. In fact, action games often do not support a mouse because its input range is so much larger than the joystick. Since the mouse can point to 1/4 million or more pixels, the game must have an appropriate response for each. With such possibilities, the true multitasking OS provides the most efficient way to handle these choices quickly - in "real time" you might say.

Live computerized show control is similar to the live action video game - except the visual and aural result is experienced on stage rather than the monitors. But show control is much more complex and requires multitasking for the following reasons:

1. It must accommodate virtually any script - not just a specific game with fixed playing variables. To do this properly, powerful computerized show control has a dual hierarchy:

A. The show control program; providing the environment for:
B. The specific show and all its cues, programmed by the user

2. It must accommodate unpredictable external input and feedback, such as:

A. Keyboard, mouse and direct control devices
B. Response messages from controlled devices
C. Triggers from actors or moving pieces
D. Safety messages from error devices, proximity sensors, gas detectors, motion controls, E-stops, deadman switches, etc.
E. Internal messages from one cue list to another (or to itself)

3. It must always respond instantly, safely and logically.

4. When recorded music, video, film or other non-live media is used, the show controller must synchronize with them via perfect time keeping and/or time code while simultaneously controlling all live (asynchronous) elements.

Live Computer Show Control in Action

Basically, live show control software tracks the times at which cues for each discipline (lights, sound, flys, etc.) are called by the SM, then executes auto follows precisely as designed. But the SM must still be able to easily override chosen follows by skipping them, executing them sooner or later, or manually repeating or executing cues as desired. The SM has to be able to stop the entire show instantly - all elements or just selected ones - then restart it exactly from where it stopped. Or, with minimum delay, move to a new position and pick up from there. Software done right makes this possible with the only limitations often being the human capacity for management of monstrous complexities.

Such requirements are considerably more demanding than those of the average application program. First, the software usually obeys show programming. Timers run, cues go manually and

follows go automatically, keyboard and external controls are monitored and MIDI Show Control instructions are processed without delay - all without affecting prearranged sequence timing. Add to this the ability to do show and cue editing while the show is running and you have computer multitasking defined.

Compare this with desk top publishing, CAD, database or even MIDI sequencer and "multi-media" programs where the user almost exclusively initiates each process one at a time: When a redraw, zoom, search or song/multimedia playback (for example) is requested, the user - and the program - usually waits until it is done. All non-multitasking software - most programs - are written this way. There is no need to write them differently since they don't run live shows, where immediate response and realtime flexibility are required.

In summary, both games and show control must be realtime but the latter really requires faster response than the former, where all moves are predictable within a few possibilities. Realtime in the popular vernacular simply means fast. In fact, the entertainment industry sometimes uses realtime instead of "runtime," which means predictably following continuous time code and/or pre-programmed sequences. Runtime applications range from MIDI sequencers to digital audio workstations and film/video editing systems.

Realtime for live theatrical show control means that over 100 separate "stop watches" individually and simultaneously keep time during discontinuous sequences (starting/stopping/zeroing) and/or follow non-predictive and broken time code. At the same time, cues are fired manually, remotely or automatically - as fast as imaginable with nothing lost, no beats skipped, nothing delayed. "Realtime" really means "instantly and perfectly" - it is not easy and it needs true preemptive multitasking.

Unique Difficulties/Rewarding Benefits

In our own experience writing multitasking show control software, we have discovered it is not only difficult to do well, but that an efficient and intelligent preemptive multitasking OS is essential. The OS we use not only allows task prioritization but also statistically weights the time allowed various tasks according to their priorities and their "foreground/background" status.

For example, a high priority task (even the highest one, such as the system clock timer, a background task) can run constantly yet the OS still services low priority tasks promptly - provided they don't ask frequently. At the same time, if many low priority tasks all ask for service simultaneously - even if none have asked recently - the OS will never neglect the high priority tasks. All tasks are promptly dealt with since high priority tasks can interrupt low priority ones (unless 'forbid' is invoked). In fact, low priority tasks can easily interrupt high priority ones if necessary. Such is the incredible power and flexibility afforded by software-driven interrupts.

The "forbid" command typically prevents other tasks from accessing show and cue data while one task is changing it. Data integrity is guaranteed even though it can be altered by many tasks - such as editing, live/external cue list management and preset control value modification/override on the fly - even while the show is running and cues are going.

This all means these complex options must be thoroughly considered. Power and flexibility must not overshadow the need to make show programming easy and intuitive - and, most importantly, have the program work the way an experienced theatre professional would expect. The OS with this complexity, power and realtime capability is AmigaDOS - running more than 4 Million Amiga computers around the world.

We are frequently asked if we can provide similar functionality on other OS's. Correct responses are:

1) No - no other OS can do what this one can do.
2) Yes - we can do something similar but not as powerfully, completely or capably - is it worth it?
3) Maybe - we are always looking for programmers able to do miracles with more common OS's.

The third answer discloses our greatest frustration. If good live show control software could be created for computers that sit on most desks, widespread understanding would be much quicker. But bad software, readily accessible, only reinforces negative opinions - so we resist the temptation to produce "impaired" versions. Good programmers we know doubt that something as powerful can be done on another OS - at the very least, they feel it would be difficult and costly. Of course, Windows NT is the current rage and we will soon discover whether it is truly able to make amends for Windows 3.1's realtime multitasking shortcomings.

Postscript

A recent attempt to create similar show control software developed it in Unix on a high powered workstation - a system costing between $10000 and $20000. The project was begun with confidence but after more than two years, it was discovered that the system lost time while running the many tasks that had been written. An enormous amount was invested and, unbelievably, the entire Unix approach was scrapped. Ironically, Unix promises to soon be available with realtime capabilities.

Yet again, theatre pushes the high technology envelope to meet its needs - because live art is always more demanding than dead 'multimedia.'

All trademarks are acknowledged to be the property of their respective owners.

Copyright ©1993 Charlie Richmond