How the Process Manager Schedules Processes

Your application achieves control of how it receives processing time by using the **Event Manager** functions **WaitNextEvent** or **EventAvail**. By calling these **Event Manager** functions, your application agrees to yield the CPU. (Only at this well-defined time can your application be switched out.) Whenever your application calls one of these functions, the **Process Manager** checks the status of your process and takes the opportunity to schedule other processes.

> **Note:** Your application can also yield processing time to other processes as a result of calling other Toolbox routines that internally call **WaitNextEvent** or **EventAvail**. For example, your application can yield the CPU to other processes as a result of calling either of the **Apple Event Manager** functions **AESend** or **AEInteractWithUser**.

In general, if any events are pending for your application, it continues to receive processing time. When your application is the foreground process, it yields time to other processes in these situations: when the user wants to switch to another application or when no events are pending for your application. Your application can also choose to yield processing time to other processes when it is performing a lengthy operation.

A major switch occurs when the **Process Manager** switches the context of the foreground process with the context of a background process (including the A5 worlds and low-memory globals) and brings the background process to the front, sending the previous foreground process to the background.

When your application is the foreground process and the user chooses to work with another application (by clicking in a window of another application, for example), the **Process Manager** sends your application a suspend event. When your application receives a suspend event, it should prepare to suspend foreground processing, allowing the user to switch to the other application. For example, in response to the suspend event, your application should remove the highlighting from the controls of its frontmost window and take any other necessary actions. Your application is actually suspended the next time your application calls **WaitNextEvent** or **EventAvail** .

After your application receives the suspend event and calls **WaitNextEvent** or **EventAvail**, the **Process Manager** saves the context of your process, restores the context of the process to which the user is switching, and sends a resume event to that process.

In response to a resume event, your application should resume processing and start interacting with the user. For example, your application should highlight the controls of its frontmost window.

A major switch also occurs when the user hides the active application (by using the Application menu). In general, a major switch cannot occur when a modal dialog box is the frontmost window. However, a major switch can occur when a movable modal dialog box is the frontmost window.

A minor switch occurs when the **Process Manager** switches the context of a process to give time to a background process without bringing the background process to the front.

For example, a minor switch occurs when no events are pending in the event queue of the foreground process. In this situation, processes running in the background have an opportunity to execute when the foreground process calls **WaitNextEvent** or **EventAvail**. (If the foreground process has one or more events pending in the event queue, then the next event is returned and the foreground process again has sole access to the CPU.)

When an application is switched out in this way, the **Process Manager** saves the context of the current process, restores the context of the next background process scheduled to run, and sends the background process an event. At this time, the background process can receive either update, null, or high-level events.

A background process should not perform any task that would significantly limit the ability of the foreground process to respond quickly to the user. A background process should call **WaitNextEvent** often enough to let the foreground process be responsive to the user. Upon receiving an update event, the background process should update only the content of its windows. Upon receiving a null event, the background process can use the CPU to perform tasks that do not require significant amounts of processing time.

The next time the background process calls **WaitNextEvent** or **EventAvail**, the **Process Manager** saves the context of the background process and restores the context of the foreground process (if the foreground process is not waiting for a specified amount of time to expire before being scheduled again). The foreground process is then scheduled to execute. If no events are pending for the foreground process and it is waiting for a specified amount of time to expire, the **Process Manager** schedules the next background process to run. The **Process Manager** continues to manage the scheduling of processes in this manner.

In version 7.0, drivers and VBL tasks in the system heap are scheduled regardless of which application is currently executing. Drivers installed in an application's heap are not scheduled to run when the application is not executing.

See the **Compatibility Guidelines** and the **Event Manager** for specific information on how your application can handle suspend and resume events and how your application can take advantage of the cooperative multitasking environment.