SNS COLLEGE OF ENGINEERING

Kurumbapalayam(Po), Coimbatore – 641 107
Accredited by NAAC-UGC with 'A' Grade
Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

Department Of Artificial Intelligence and Data Science

Course Name – Operating Systems

II Year / IV Semester

Unit 1 - THREADS ISSUES

Threading Issues

The fork and exec system calls:

The fork() system call is used to create a separate duplicate process

When an Exec() is invoked , the program specified in the parameter to exec() will replace the entire process – including all threads

Issue :If one thread forks, is the entire process copied, or is the new process single-threaded?

Solution:

A: System dependant.

A: If the new process execs right away, there is no need to copy all the other threads. If it doesn't, then the entire process should be copied.

A: Many versions of UNIX provide multiple versions of the fork call for this purpose.

Which of the two versions of fork() to use depends on the application.

- If exec () is called immediately after forking

  Then duplicating all threads is unnecessary as the program specified in the parameters to exec() will replace the process.

In this instance duplicating only the calling thread is appropriate.

If the separate process does not call exec() after forking

- Then the separate process should duplicate all threads

Thread cancellation:

- Task of terminating a thread before it has completed

Threads that are no longer needed may be cancelled by another thread in one of two ways:

Asynchronous Cancellation cancels the thread immediately.

Deferred Cancellation sets a flag indicating the thread should cancel itself when it is convenient. It is then up to the cancelled thread to check this flag periodically and exit nicely when it sees the flag set.

(Shared) resource allocation and inter-thread data transfers can be problematic with asynchronous cancellation.

- Where the difficulty with cancellation lies:
- In situations where
- Resources have been allocated to a canceled thread
- A thread is cancelled while in the midst of updating data it is sharing with other threads
- Often the OS will reclaim system resources from a canceled thread but will not reclaim all resources
- Therefore canceling a thread aynchronously may not free a necessary system wide resource.

- With Deferred cancellation:

One thread indicates that a target thread is to be canceled

But cancellation occur only after the target thread has checked a flag to determine if it should be cancelled or not

This allows a thread to check whether it should be canceled at a point when it can be canceled safely.

Signal Handling

Q: When a multi-threaded process receives a signal, to what thread should that signal be delivered?

A: There are four major options:

- Deliver the signal to the thread to which the signal applies.
- Deliver the signal to every thread in the process.
- Deliver the signal to certain threads in the process.
- Assign a specific thread to receive all signals in a process.

The best choice may depend on which specific signal is involved.

UNIX allows individual threads to indicate which signals they are accepting and which they are ignoring. However the signal can only be delivered to one thread, which is generally the first thread that is accepting that particular signal.

UNIX provides two separate system calls, **kill(pid, signal)** and **pthread_kill(tid, signal)**, for delivering signals to processes or specific threads respectively.

Windows does not support signals, but they can be emulated using Asynchronous Procedure Calls (APCs). APCs are delivered to specific threads, not processes.

Thread-Local Storage (was Thread-Specific Data)

Most data is shared among threads, and this is one of the major benefits of using threads in the first place.

However sometimes threads need thread-specific data also.

Most major thread libraries (pThreads, Win32, Java) provide support for thread-specific data, known as **thread-local storage** or **TLS.** Note that this is more like static data than local variables,because it does not cease to exist when the function ends.