

# POSIX API Enhancements

PJ28623

PJ28765



(c) Copyright IBM Corporation 2002

Any references to future plans are for planning purposes only. IBM reserves the right to change those plans at its discretion. Any reliance on such a disclosure is solely at your own risk. IBM makes no commitment to provide additional information in the future.



# Areas targeted

- UNIX IPC (InterProcess Communication) - **PJ28623**
  - ▶ Semaphores
  - ▶ Message Queues
- Process Initiation, Dispatch - **PJ28765**
  - ▶ fork()
  - ▶ exec()



# Rationales

- **To make ports easier**
  - ▶ All functions in this set are typically "heavy use" API calls for industrial-strength UNIX applications.
- **Little or no value is seen to native TPF programs**
  - ▶ Corresponding native TPF calls, where they exist, are bound to be more efficient.
  - ▶ Many of these functions were implemented by "wrapping" existing TPF services.
- **To complete the IPC set**
  - ▶ Shared memory (shmem) was the first; semaphores and UNIX message queuing complete the set.



# Warnings

- UNIX places great value and accuracy on a structure known as IPC, which contains the user ID, group ID, PID, and permissions for objects described in this presentation.
  - See `sys/ipc.h` for details.
- TPF will return the IPC structure or accept it as an argument wherever called for; however, *the contents may not be entirely dependable*: TPF applications & systems tend not to implement POSIX-style user IDs.



# Semaphores

- Typically used as *mutexes between processes*, may define *count events* or *bit events*, too.
  - ▶ Sound familiar? (EVNTC, POSTC, WAITC)
  - ▶ TPF had semaphore-like facilities before UNIX even existed, UNIX semantics very different ... any and all processes that know the semaphore set ID may operate against it.
- UNIX semaphores
  - ▶ Have attributes of *permission* (by user ID)
    - Permissions may be tested, process need not sleep/wait if not available for reading or writing.
    - Track is kept of the last process (PID) to have altered any member of the semaphore set.
  - ▶ Are typically created in *sets*
    - 'arrays' of semaphore objects with at least one member, possibly more.
  - ▶ Are *accessed* in sets
    - Only one process may address a semaphore set at a time: others must sleep/wait.
  - ▶ Have *values*
    - May be copied into or from, incremented/decremented, bit-mask manipulated.



# Semaphore API Calls

- **semget()**

- ▶ Gets an existing semaphore set ID by its key value, or creates the set if not already extant

- **semop()**

- ▶ Performs operations against individual semaphore values, argument names the specific 'phore in the set to be operated upon, along with a "command code":
  - Copies into or from the semaphore via local buffer
  - Increments the semaphore value
  - Tests for read/alter permission on the semaphore, may optionally sleep/wait until available
  - semop() operations may be "undone" when the issuing process abandons the semaphore (see semctl()).

- **semctl()**

- ▶ Operate on the semaphore set
  - Copies into or from the entire set's values via local buffer
  - Change permissions on all semaphores, or values of some based on uid/gid
  - Destroy the semaphore set, awakening all sleeping/waiting processes
- ▶ Get PID of last process to alter an individual semaphore
- ▶ Get list of PIDs waiting on an individual semaphore.



# Message Queuing

- ***Not even a substitute for Websphere MQ (formerly MQSeries)***
  - ▶ UNIX msg queues not known outside CPC, but are not I-stream unique
  - ▶ Serious limitations to scalability
    - No single message may exceed 4K in size
    - Sum of all messages and queues may not exceed 96K
    - No more than 128 messages per queue may exist
    - No more than 64 queues may exist
  - ▶ All queues wiped out at IPL
    - In-memory structures only
  - ▶ Implemented because some UNIX ported code needs it
  - ▶ IPC structure (permissions) passed in and out of API (refer to *Warnings*, slide #4).
- Design/implementation recommendations:
  - ▶ Use Websphere MQ for MOM (message oriented middleware) purposes, particularly those where data must be shared between machines.
  - ▶ Use UNIX msg queues for simple queuing applications that don't need large numbers/quantities of data *and* where Websphere MQ not appropriate.



# Message Queuing API

- `msgget()`
  - ▶ Get msg queue ID, or create one with the specified ID if not present.
- `msgsnd()`
  - ▶ Put message on queue.
- `msgrcv()`
  - ▶ Take message off of queue, place it in buffer.
- `msgctl()`
  - ▶ Copy entire queue into local buffer, leaving queue in place.
  - ▶ Write one or more messages from local buffer into queue.
  - ▶ Delete queue.



# fork()

- Creates a partial copy of caller's address space into new process, launches new process
  - ▶ Most POSIX attributes of the parent copied
    - File descriptors
    - Heap & stack space
    - Environment variables.
  - ▶ Some POSIX attributes not copied/inherited
    - Timer values
    - Signal sets.
- **ECB (TPF-unique) data not inherited by child address space**
  - ▶ Data levels all unoccupied
  - ▶ EBW/EBX zeros
  - ▶ TO2 objects not inherited.
- Following *fork()* execution, both parent and child processes resume execution at NSI (*fork()* statement + 1).
  - ▶ Return code from *fork()* tells us whether this is the child (zero) or parent (nonzero) process.
    - Parent gets child PID as return code, may choose to wait on child completion based on its PID.



# exec()

- There is no *exec()* call: this is actually a **family** of functions that call a program
  - ▶ Differences between *exec* family functions are:
    - The way in which arguments and environment variables are passed to called program.
    - How the called program is located (absolute path or via the PATH environment variable).
  - ▶ File descriptors open in the *exec* caller remain open following the call, except for those altered with FD\_CLOEXEC.
  - ▶ All other POSIX attributes of the address space continue in force following *exec*.
- There is no return from *exec* when the call is successful!!
  - ▶ Only when there are error returns (*rc* = -1) is NSI executed.
    - Specific error condition identified in *errno*
  - ▶ In other words, the caller ECB/EVM executes in the new program and will exit to TPF if the new program issues a *return()* call.



# exec() API

## ■ **execl()**

- ▶ Absolute path to executable file must be named, args passed as a single string, caller's environment "inherited".

## ■ **execv()**

- ▶ Absolute path to executable file must be named, args passed as an array of pointers to type char, caller's environment "inherited".

## ■ **execle()**

- ▶ Absolute path to executable file must be named, args passed as a single string.
- ▶ Environment passed as an array of pointers to type const char, replaces caller's environment.

## ■ **execve()**

- ▶ Absolute path to executable file must be named, args passed as array of pointers to type char.
- ▶ Environment passed as an array of pointers to type const char, replaces caller's environment.

## ■ **execlp()**

- ▶ PATH searched for executable file, args passed as a single string, caller's environment "inherited".

## ■ **execvp()**

- ▶ PATH searched for executable file, args passed as a single string.
- ▶ Environment passed as an array of pointer to type const char, replaces caller's environment.

MQSeries, and Websphere are registered trademarks of International Business Machines Corporation in the United States, other countries, or both. UNIX is a registered trademark of The Open Group in the United States and other countries. Other company, product, and service names may be trademarks or service marks of others.

