Jim Fawcett
CSE775 – Distributed Objects
Spring 2012

LINUX PROCESSES & THREADS

References

- Advanced Programming in the Unix Environment, Stevens & Rago, Addison Wesley, Second Edition, 2005
- The Linux Programming Interface, Kerrisk, no starch press, 2010
- Programming with POSIX Threads, Butenhof, Addison Wesley, 1997

Linux Processes

- A Linux process is a kernel entity to which system resources are allocated to execute a program.
- A Process consists of:
  - User-space memory containing a program's code and variables and kernel data structures that hold information about the process
  - This includes ids associated with the process, virtual memory tables, table of open file descriptors, signal handling information, process resource state and limits, current working directory, …

Process Structure

- Text segment (sharable, read only)
  - Machine language program instructions
- Initialized data segment
  - Global and local static data that are initialized, read when the program is loaded
- Uninitialized data segment (not stored on disk)
  - Uninitialized global and static data, filled with zeros when loaded
- Stack
  - Dynamically allocated stack frames for each program scope.
- Heap
  - Area for dynamic memory allocations made by program.

Creating a Child Process

- There are two primary ways to create a child process:
  - Call fork() which creates a clone of the parent process. Usually there is distinct code for parent and for child. See the Fork example.
  - Call one of the exec() functions. These fork() but then purge the process of the parent code and data and load another program for execution. See Execl example.

Linux Threads

- A Linux thread has:
  - A process unique thread id
  - A set of register values
  - A stack
  - A scheduling priority and policy
  - A signal mask
  - An errno value
  - Thread specific data

Linux Threads

- Headers
  - #include <pthread.h>
- Create Thread
  - Int pthread_Create(
    pthread_t* restrict pTid,
    const pthread_attr_t* restrict pAttr,
    void* (*pRunfunc)(void*),
    void* restrict pArg
    );
  - Returns 0 if OK, error number on failure

Thread Creation

- Thread starts running function pointed to by pRunfunc with single argument *pArg
  - This, of course, could be a struct of arguments
- pTid points to thread id supplied by create
- pAttr points to thread attributes structure
  - Null implies default attributes
- Terminates when *pRunfunc completes.
- No guarantees whether creator or thread run first.

Thread Creation

- Linux creates a “thread” by cloning the parent thread’s process. That clone shares part of the parent’s execution context like memory and file descriptors.
- Each thread has its own stack.
- Two threads can share global data and anything passed to both in arg structures.

Thread Termination

- There are three ways a thread can terminate without ending its parent process:
  - Return from *pRunfunc. Return value is thread's exit code.
  - Thread can be canceled by another thread in same process:
    - Int pthread_cancel(pthread_t tid);
  - Call pthread_exit(void* returnValue)

Wait for Thread to Complete

- One thread, usually the parent, can wait for termination of a thread by calling:
  - int pthread_join(pthread_t tid, void** returnVal);

  The return value is 0 on success, otherwise a failure code.

That’s All Folks