Pthreads and Semaphores

This document describes the system calls to create and manage *threads* in the POSIX (Portable Operating System Interface) version of `Unix`. This calls are similar to those found in `Windows NT` and `Solaris`.

Threads [top]

Thread Type [top]

The thread header file to include is pthread.h. This header file contains the definition of a *type*, `pthread_t`. This type is basically an integer (On hawk, it is just defined as an `unsigned int`. Its use is as a *thread identifier*.

```
Example: Declare two variables t1, and t2 to hold thread id's:

        #include <pthread.h>

        pthread_t t1, t2;
```

Prototype function for a thread [top]

The system call that creates a thread is passed the name of a function in the program code which that thread will execute. The prototype of this function must be like this function declaration of `f`. That is, one parameter of type `void *` and a return type, also `void *`. This is not a great restriction since the parameter can be the *address* of any item. It does generally require that the function use a conversion (or cast) on the parameter before using it.

```
void * f(void *p);
```

If f expects p to be a pointer to a `double`, then f might be written like this:

```
void * f(void *p)
{
   double *dp = (double *) p;
   double y;

   y = *dp;
   // do something with the double value in y
   ...

   return NULL;
}
```

Thread Creation Suspended[top]

When a pthread is to be created, it is created with certain default attributes. These attributes can be changed, but the defaults will do for now.

However, sometimes the calling thread does not want the new thread to be READY to execute at creation. For example, the calling thread may want to create more threads first and do other initialization. Finally after all that is completed, then all the new threads could be made READY and thus elligible to execute. The default is that new threads are READY.

With pthreads this default behavior can be changed in a slightly awkward way. Each thread in its function should **initially suspend itself!** Then the creating thread can call pthread_continue(tid) to let each thread begin executing the rest of its function.

```
int pthread_suspend(pthread_t tid);

A thread would suspend itself by:

   pthread_t me = pthread_self();

   pthread_suspend( me );
```

Create a thread [top]

The system call to create a thread is:

```
      #include <pthread.h>

      int pthread_create(
         pthread_t *thread,
         const pthread_attr_t *attr,
         void *(*start_routine)(void *),
         void *arg
      );

where,

           thread    Pointer to the location where the created thread's ID
                     is to be returned.

           attr      Pointer to the thread attributes object describing the
                     characteristics of the created thread. If the value is
                     NULL, default attributes will be used.

           start_routine
                     Function to be executed by the newly created thread.

           arg       Parameter to be passed to the created thread's
                     start_routine.
```

The return value is 0 if successful.

Wait for a thread [top]

A thread that creates other threads can wait for any one of these threads to finish or for a particular one to finish by calling the following function.

```
      #include

      int pthread_join(
         pthread_t thread,
         void **value_ptr
      );

 PARAMETERS
           thread    Thread whose termination is awaited by the caller.

           value_ptr Pointer to the location where the exit status of thread
                     is returned. This parameter can be NULL.
```

Other Thread Calls [top]

A running thread can get its own thread id by calling `pthread_self()`. A thread can exit but not terminate any other thread in the process by calling pthread_exit():

```
void pthread_exit(void *status);
pthread_t pthread_self();
```

where `status` is either 0 or the address of a variable in which to store the exit status of the thread. This location should not be a local variable.

Unblock a suspended thread:

```
pthread_continue(pthread_t thrd);
```

Semaphores [top]

Semaphore type [top]

The semaphore.h header file contains the definition of a semaphore type, sem_t. This type is a structure with several data members. However, you will not directly use any of the data members. The only way to use a semaphore is through the two functions that provide atomic operations on the semaphore. You can use the semaphore type to declare a semaphore variable, but it is not properly initialized by this declaration. A separate system call is the only way to properly initialize the semaphore.

```
Example.
   #include <sys/semaphore.h>

    sem_t s;
```

Initializing a semaphore [top]

The system call to initialize a semaphore is `sem_init`.

```
      int sem_init(sem_t *sem, int pshared, unsigned int value);

where,

      sem       address of the declared semaphore
      pshared   should be 0 (not shared with threads in other processes)
      value     the desired initial value of the semaphore

Return          The return value is 0 if successful.
```

```
Example:

  #include <semaphore.h>

  sem_t s;
  if ( sem_init(&s, 0, 1) != 0 )
  {
     // Error: initialization failed
  }
```

Semaphore Operations [top]

The two operations on a properly initialized semaphore are `sem_wait` and sem_post:

```
int sem_wait(sem_t *sp);
int sem_post(sem_t *sp);

Example:

  sem_t s;

  // Assume sem_init has been called to initialize s.

  sem_wait(&s);
  ....

  sem_post(&s);
```