Threads

Lecture 10

Threads

- Threads are lightweight processes
- Each process can execute several threads
    - The threads execute <u>independently</u>
    - Threads <u>share</u> the global variables and OS resources
    - Each thread has its <u>own local variables</u> and follow its <u>own execution flow</u>

Threads

- In practice
  - Main program creates threads
    - By specifying an entry point function and an argument.
  - The main program and each created thread run independently.
    - They share global variables.
    - They do not share local variables.

Threads

- Example of functions to handle threads
  - Creation
  - Exit
  - Cancellation
  - Synchronization

Threads

- Example
  - Alternating threads
    - Creates 3 threads
    - Let the system execute them concurrently
    - Wait for them to finish at main

Threads

```
int
main ( )
{
    int   i;

    for (i = 0; i < 3; i++)
          create thread to execute function loop with parameter i

    wait for each thread to finish
}
```

Threads

```
void
loop (int n)
{
        int i;

        for (i = 0; i < 20; i++)
        {
                printf("Thread %d\n", n);
                sleep (1);
        }
}
```

Threads

- Need synchronization
- Solution:
  - Lock

Locking

- Sharing variables
  - Requires mutual exclusion
  - Lock/unlock to avoid a race condition
- Have one lock for each independent critical region

Locking

```
//No synchronization -->> BAD!!

int count = 0;

void *update ( )
{
        int    i;

        for (i = 0; i < 1000; i++)
        {
            count++;
        }
}
```

```
//Using lock

int count = 0;

void *update ( )
{
       int    i;

       for (i = 0; i < 1000; i++)
       {
             lock
             count++;
             unlock
     }
}
```

pthread library

- We will use the Linux pthread library
  - Main information
    - man pthread
  - There are man pages for specific functions
  - Include ".h" file as shown in the man page: #include <pthread.h>
  - Compile using -lpthread

pthread library

- Operations

```
#include <pthread.h>

pthread_t thread;

int pthread_create (
          pthread_t *thread,                          // thread id
          const pthread_attr_t *attr,                 // attributes
          void *(*start_routine)(void *),             // function
          void *arg);                                 // argument
```

pthread library

- Operations

```
void pthread_exit (void *value_ptr);

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

pthread library

- Example using pthreads
  - Alternating threads
    - Creates 3 threads
    - Let the system execute them concurrently
    - Wait for them to finish at main

pthread library

```
#include <stdio.h>
#include <pthread.h>

void *loopThread (void *arg);

int main()
{
    int          i;
    pthread_t    thr[3];

    for (i = 0; i < 3; i++)
        pthread_create (&thr[i], NULL, loopThread, (void *) i);

    for (i = 0; i < 3; i++)
        pthread_join (thr[i], NULL);
}
```

pthread library

```
void *
loopThread (void *arg)
{
        int i, j;
        int threadNo = (int) arg;

        for (i = 0; i < 20; i++)
        {
                printf("Thread %d\n", threadNo);
                sleep (1);
        }
}
```

pthread -- mutex lock

- Functions

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

INIT
int pthread_mutex_init (pthread_mutex_t * mutex,
                        const pthread_mutexattr_t * attr);
LOCK
int pthread_mutex_lock (pthread_mutex_t *mutex);

UNLOCK
int pthread_mutex_unlock (pthread_mutex_t *mutex);
```

pthread -- mutex lock

```
//No synchronization -->> BAD!!

int count = 0;

void *update ( )
{
        int     i;

        for (i = 0; i < 1000; i++)
        {
            count++;
        }
}
```

```
//Using mutex lock
//pthread_mutex_init called in main

int count = 0;
pthread_mutex_t mutex =
    PTHREAD_MUTEX_INITIALIZER;

void *update ( )
{
        int     i;

        for (i = 0; i < 1000; i++)
        {
             pthread_mutex_lock (&mutex);
             count++;
             pthread_mutex_unlock (&mutex);
    }
}
```

Threads -- Deadlocks

- Danger!
  - When one or more threads are waiting for one another and no thread can proceed
- Example
  - One thread is waiting for itself
  - Two threads are waiting for each other
  - Several threads are waiting for one another in a cycle

Thread Usage

- Splitting the computation
  - Tasks
    - Example: auto-saver
  - Data
    - Example: splitting the operation on an array
  - Work
    - Example: calculating a series