STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY

CS 492: Operating Systems

Thread Programming

Instructor: Iraklis Tsekourakis

Email: itsekour@stevens.edu

Pthread: Thread Management

Creating and Terminating Threads

- Routines:
  - **pthread_create (thread,attr,start_routine,arg)**
  - **pthread_exit (status)**
  - **pthread_cancel (thread)**
  - **pthread_attr_init (attr)**
  - **pthread_attr_destroy (attr)**

Pthread_create

- *pthread_create* creates a new thread and makes it executable. This routine can be called any number of times from anywhere within your code.

- Syntax: *int pthread_create(pthread_t * thread, pthread_attr_t * attr, void * (*start_routine)(void *), void * arg);*

Pthread_create

- Syntax: *int pthread_create(pthread_t * thread, pthread_attr_t * attr, void * (*start_routine)(void *), void * arg);*

- Arguments:
  - thread: Pthread_create returns the new thread ID
  - attr: set thread attributes.
  - start_routine: the routine that the thread will execute once it is created
  - arg: A single argument that may be passed to *start_routine*.
    - All arguments must be passed by reference and cast to (void *).
    - NULL may be used if no argument is to be passed.

Pthread_create

- Return value:
  - If successful, the *pthread_create()* function returns zero.
  - Otherwise, an error number is returned to indicate the error.

Pthread_exit

- *pthread_exit* is used to explicitly exit a thread.
- Syntax: void pthread_exit(void **value_ptr*);
- Cleanup: the pthread_exit() routine does not close files; any files opened inside the thread will remain open after the thread is terminated.

Example

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5
void *PrintHello(void *threadid)
{
   long tid;
   tid = (long)threadid;
   printf("Hello World! It's me,
thread #%ld!\n", tid);
pthread_exit(NULL);
}
```

```
int main (int argc, char *argv[])
 {
      pthread_t threads[NUM_THREADS];
      int rc;
      long t;
      for(t=0; t<NUM_THREADS; t++){
        printf("In main: creating thread %ld\n", t);
        rc = pthread_create(&threads[t], NULL,
PrintHello, (void *)t);
        if (rc){
          printf("ERROR; return code from
pthread_create() is %d\n", rc);
          exit(-1);
        }
     }
//pthread_exit(NULL);
}
```

Thread Argument Passing

- One Argument

```
long taskids[NUM_THREADS];

for(t=0; t<NUM_THREADS; t++)
 {
      taskids[t] = t;
      printf("Creating thread %ld\n", t);
      rc = pthread_create(&threads[t], NULL, PrintHello, (void *) taskids[t]);
      ...
 }
```

Thread Argument Passing (Cont.)

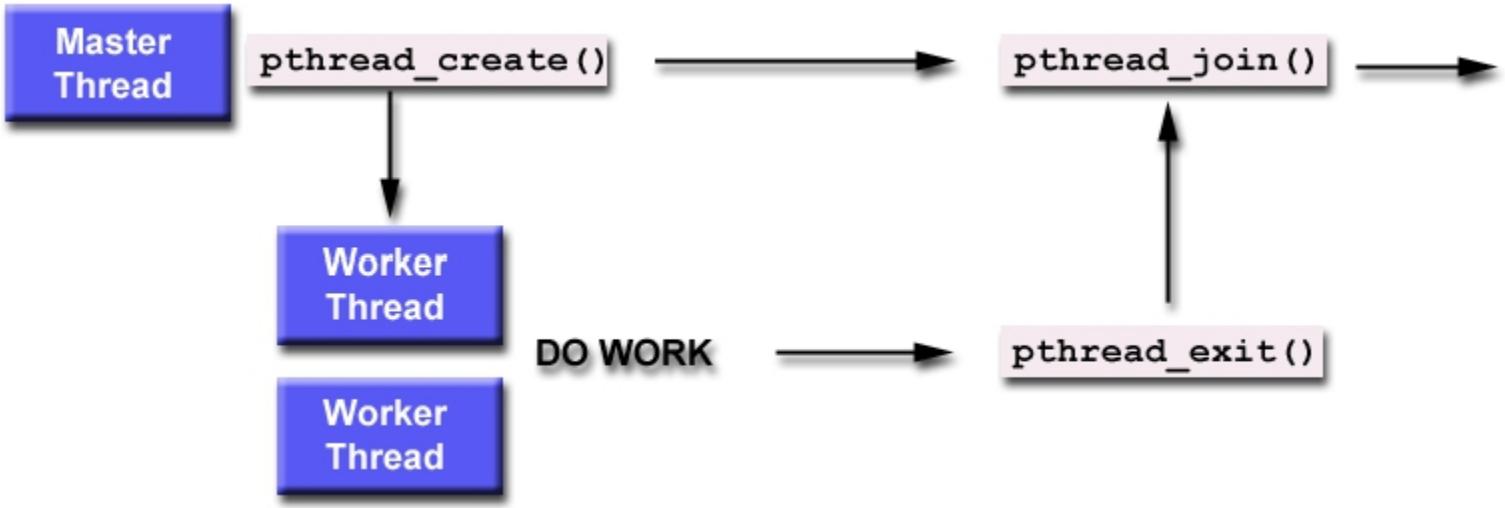
- More than One Argument

```
struct thread_data{
  int thread_id;
  int sum;
  char *message;
};
struct thread_data
thread_data_array[NUM_THREADS];
void *PrintHello(void *threadarg) {
   struct thread_data *my_data;
   ...
   my_data = (struct thread_data *) threadarg;
   taskid = my_data->thread_id;
   sum = my_data->sum;
   hello_msg = my_data->message; ...
}
```

```
int main (int argc, char *argv[]) {
      ...
      thread_data_array[t].thread_id = t;
      thread_data_array[t].sum = sum;
      thread_data_array[t].message =
messages[t];
      rc = pthread_create(&threads[t], NULL,
PrintHello, (void *) &thread_data_array[t]);
       ...
 }
```

Pthread_join

- "Joining" is one way to accomplish synchronization between threads



Pthread_join

- Syntax: int pthread_join(pthread_t *thread*, void ***value_ptr*);
- The **pthread_join**() function waits for the thread specified by *thread* to terminate. If that thread has already terminated, then **pthread_join**() returns immediately.

Pthread_join

- Syntax: int pthread_join(pthread_t *thread*, void ***value_ptr*);
- Arguments:
  - *thread* specifies the thread to be terminated.
  - *value_ptr*: If value_ptr is not NULL, then pthread_join() copies the exit status of the target thread (i.e., the value that the target thread supplied to pthread_exit(3)) into the location pointed to by * *value_ptr*.

Pthread_join

- Return values
  - If successful, the *pthread_join*() function shall return zero;
  - otherwise, an error number shall be returned to indicate the error.

Pthread_join Example

```
typedef struct str_thdata
{
   int thread_no;
   char message[100];
} thdata;
```

```
int main()
{
   pthread_t thread1, thread2;
  /* thread variables */
   thdata data1, data2;
  /* structs to be passed to threads */

   /* initialize data to pass to thread 1 */
   data1.thread_no = 1;
   strcpy(data1.message, "Hello!");

   /* initialize data to pass to thread 2 */
   data2.thread_no = 2;
   strcpy(data2.message, "Hi!");
```

```
   //Continued from main()
   /* create threads 1 and 2 */
   pthread_create (&thread1, NULL, (void *)
&print_message_function, (void *) &data1);
   pthread_create (&thread2, NULL, (void *)
&print_message_function, (void *) &data2);


   /* Main block now waits for both threads to
terminate, before it exits. If main block exits, both
threads exit, even if the threads have not
finished their work */
   pthread_join(thread1, NULL);
   pthread_join(thread2, NULL);
...
} /* main() */
```

More Resources

- Many good POSIX Thread programming tutorial websites
  - http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html
  - https://computing.llnl.gov/tutorials/pthreads/
  - And more..

Pthread: Mutex

Use of Mutex

- A typical sequence in the use of a mutex is as follows:
  - Create and initialize a mutex variable
  - Several threads attempt to lock the mutex
  - Only one succeeds and that thread owns the mutex
  - The owner thread performs some set of actions
  - The owner unlocks the mutex
  - Another thread acquires the mutex and repeats the process
  - Finally the mutex is destroyed

Creating and Destroying Mutexes

- Routines
  - **pthread_mutex_init (mutex,attr)**
  - **pthread_mutex_destroy (mutex)**
  - **pthread_mutexattr_init (attr)**
  - **pthread_mutexattr_destroy (attr)**

Creating Mutexes

- Mutex variables must be declared with type pthread_mutex_t
- Two ways to initialize a mutex variable:
  - Static:
    - pthread_mutex_t mymutex = PTHREAD_MUTEX_INITIALIZER;
  - Dynamic:
    - pthread_mutex_t mymutex = pthread_mutex_init();
    - This method permits setting mutex object attributes, *attr*.

Destroy Mutex

- Syntax: pthread_mutex_destroy (mutex)
- Destroy the mutex object referenced by *mutex*; the mutex object becomes uninitialized.
- It is safe to destroy an initialized mutex that is unlocked. **Attempting to destroy a locked mutex results in undefined behavior.**

Locking and Unlocking Mutexes

- Routine:
  - **pthread_mutex_lock (mutex)**
  - **pthread_mutex_trylock (mutex)**
  - **pthread_mutex_unlock (mutex)**

Pthread: Condition Variables

Condition Variables V.S. Mutex

- Both mutex and condition variables are for thread synchronization.
  - Mutexes implement synchronization by controlling thread access to data
  - Condition variables allow threads to synchronize based upon the actual value of data.

Condition Variables in Pthread

- Routines
  - **pthread_cond_init (condition,attr)**
  - **pthread_cond_destroy (condition)**
  - **pthread_condattr_init (attr)**
  - **pthread_condattr_destroy (attr)**

Creating Condition Variables

- Condition variables must be declared with type pthread_cond_t, and must be initialized before they can be used.
- *pthread_cond_init (condition,attr)*
  pthread_condattr_init (attr)
- Two ways to initialize a condition variable:
  - Static:
    - pthread_cond_t myconvar = PTHREAD_COND_INITIALIZER;
  - Dynamic:
    - pthread_cond_t myconvar = pthread_cond_init() routine.

Destroying Condition Variables

- pthread_cond_destroy (condition)
- pthread_condattr_destroy (attr)

Waiting and Signaling on Condition Variables

- pthread_cond_wait (condition, mutex): blocks the calling thread until the specified *condition* is signalled.
  - Should be called while *mutex* is locked, and it will automatically release the mutex while it waits.
- pthread_cond_signal (condition): signal another thread and wake it up
- pthread_cond_broadcast (condition) : signal multiple threads and wake all of them

Example: Mutex and Condition Variables

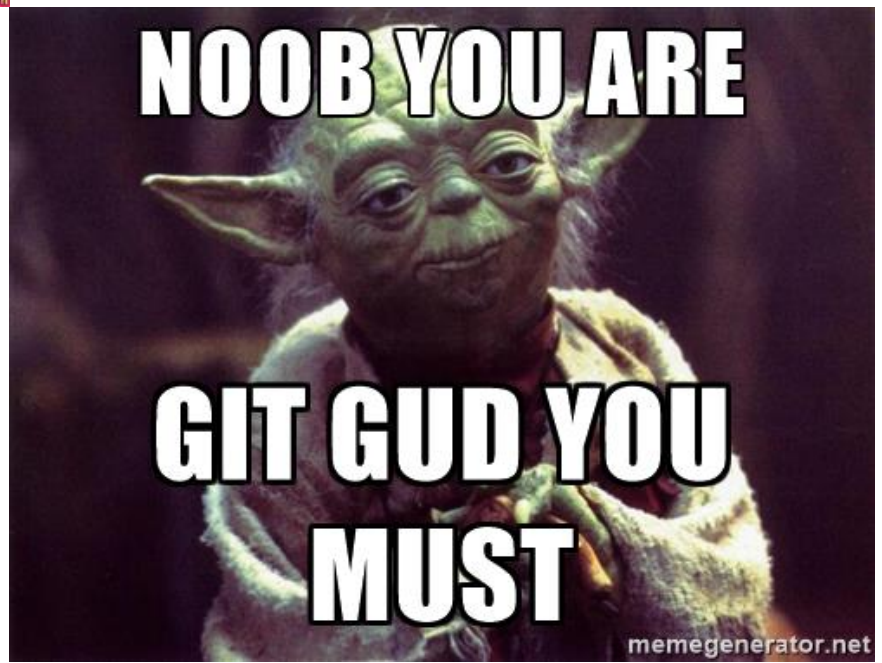
```
pthread_mutex_t count_lock;
pthread_cond_t count_nonzero;
unsigned count;
```

```
decrement_count()
{
    pthread_mutex_lock(&count_lock);
     while (count == 0)
          pthread_cond_wait(&count_nonzero,
&count_lock);
     count = count - 1;
     pthread_mutex_unlock(&count_lock);
}
```

```
increment_count()
 {
   pthread_mutex_lock(&count_lock);
   count = count + 1;
    if (count == 1)
      pthread_cond_signal(&count_nonzero);
   pthread_mutex_unlock(&count_lock);
 }
```

VERSION CONTROL


BRO
DO YOU EVEN GIT?
quickmeme.com


NOOB YOU ARE
GIT GUD YOU MUST
memegenerator.net

Version Control Systems

- Version control (or revision control, or source control) is all about managing multiple versions of documents, programs, web sites, etc.
  - Almost all "real" projects use some kind of version control
  - Essential for team projects, but also very useful for individual projects
- Some well-known version control systems are CVS, Subversion, Mercurial, and Git
  - CVS and Subversion use a "central" repository; users "check out" files, work on them, and "check them in"
  - Mercurial and Git treat all repositories as equal
- Distributed systems like Mercurial and Git are newer and are gradually replacing centralized systems like CVS and Subversion

Why Version Control?

- For working by yourself:
  - Gives you a “time machine” for going back to earlier versions
  - Gives you great support for different versions (standalone, web app, etc.) of the same basic project
- For working with others:
  - Greatly simplifies concurrent work, merging changes
- **For getting an internship or job:**
  - Any company with a clue uses some kind of version control
  - Companies without a clue are bad places to work
  - **If you haven’t already, create a github with samples of your code.**

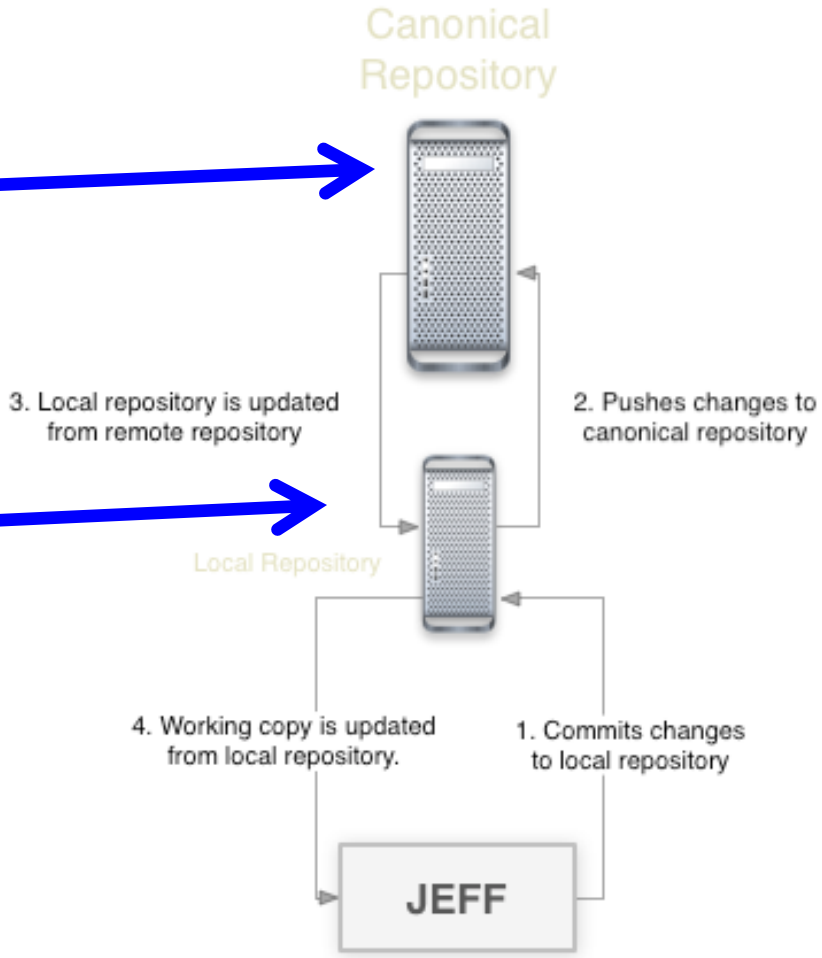
Why Git?

- Git has many advantages over earlier systems such as CVS and Subversion
  - More efficient, better workflow, etc.
  - See the literature for an extensive list of reasons
  - Of course, there are always those who disagree (we don't talk to them)
  - Most companies use Git
- Best competitor: Mercurial
  - Same concepts, slightly simpler to use
  - Much less popular than Git

Atlassian Bitbucket

Your repo in bitbucket

Your local repo



Typical Workflow

- `git pull` *`remote_repository`*
  - Get changes from a remote repository and merge them into your own repository
- `git status`
  - See what Git thinks is going on
  - Use this frequently!
- Work on your files (remember to add any new ones)
  - `git add *.c`
- `git commit –m` *“What I did”*
- `git push`
- MORE: https://try.github.io

Don’t forget!