CS 350 - Operating Systems **Winter 2018**

Lecture 2: January 8, 2018

Lecturer: Lesley Istead *Notes By: Harsh Mistry*

2.1 Threads and Concurrency

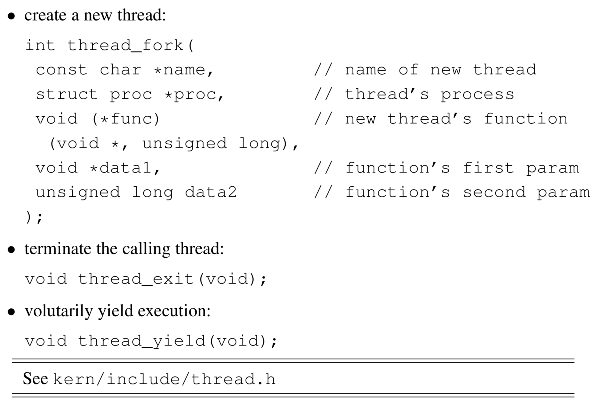
What is a Thread?

- Threads provide a way for programmers to express **concurrency** in a program
- A normal **sequential program consist of single thread of execution**
- In threaded concurrent programs there are multiple threads of execution, all occurring at the same time.

Key Ideas

- A thread can create new threads using `thread_fork`
- New threads can start execution in a function specified as a parmeter to `thread_fork`
- The original thread proceed concurrently, as two simultaneous sequential threads of execution
- All threads share access to the program's global variables and heap
- Each thread's function activations are private to the thread.

OS/161 Thread Interface



- create a new thread:

```
int thread_fork(
 const char *name,          // name of new thread
 struct proc *proc,         // thread's process
 void (*func)               // new thread's function
   (void *, unsigned long),
 void *data1,               // function's first param
 unsigned long data2        // function's second param
);
```

- terminate the calling thread:

```
void thread_exit(void);
```

- volutarily yield execution:

```
void thread_yield(void);
```

See `kern/include/thread.h`

Taken from lecture slides

Why Threads?

- parallelism exposed by threads enables parallel execution if the underlying hardware supports it. programs can run faster
- parallelism exposed by threads enables better processor utilization
  - if one thread has to block, another may be able to run