CSE320 System Fundamentals II Threads

YOUNGMIN KWON

Threads

A thread is a logical flow that runs in the context of a process

Each thread has its own thread context

- Thread ID (TID),
- Stack, stack pointer
- Program counter,
- General-purpose registers, Condition codes (Flags)

Threads

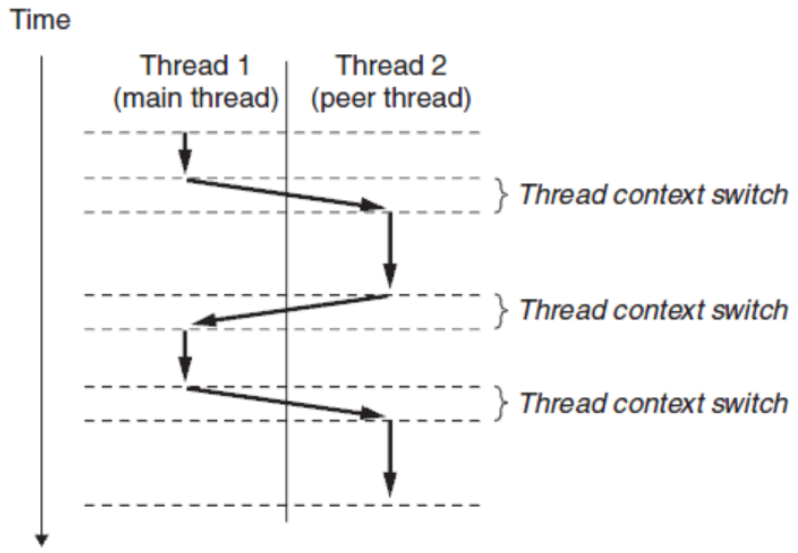
Multiple threads run in the context of a single process
- They share the entire virtual address space of the process
- Code, Heap, Shared libraries, Open files

Thread scheduling is done by the kernel
- Slow system calls like read or sleep
- Interrupted by the system timer

Threads

Each process begins life as a single thread called the main thread

Later, the main thread creates peer threads



Threads

A thread context is much smaller than a process context => context switch is faster

Threads associated with a process form a pool of peers

- No rigid parent-child hierarchy
- A thread can kill any of its peers
- A thread can wait for any of its peers to terminate
- Each peer can read/write the same shared data

POSIX Threads (Pthreads)

A standard interface for manipulating threads from C programs

Defines about 60 functions that allow programs

- to create, kill, and reap threads
- to share data safely with peer threads
- to notify peers about changes in the system state

Creating Threads

```c
#include <pthread.h>
typedef void *(func)(void *);

// To create a new thread
int pthread_create(pthread_t *tid,
                   pthread_attr_t *attr,
                   func *f,
                   void *arg);

// To get the thread id of its own
pthread_t pthread_self(void);
```

Terminating Threads

```
#include <pthread.h>

void pthread_exit(void *thread_return);
// Terminate explicitly
// If main thread calls pthread_exit,
// it will wait for all other peer threads to
// terminate, terminate itself, and terminate
// the process

int pthread_cancel(pthread_t tid);
// Terminate the thread with the ID tid
```

Reaping Terminated Threads

```
#include <pthread.h>
int pthread_join(pthread_t tid,
                 void **thread_return);

// - blocks until thread pid terminates,
// - update thread_return to point to the return
//   value of the thread routine,
// - reap the memory resource

// - unlike waitpid, there is no way to wait for
//   an arbitrary thread to terminate
```

Creating Threads

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

void *thread_func(void *vargp) {
    printf("%s\n", (char*)vargp);
    pthread_exit("world");
    //return "world";
    return NULL;
}
int main() {
    pthread_t tid;
    pthread_create(&tid, NULL, thread_func, "hello");

    void *ret;
    pthread_join(tid, &ret);
    printf("%s\n", (char*)ret);
    return 0;
}
```

Detaching Threads

```
#include <pthread.h>
int pthread_detach(pthread_t tid);

// - For a joinable thread (default) its
//   memory resource is not freed until the
//   thread is reaped

// - A detached thread cannot be reaped or
//   killed by other threads
// - Its memory resources are automatically
//   freed when it terminates
```

Initializing Threads

```
#include <pthread.h>
pthread_once once_control = PHTREAD_ONCE_INIT;
int pthread_once(pthread_once *once_control,
                 void (*init_routine)(void));

// - Useful to initialize shared global
//   variables dynamically
// - init_routine is called only once even if
//   pthread_once is called multiple times
```

```
//charcount.c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void* charcount(void *vargp);

int main() {
    pthread_t tid = pthread_self();
    printf("main: %u\n", (unsigned int)tid);

    while(1) {
        char str[100];
        pthread_t tid;
        scanf("%99s", str);
        if(strcmp(str, "quit") == 0)
            break;
        pthread_create(&tid, NULL, charcount, strdup(str));
    }
    return 0;
}
```

```
void* charcount(void *vargp) {
    char *str = (char*)vargp;
    int count[256] = {0,};
    pthread_t tid = pthread_self();
    int i;

    pthread_detach(tid);
    printf("server %u\n", (unsigned int)tid);
    for(i = 0; str[i]; i++)
        count[ str[i] ]++; // Counting the occurrence of each char
    for(i = 0; i < 256; i++) {
        if(count[i] > 0)
            printf("%u: '%c': %d\n", (unsigned int)tid, i, count[i]);
    }

    free(str);
    return NULL;
}

$ gcc charcount.c -pthread
```

Questions?