

Threads

CS241 Discussion Section

Week 3

2/05/07 – 2/09/07

Outline

- SMP2 Overview
- Pthreads
- Thread Safety

SMP2 Overview

SMP2 is an introduction to threads

Goal: sort a string in parallel using “enzymes”

Threads vs. Processes

What are the main differences?

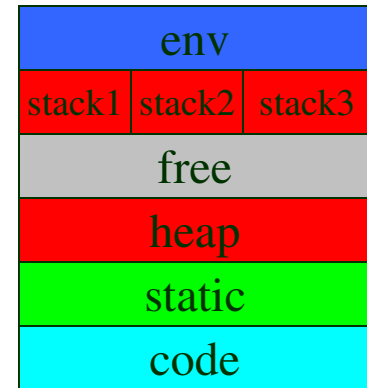
Memory / address space

Scheduling (concurrent execution)

Overhead

When should we use a process?

A thread?



POSIX Threads (Pthreads)

Standardized, portable thread API

To use POSIX thread functions

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

```
gcc -o main main.c -lpthread
```

Problem 1

Hello World! (thread edition)

See `p1-hello.c`

What's wrong?

Hint: how many threads are there?

Solution: Joining Threads

Remember zombie processes?

Child exited, but parent doesn't wait() for it

We can wait on threads, too, but we call it “joining”:

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

Detaching Threads

We have another option:

```
int pthread_detach (...);
```

Lets the system reclaim the thread's resources
after it terminates

Good practice – call `pthread_detach` or
`pthread_join` for every thread you
create

Exiting the process

We know what happens when main returns.

But what if one of the other threads calls `exit()`? Try it out.

Hello World!

Everything seems to work; “Hello World!” is printed out correctly.

But does it *always* work?

More on this later...

Aside: Function Pointers

Let's take a look at `pthread_create()`:

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

What is `start_routine`?

Good news: we can just give the function name for this argument.

Problem 2

See `p2-square.c`

What is the data type of `square`?

Create a variable `f` of this type.

Use it to call the `square` function, rather than calling `square()` directly.

Back to threads...

It is possible, in theory, for our program to print “World!” before “Hello”.

How can we fix this?

Passing Arguments to Threads

```
void *arg
```

Pointer to any data type

Have to cast it to a specific pointer type before dereferencing

Let's make `world_thread` wait on `hello_thread` (using `pthread_join`).

Thread Return Values

Threads return a `void*`, too. Return value can be retrieved by `pthread_join`

Be careful about not returning pointers to local variables!

Have each thread return a pointer to the string they print out.

Print these out again in `main()`.

Can threads have more than one argument?

Yes! Sort of.

We can pass a pointer to a struct, e.g.:

```
typedef struct {  
    int arg1;  
    char *arg2;  
} myargs;  
myargs a;  
pthread_create(..., myfunc, &a);
```

Aside: Review of structs in C

Keyword `struct` used to define complex data types.

Structs can contain variables, arrays, pointers, other structs...

Can structs contain pointers to functions?

Does that remind you of anything?

Problem 3: Remember linked lists?

Functions to create and print a linked list are provided (see p3-list.c)

Recreate the `list_node` struct

Now, create two threads to do this

Thread 1 calls `create_list`

Thread 2 calls `print_list`

Don't use any global variables!

Concurrency

Threads execute concurrently

True concurrency on multiple processors

Interleaving on a uniprocessor machine

All memory, except the stack, is shared
between the threads in a process

What happens if multiple threads access a shared
variable concurrently?

Problem 4: Modifying a shared variable

Write a program with global variable $x = 0$

One thread increments it N times ($x++$)

One thread decrements it N times ($x--$)

main() joins the threads and prints out x

Is the value of x always 0?

Run it a few times each with $N = 100, 1000, 1e5, 1e6, \dots$

What is going on?



Thread 1

Thread2

`x++;`

`x--;`

What is really going on

Thread 1		Thread2
read x	(100)	
Increment	(101)	
Context switch!		read x (100)
		Decrement (99)
		write x (99)
write x	(101) 	Context switch!

A few useful Pthreads functions

POSIX function	Description
pthread_create	create a thread
pthread_detach	set thread to release resources
pthread_equal	test two thread IDs for equality
pthread_exit	exit a thread without exiting process
pthread_join	wait for a thread
pthread_self	find out own thread ID