26. Concurrency: An Introduction

Operating System: Three Easy Pieces

Thread

- A new abstraction for a single running process

- Multi-threaded program
  - A multi-threaded program has more than one point of execution.
  - Multiple PCs (Program Counter)
  - They share the share the same address space.

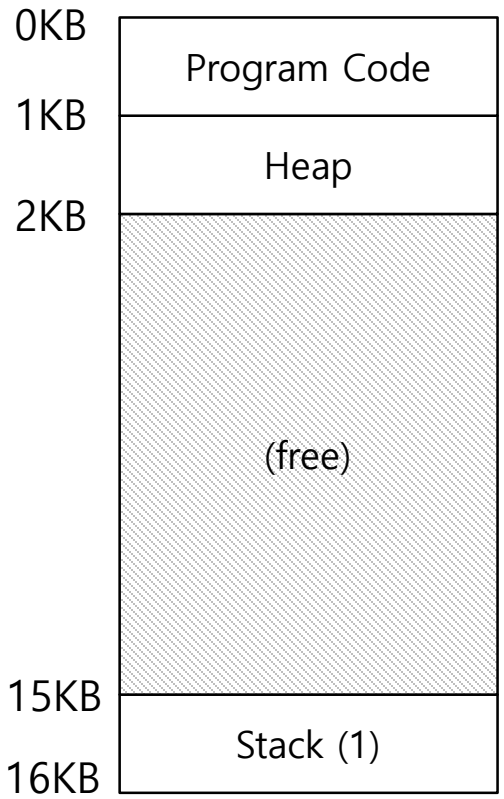
Context switch between threads

- Each thread has its own program counter and set of registers.
  - One or more **thread control blocks(TCBs)** are needed to store the state of each thread.

- When switching from running one (T1) to running the other (T2),
  - The register state of T1 be saved.
  - The register state of T2 restored.
  - The address space remains the same.

The stack of the relevant thread

- There will be one stack per thread.

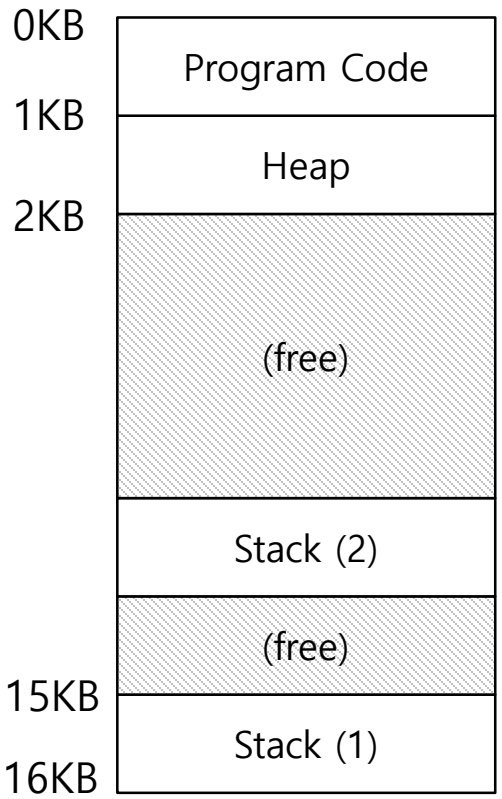


The code segment: where instructions live

The heap segment: contains malloc'd data dynamic data structures (it grows downward)

(it grows upward)
The stack segment: contains local variables arguments to routines, return values, etc.

A Single-Threaded Address Space



Two threaded Address Space

Race condition

- Example with two threads
  - counter = counter + 1 (default is 50)
  - We expect the result is 52. However,

OS	Thread1	Thread2	(after instruction) PC	%eax	counter
	before critical section		100	0	50
	mov 0x8049a1c, %eax		105	50	50
	add $0x1, %eax		108	51	50
interrupt					
save T1's state					
restore T2's state			100	0	50
		mov 0x8049a1c, %eax	105	50	50
		add $0x1, %eax	108	51	50
		mov %eax, 0x8049a1c	113	51	51
interrupt					
save T2's state					
restore T1's state			108	51	50
	mov %eax, 0x8049a1c		113	51	**51**

Critical section

- A piece of code that accesses a shared variable and must not be concurrently executed by more than one thread.
  - Multiple threads executing critical section can result in a race condition.
  - Need to support **atomicity** for critical sections (**mutual exclusion**)

Locks

- Ensure that any such critical section executes as if it were a single atomic instruction (**execute a series of instructions atomically**).

```
1    lock_t mutex;
2    . . .
3    lock(&mutex);
4    balance = balance + 1;
5    unlock(&mutex);
```

- Disclaimer: This lecture slide set was initially developed for Operating System course in Computer Science Dept. at Hanyang University. This lecture slide set is for OSTEP book written by Remzi and Andrea at University of Wisconsin.