

Multithreading Applications in Win32

(The Complete guide to Threads)

Seong Jong Choi
chois@mmlab.net

[Multimedia Lab.](#)

Dept. of Electrical and Computer Eng.
University of Seoul
Seoul, Korea

0. Contents (1/2)

PART I Threads in Action

1. Why You Need Multithreading
2. Getting a Feel for Threads
3. Hurry Up and Wait
4. Synchronization
5. Keeping Your Threads on Leash
6. Overlapped I/O, or Juggling Behind Your Back

PART II Multithreading Tools and Tricks

7. Data Consistency
8. Using the C Run-time Library
9. Using C++
10. Threads in MFC

0. Contents(2/2)

- 11. GDI /Windows Management
- 12. Debugging
- 13. InterProcess Communication
- 14. Building DLLs

PART III Multithreading in Real-World Application

- 15. Planning an Application
- 16. ISAPI
- 17. OLE, ActiveX, and COM

0. Main Objectives

- Simple definition

: Multiple threads allow a program to be divided into that can operate independently from each other.

process vs thread.

Multitasking vs Multithreading.

1 Overview

- This Chapter explains why multithreading is an important asset for both the developer and the end user.
- The root of problems in multitasking.
 - The meaning of terms like thread and context switch.
 - Race condition.

1.1 A Twisting, Winding Path(1/2)

- MS-DOS

- V 1.0 : no support for subdirectories, no batch language.

A running program got control of the entire machine.

Without a process model.(impassible to have even the concept of a multitasking or multithreaded OS)

- V 2.x : the installation of operating system extension (=TSRs)

TSRs:Terminate and Stay Resident.

- MS-DOS is not multitasking and definitely not multithreaded.

1.1 A Twisting, Winding Path(2/2)

- MS Windows

- V 1~3.x : allowed several applications to run at the same time, but it was the responsibility of each application to share the processor (Cooperative multitasking)
- Unix,VMS...:Preemptive multitasking.
- OS/2:memory protection,preemptive multitasking.

What is the difference between cooperative multitasking and preemptive multitasking?

- Windows NT to Rescue

- Full support for preemptive multitasking for 32-bit application.
- Win32s:a new set of API s.

1.2 Cooperative Threads

- The equivalent of cooperative multitasking, except that everything is happening within the same application.

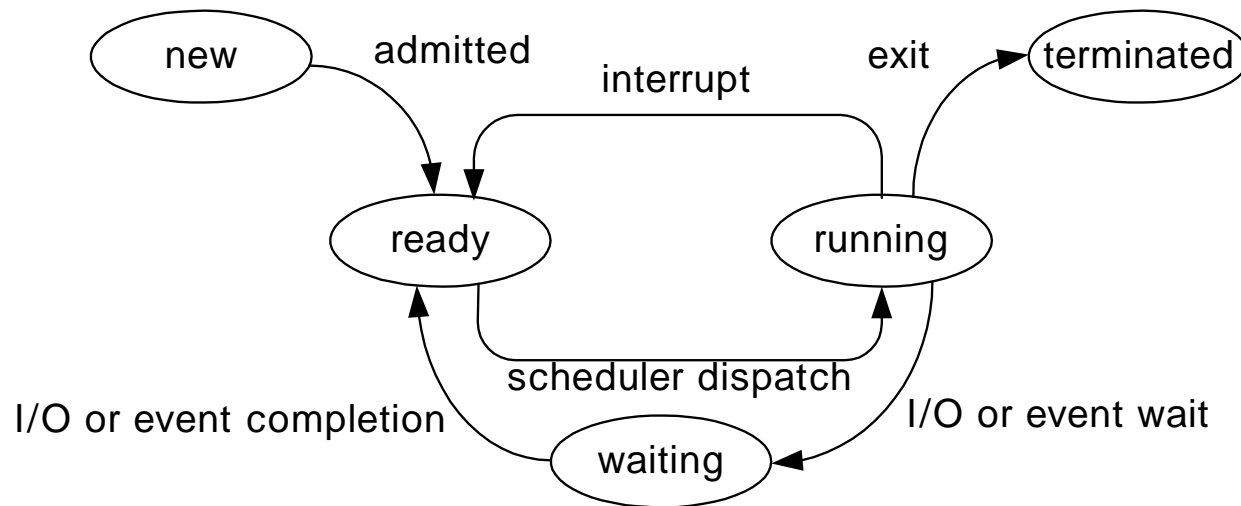
1.3 Why Users Want Threads

- Examples
 - Under Windows NT 3.x, using File Manager.
 - In the Windows 95, copy files from one to another. but Multithreading in Windows 95 is not perfect.
 - Using a CD-ROM (seeking time 200ms)

1.4 Win32 Basics(1/3)

- Processes

- Memory(2GB) + Resources(include kernel objects[file handles, threads])
- It provides a place for memory and threads to live.
- Diagram of process state



1.4 Win32 Basics(2/3)

- Memory
 - Three basic types:
 - The **Code** is executable part of the program.
 - The **Data** is where all of the variables in your program that are not local to a function are placed.
 - The **Stack** is your call stack and local variables.
- Threads
 - A single sequential flow of control within a program.

1.4 Win32 Basics(3/3)

- Why Not Use Multiple Processes?

- High cost. Threads are cheap.
- ...slower than using threads.
- Difficulty to pass handles between processes.

All thread can share window handles.(both the handles and the threads live in the same process.)

- non-windowed application.

Ex)Web server.(tremendous overhead)

1.5 Context Switching(1/4)

- What is a context?

-value of CPU registers, the process state, memory management information in the PCB of process.

Pointer

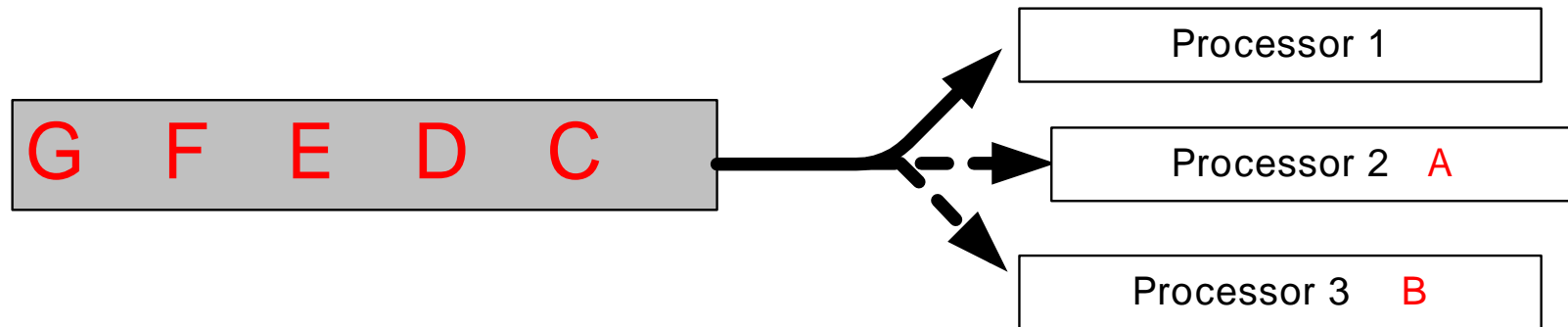
Pointer	Process State
Process number	
Program counter	
registers	
Memory limits	
List of open files	
.	
.	

1.5 Context Switching(2/4)

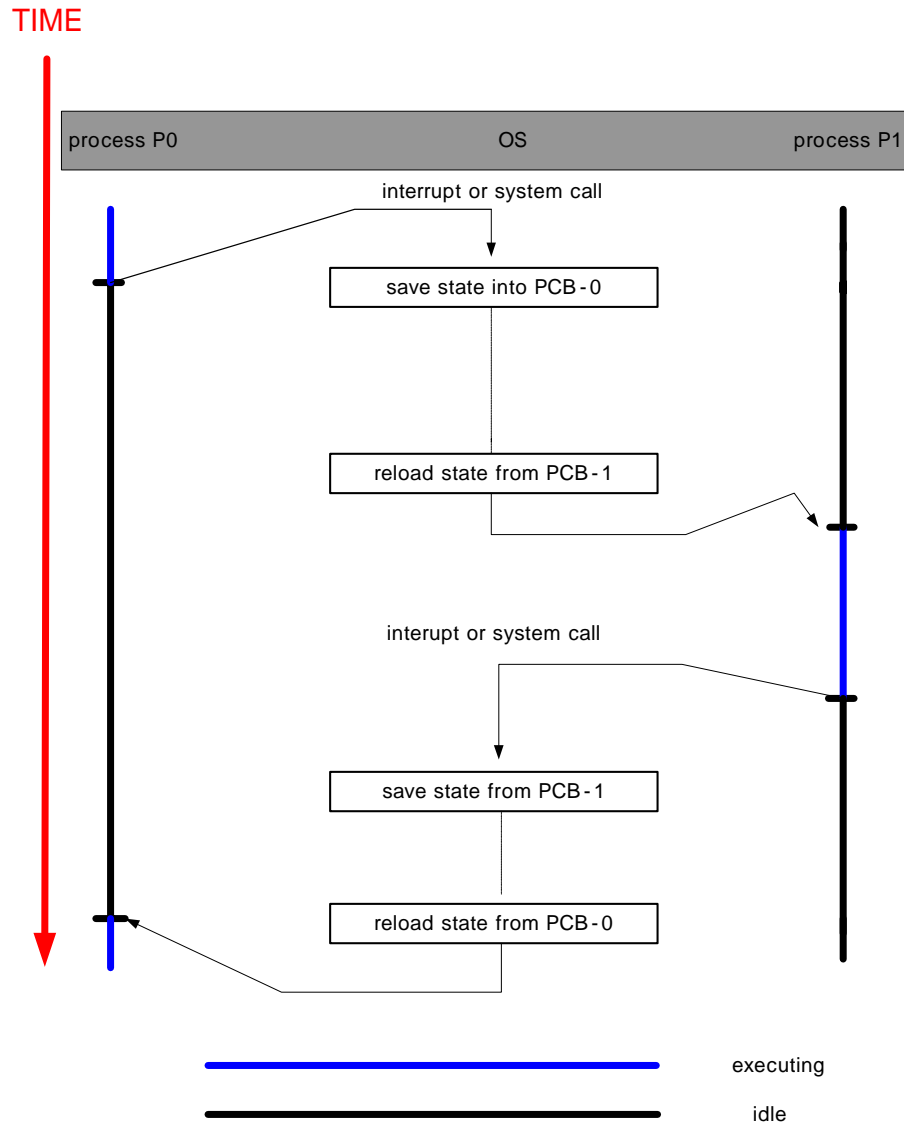
- Context switching: Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process.
- In preemptively multitasked system, the operating system **saves the current state** of the thread by **copying the thread's registers** from the stack into a **CONTEXT structure** and saving the structure for later use.
- To Switch to a different thread, the OS points the processor at the memory of the thread's process, then **restores the registers** that were saved in the CONTEXT structure of a new thread.
- Thread in the same process share all of their memory.(for communication)

1.5 Context Switching(3/4)

- Context switch Performance.
 - There is a small performance penalty paid for each context switch.(about time-see the next)
- SMP(Symmetric Multi Processing)



1.5 Context Switching(4/4)



1.6 Race Conditions(1)

- In a preemptive system, the order of multiple threads becomes **unpredictable**. (Race condition)
- The Linked List Example (see the next)

1.6 Race Conditions: Example Linked List

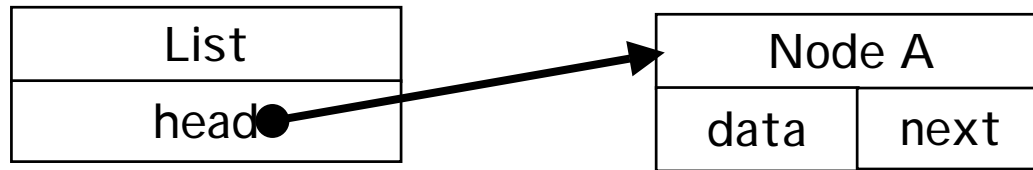
```
Struct Node{
    struct Node *next;
    int data;
};
```

```
Struct List {
    struct Node *head;
};
```

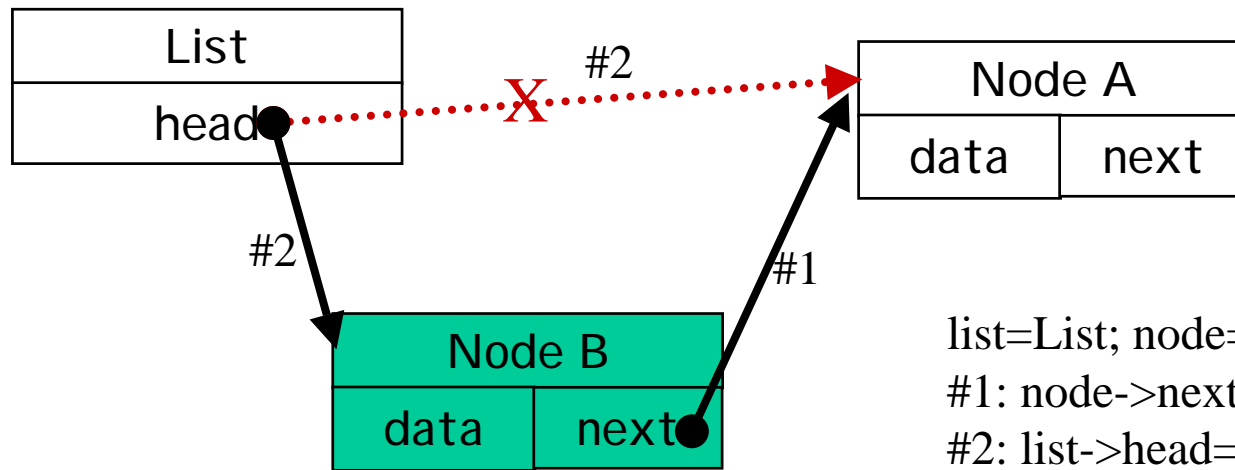
//Adding a node to the head of a list

```
void AddHead(struct List *list, struct Node *node)
{
    node->next = list -> head; //step #1
    list->head = node; //step #2
}
```

Linked List & Steps for Adding a Node



Initial Configuration



```
list=List; node=Node B;  
#1: node->next=list->head;  
#2: list->head=node;
```

Steps for Adding Node B

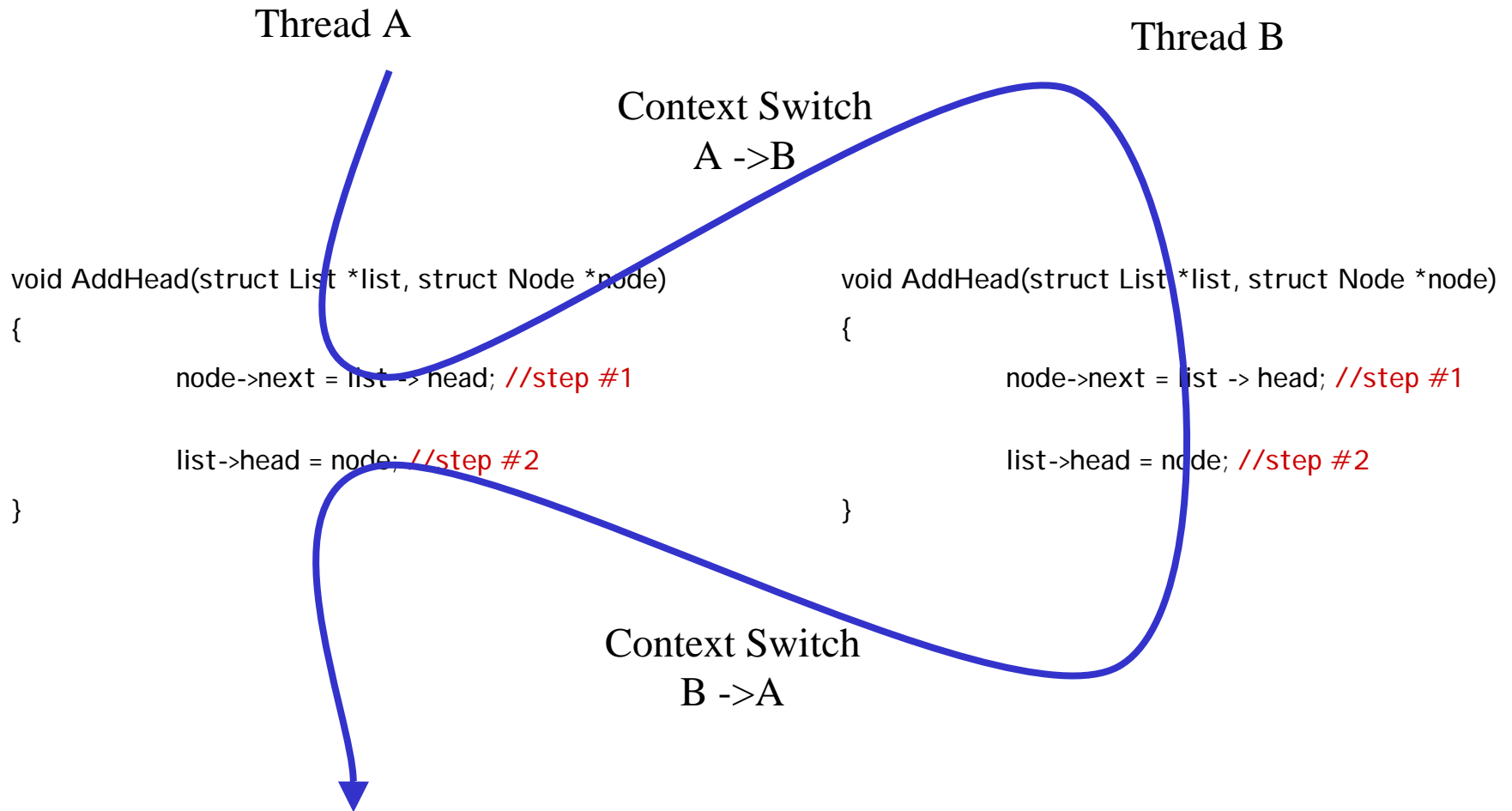
Scenario

Two thread, A and B, are executing AddHead().

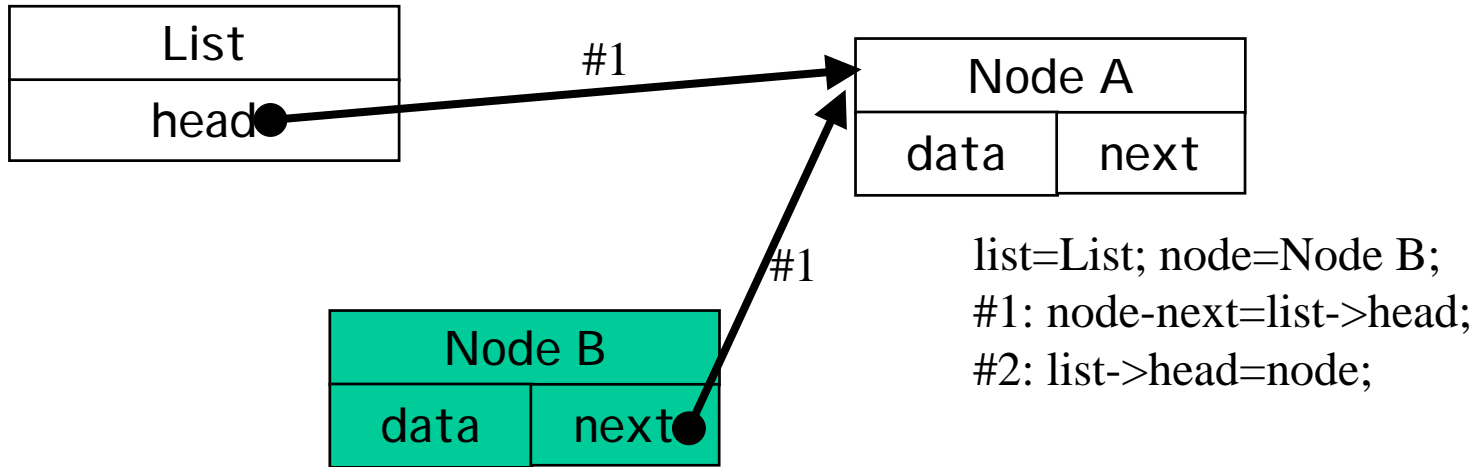
1. Initially, Thread A is running.
2. Thread A finishes step #1 of AddHead().
3. Context switch and Thread B is running.
4. Thread B finishes AddHead().
5. Context switch and Thread A is running.
6. Thread A finishes step #2.

See the next diagram...

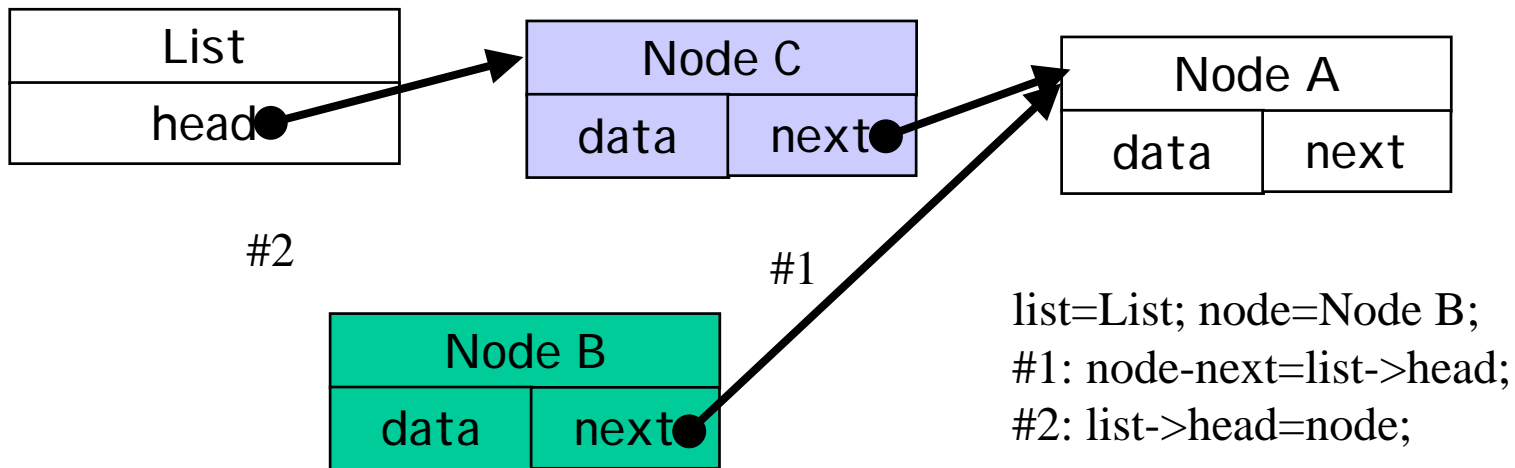
Scenario



1.6 Race Conditions(4/4)

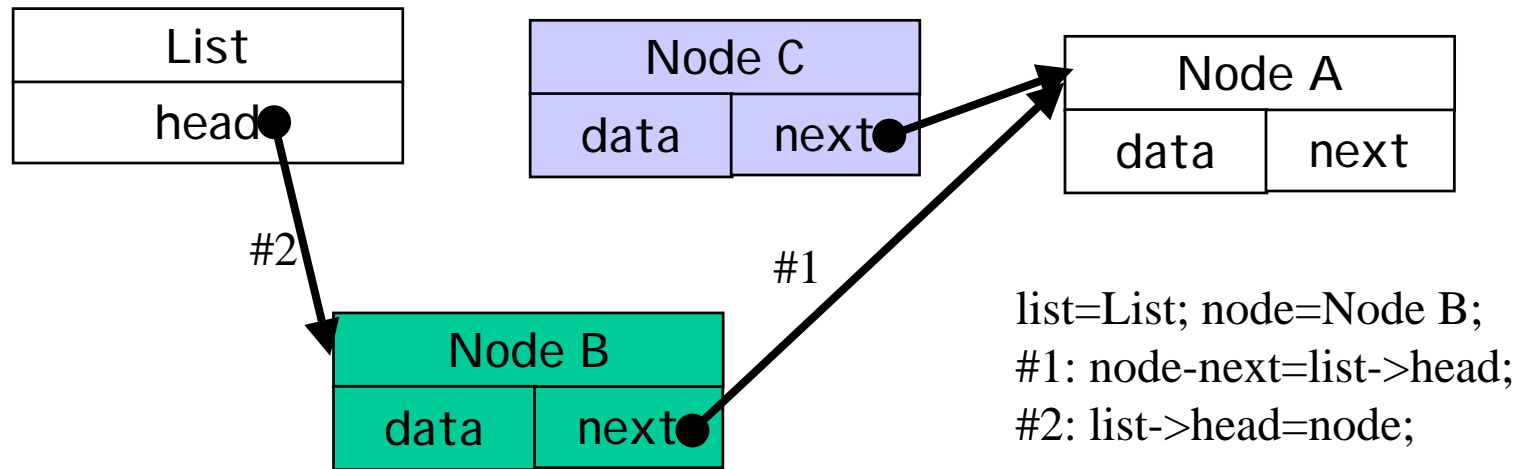


Linked list after step 1 of thread A



context switch and thread B adds Node C: completes step 1 & 2

1.6 Race Conditions(4/4)



Context switch and completes step 2

- What is this?????

1.7 Atomic Operations

- An operation that will complete without ever being interrupted is called an **atomic** operation.

- Linked list example

```
int flag;
```

```
AddHead(struct List * list , struct Node * node)
```

```
{
```

```
    while(flag != 0)
```

```
        ;
```

```
    flag = 1;
```

```
    next = list node -> -> head;
```

```
    List -> head = node;
```

```
    Flag = 0;
```

```
}
```

- Test and Set instruction.-flag.

1.8 How Threads Communicate

- An obvious solution is to use global data since all threads can read and write it.
- Communicating between threads can be quite tricky.-In PART II of this book.

1.9 Good News/Bad News

- Unless very carefully designed, multithreaded programs tend to be unpredictable, hard to test, and hard to get right.
- Multiple threads working on exactly the same thing : there is a high probability that something will go wrong unless the tasks are very carefully coordinated.
- Multiple threads working on separate tasks : they will need to share many things.
- Need very carefully plan!!!