

Multithreading Applications in Win32

(Introduction to Operating Systems)

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 APIs.

1.2 Cooperative Threads

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

1.3 Why Users Want Threads

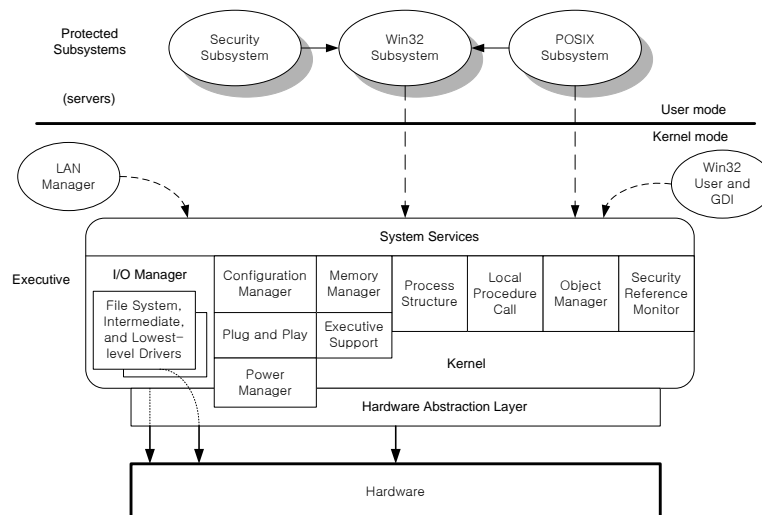
- Examples
 - Under Windows NT 3.x, using File Manager.
 - In the Windows 95, copy files from one to another. but Multithreading in Windows 95 is not perfect.
 - Using a CD-ROM (seeking time 200ms)

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-9

MS Windows OS Components



2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-10

Kernel and HAL

- Kernel
 - schedules threads for execution.
 - transfers control to handler routines when interrupts and exception occur.
 - performs low-level multiprocessor synchronization.
 - implements system recovery procedures after a power failure occurs.
 - Ex. of kernel routines: KeAcquireSpinLock, KeSetPriorityThread
- HAL
 - exports routines that abstract platform-specific hardware details about caches, I/O buses, interrupts, device registers, etc.
 - provides an interface between the platform's hardware and the system software.
 - Ex. of HAL routines: HalGetBusData, WRITE_PORT_UCHAR, HalGetInterruptVector, HalTranslateBusAddress

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-11

Process and Thread

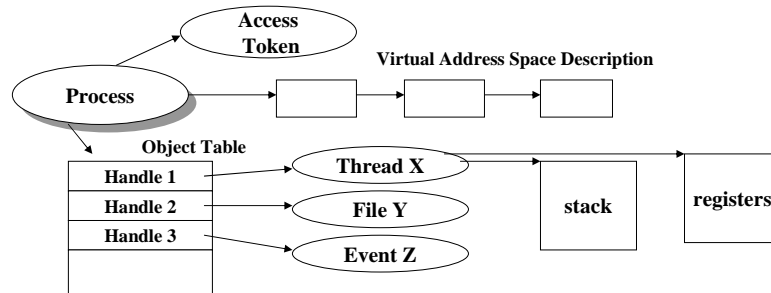
- Process is an environment which includes
 - a virtual address space,
 - executable code,
 - data,
 - object handles,
 - environment variables,
 - a base priority,
 - and minimum and maximum working set sizes.
- Thread is an execution which includes stack and its own copy of registers.
- Process can have multiple threads,
 - so, synchronization methods between threads are required.
 - Ex. of synch. methods : event, semaphore, mutant, timer objects

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-12

Process and Thread Structure



2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-13

Thread

- Thread Context
- the thread's set of machine registers,
- the kernel stack,
- a thread environment block,
- and a user stack in the address space of the thread's process

2006-09-06

Seong Jong Choi

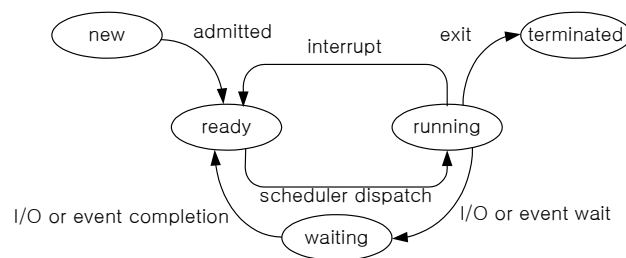
Multithreading CHAPTER 1-14

Primary Thread

- Each process is started with a single thread, often called the *primary thread*, but can create additional threads from any of its threads.

Thread States

- Thread State Transition Diagram



1.5 Context Switching(1/4)

- Process Context

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

Pointer

Pointer	Proce ss State
Process number	
Program counter	
registers	
Memory limits	
List of open files	
· ·	

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-17

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)

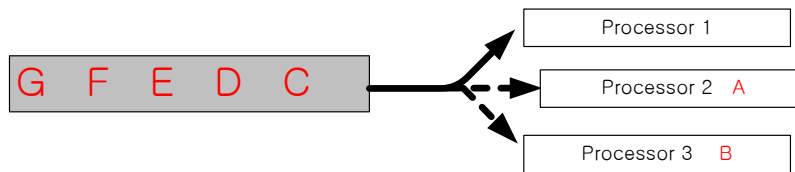
2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-18

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)

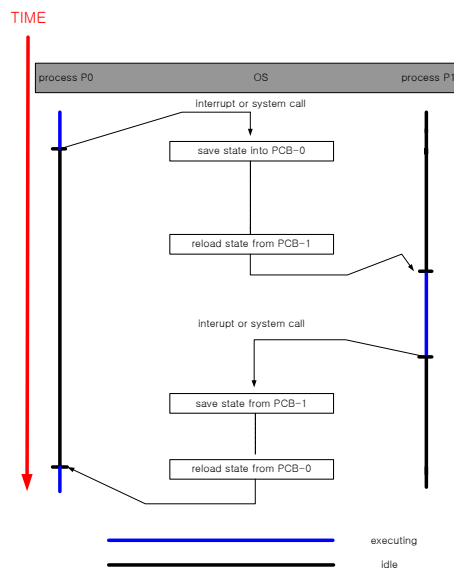


2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-19

1.5 Context Switching(4/4)



2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-20

Execution Context

- **Exceptions**
 - **user-mode thread**가 커널 서비스를 요청하는 경우, **exception**이 발생하여 **kernel-mode**로 진입한다.
 - **exception**을 발생시킨 **thread**의 **context**에서 수행된다.
- **Interrupts**
 - **hardware or software interrupts**가 발생하는 경우에 이를 처리하기 위해 **kernel-mode**로 진입한다.
 - 어느 **thread**의 **context**에서 수행되고 있는지 추측 불가능.
- **Kernel-Mode Threads(System Threads, or Device Driver Worker Threads)**
 - This thread runs exclusively in kernel mode, so it has no user-mode context and cannot access user address space.
 - **device driver**는 자체 동작을 위해 **kernel-mode thread** 생성가능.
 - NT Kernel 자체도 자신이 생성한 **kernel-mode thread**에 의해 수행됨.

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-21

Process vs. Thread

- **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)

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-22

Virtual Memory Management

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

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-23

Virtual Memory Management

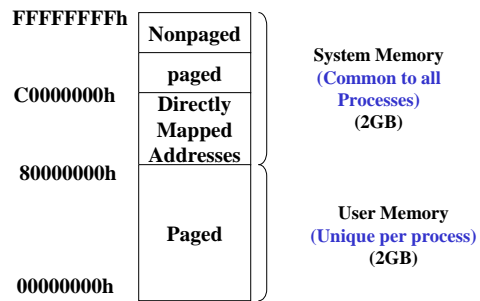
- provides user process with 4G bytes linear address space.
- protects and enables sharing between process's address space.
- paging mechanism : page table, page frame
- non-paged pool :
 - a portion of system memory which is never paged to disk
- paged pool :
 - can be paged to disk

2006-09-06

Seong Jong Choi

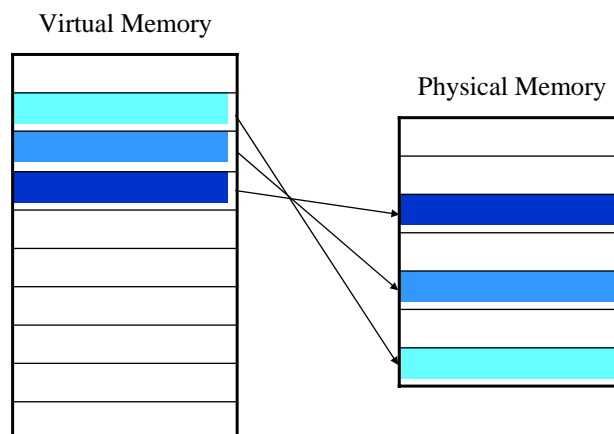
Multithreading CHAPTER 1-24

Virtual Memory Management



Virtual Address Space

Memory Mapping



Virtual Address Space Layout

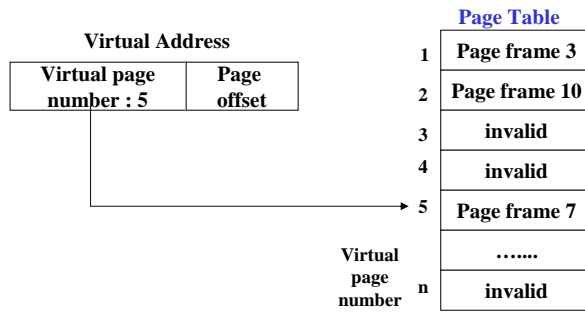
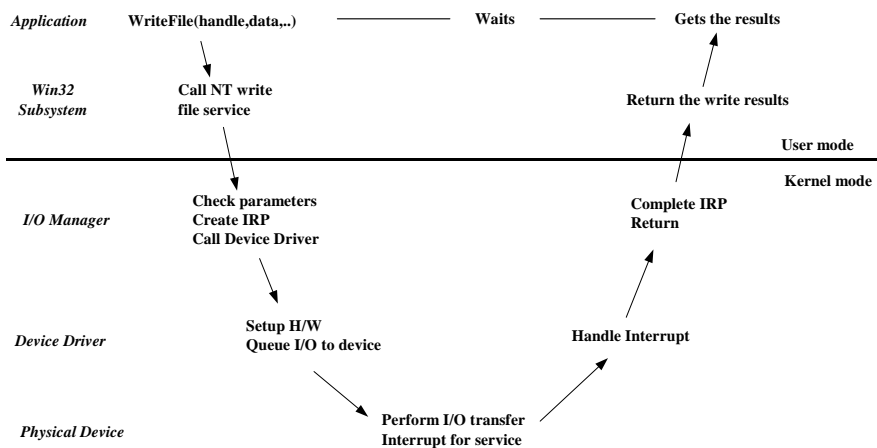
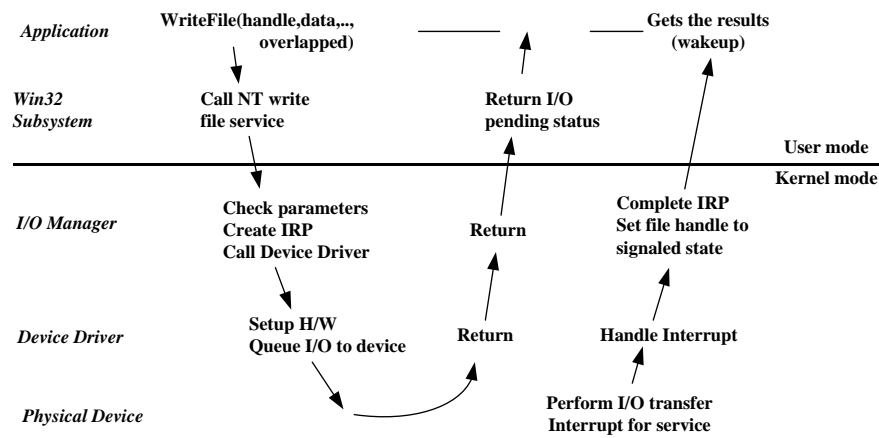


Fig 7. Conceptual Page Table

Synchronous I/O Processing



Asynchronous I/O Processing



2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-29

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)

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-30

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)
//Note: Call by pointer
{
    node->next = list -> head; //step #1
    list->head = node; //step #2
}
```

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-31

Linked List & Steps for Adding a Node



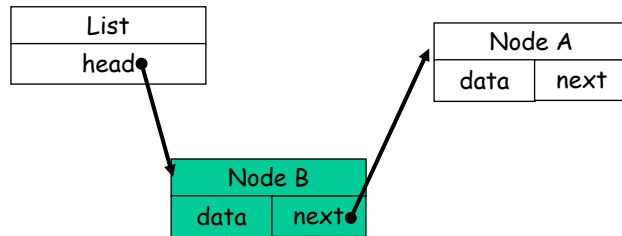
Initial Configuration

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-32

Linked List & Steps for Adding a Node



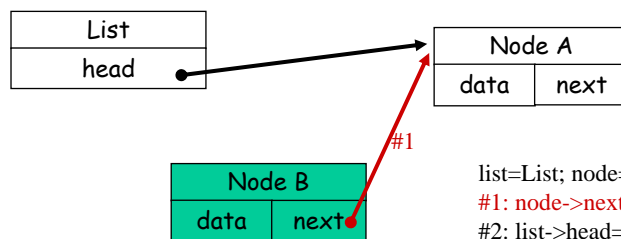
After adding Adding Node B in front of the list

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-33

Linked List & Steps for Adding a Node



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

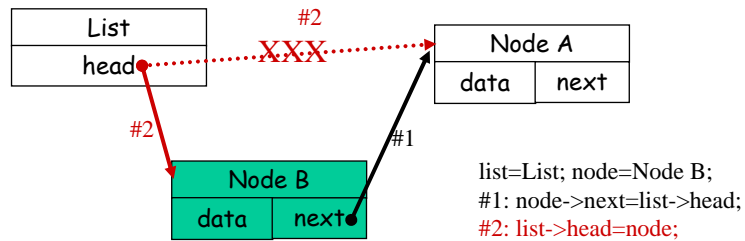
Step #1 for Adding Node B

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-34

Linked List & Steps for Adding a Node



Steps #2 for Adding Node B

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-35

Scenario 1 – Normal Operation

- Two thread, 1 and 2, are executing AddHead() on a list
 - Initially, the list has Node A
 - Thread 1 wants to add Node B
 - Thread 2 wants to add Node C

1. Initially, Thread 1 is running.
2. Thread 1 finishes AddHead().
3. Context switch and Thread 2 is running.
4. Thread 2 finishes AddHead().

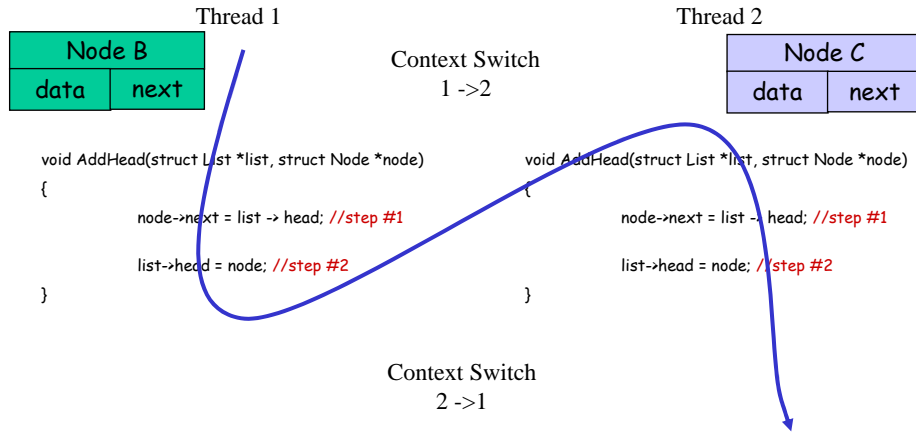
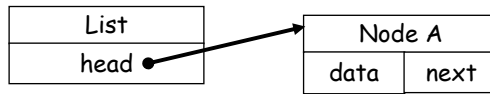
See the next diagram...

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-36

Scenario 1

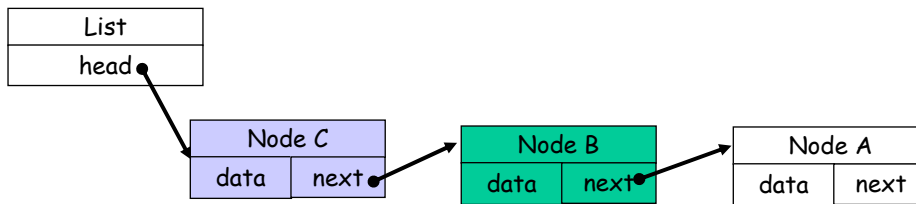


2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-37

Scenario 1



After the operation

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-38

Scenario 2: Race Condition

- Two thread, 1 and 2, are executing AddHead() on a list
 - Initially, the list has Node A
 - Thread 1 wants to add Node B
 - Thread 2 wants to add Node C
1. Initially, Thread 1 is running.
 2. Thread 1 finishes step #1 of AddHead().
 3. Context switch and Thread 2 is running.
 4. Thread 2 finishes AddHead().
 5. Context switch and Thread 1 is running.
 6. Thread 1 finishes step #2.

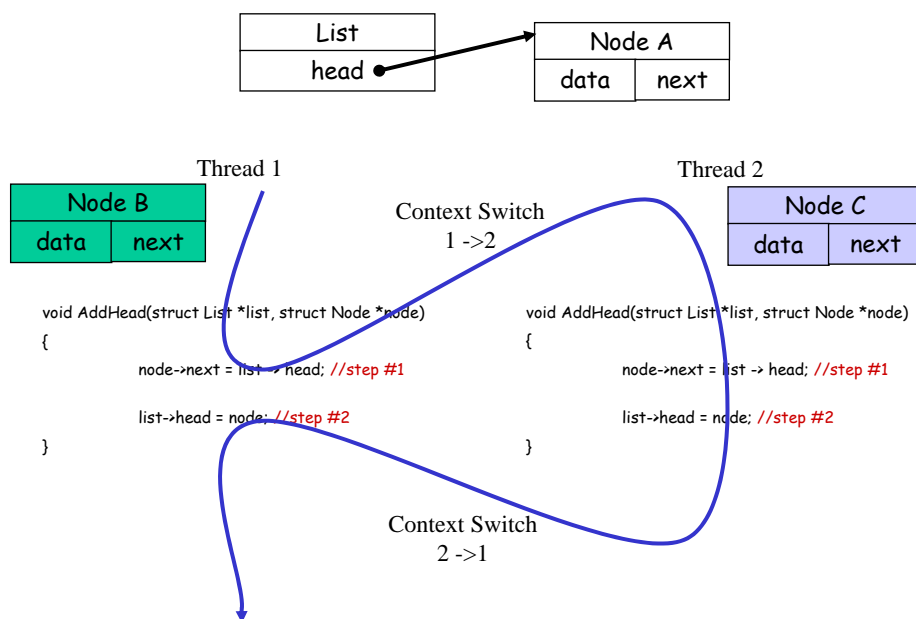
See the next diagram...

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-39

Scenario

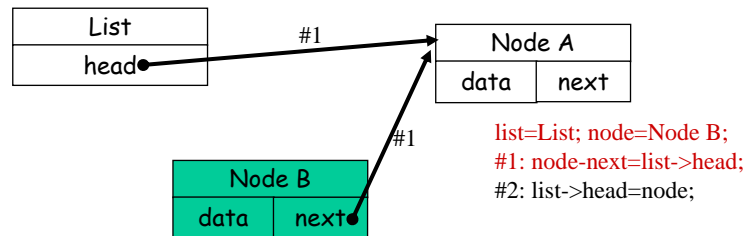


2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-40

Race Conditions



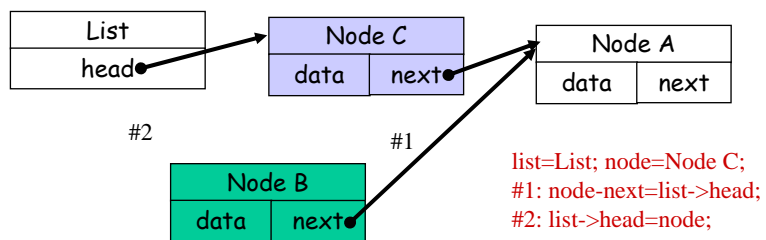
Linked list after step #1 of thread 1

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-41

Race Conditions



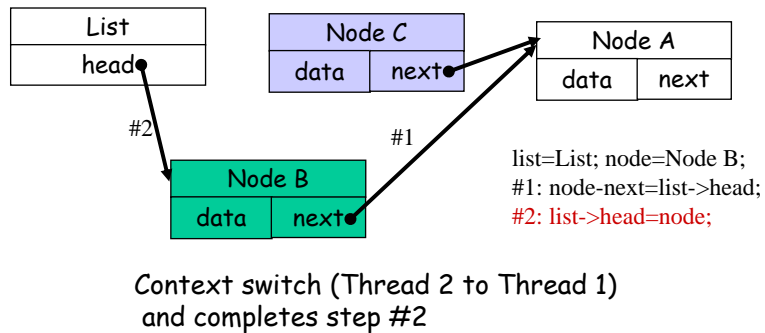
context switch (thread 1 to thread 2)
and thread 2 adds Node C: completes step 1 & 2

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-42

Race Conditions



- What is this????
- We created a monster!!!!

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-43

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.

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-44

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!!!

MS VC++

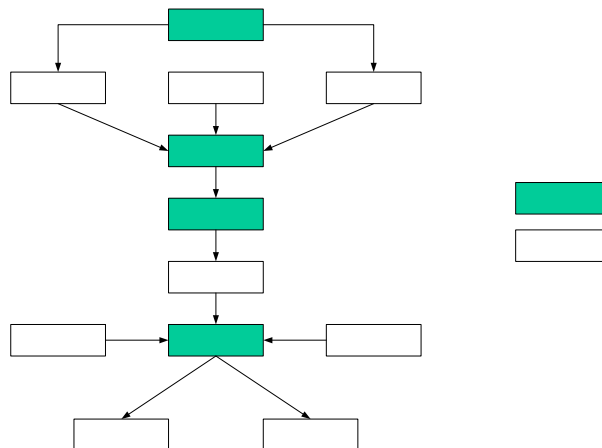
- **Integrated Development Environment**
 - Language Sensitive Text Editor
 - Preprocessor
 - Compiler
 - Linker
 - Wizard
- **Project (.dsp)**
 - A collection of all the necessary information to build a binary executables (.exe .dll)
 - Files (source, header)
 - Compile options
 - Link options
- **Work Space (.dsw)**
 - A collection of projects

2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-47

Build Process



2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-48

Assembly Code

- Project Setting -> C/C++ -> Category -> Listing files -> Listing file type -> Assembly, Machine Code, and Source
- Then, compile
- You'll see xxx.cod file in the debug directory

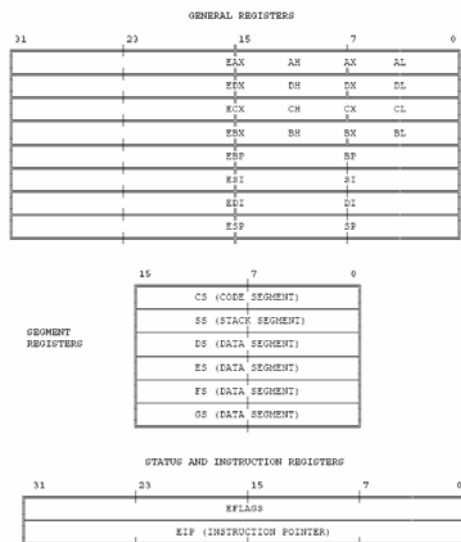
2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-49

Intel 80386 Registers

Figure 2-5. 80386 Applications Register Set



Page 31 of 421

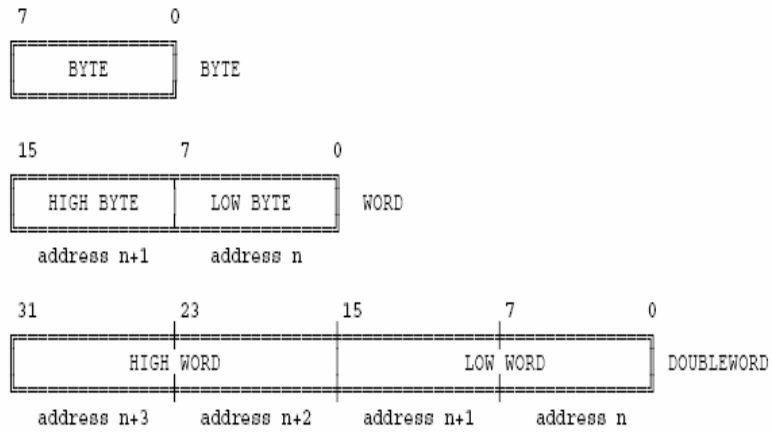
2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-50

Intel Fundamental Data Type

Figure 2-2. Fundamental Data Types



2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-51

Assembly code

- Instruction = operation [operand] [, operand]
- Examples
 - Data movement: `mov destaddr, eax`
 - Stack operation: `pop eax`
 - Arithmetic, logic, comparison, etc

2006-09-06

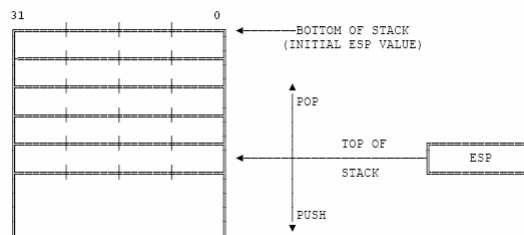
Seong Jong Choi

Multithreading CHAPTER 1-52

Stack Instructions

- PUSH
 - Decrement the stack pointer (ESP)
 - Then, transfer source to the stack indicated by ESP
 - Push eax
- POP
 - Transfer data at the current top of stack (ESP)
 - Then, increment ESP
 - Pop eax

Figure 2-7. 80386 Stack



2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-55

Stack Instructions

Figure 3-1. PUSH

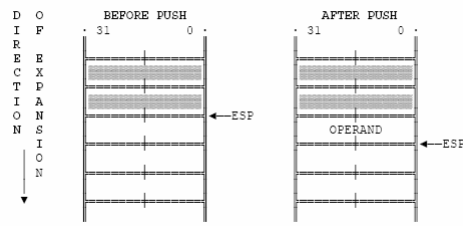
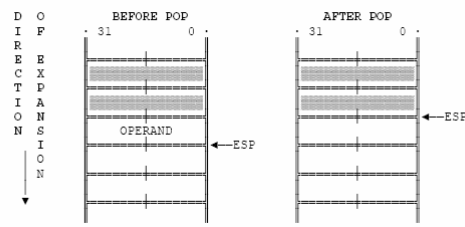


Figure 3-3. POP



2006-09-06

Seong Jong Choi

Multithreading CHAPTER 1-56