**OS책 문제 풀이**

권진우

운영체제의 주요 목적 : 사용자에게 컴퓨터 하드웨어의 프로그램 실행을 편리하고 효율적으로 제공, CPU할당되는 자원들의 시간을 분할하여 공평하고 효율적으로 프로세스 할당 사용자 모드에서 시스템 관련 명령어와 같은 특권 명령어 접근을 제한하고 I/O장치 관리

OS가 자원을 낭비할 때 : 사용자 시스템은 사용자를 위한 시스템이 되어야 한다. 따라서 CPU 사이클 일부를 버리고 시스템과 사용자간의 상호작용을 위해 CPU 자원을 쓰는 것은 사용자의 요구를 위한 작업을 수행할 수 있게 해준다.

실시간 환경을 대상으로 하는 운영체제 개발할때 중요한것 : 시스템이 특정시간 내에 작업을 완료하지 못했을 때, 실행중인 시스템은 오류가 나는 원인이 될 수 있다. 따라서 실시간 시스템을 위한 운영체제를 작성하려면, 개발자는 자신의 스케쥴링 계획을 응답시간이 시간제한을 초과하지 않도록 하여야 한다.

운영체제 만들 때, 운영체제가 포함해야 하는 것과 하지 말아야 할 것 : 운영체제를 정의할 때 많은 프로그램을 포함시키는 것은 커널 내에서 실행되는 프로세스가 많다는 점에서 보안상 취약할 수 있으며 이러한 프로그램을 많이 넣을 경우 OS가 무거워질 수 있다.

커널 모드와 사용자 모드 구분을 보호 시스템의 기초적인 형태로 사용할 수 있는 방법은? : 커널 모드와 사용자 모드의 구별이 초보적인 형태로 보호를 해줄 수 있는 이유는 특권 명령어와 같은 CPU사이클 제어와 같은 것은 사용자가 제어 할 수 없는 OS가 제어하는 Kernel모드에서만 명령어를 쓸 수 있도록 함으로써 사용자가 하드웨어에 직접 액세스하지 못하게 OS를 통해 서비스를 받고 따라서 사용자의 명령어가 특권명령어를 사용할 수 없게 함으로서 컴퓨터의 주요 자원을 보호할 수 있다.

특권명령어가 필요한 지침 : 타이머 값, 메모리 삭제, 인터럽트 중단, 장치의 상태표 수정, 입출력장치 액세스

특권명령어 필요없는 지침 : 시계 읽기, 함정 명령어 지시, 사용자모드에서 커널모드 전환

운영체제를 특정 메모리 파티션에 넣어 접근 불가하게 만들때 문제점 : 운영체제가 요구하는 데이터(암호)는 때에 따라서 보호되지 않은 메모리에 저장하거나 통과해야 하므로 권한이 없는 사용자도 접근 할 수 있어야한다. + OS 공간을 고정적으로 차지하여 메모리가 낭비된다.(OS 확장 불가)

CPU 모드 : Kernel 모드, 사용자 모드

캐시가 유용한 이유 2가지 : 캐시는 둘 이상의 저장장치가 데이터를 교환할 때 유용하며 두 저장장치 사이간 속도차이로 인한 병목현상을 줄여줄 수 있으며 속도의 버퍼를 제공한다고 할 수 있다. 고속 장치(Ex. CPU)가 캐시에서 필요로 하는 데이터를 발견하면, 더 느린 장치를 기다릴 필요가 없다. 캐시의 데이터는 구성요소의 데이터와 일치해야 하는 것이다. 만약 구성품이 변경된 데이터 값을 갖고 있고, 지난 데이터가 캐시에 있는 경우, 캐시도 변경된 데이터 값으로 변경이 되어야 한다.(변경된 데이터가 캐시에 업데이트되지 않는 아주 잠깐의 순간이 존재 가능) 이것은 특히 두 개 이상의 프로세스가 기준점에 접근하고 있는 다중 프로세서 시스템에서 문제가 되고 구성요소는 동일한 크기의 캐시에 의해 제거될 수 있지만, 다음과 같은 경우 해당된다. 구성품은 동등한 상태절약 능력을 가지고 있다. 즉, 부품이 전기를 제거할 때 데이터를 보관하는 경우, 캐시는 또한 데이터를 보관하여야 한다. 캐시가 클 경우 그 저장 용량에 따른 가격이 매우 비싸질 수 있다. 또한 캐시(cache)는 휘발성 메모리이다.

분산 시스템의 클라이언트 서버 모델과 피어간 모델 차이점 : 클라이언트 시스템은 1개의 서버가 여러 클라이언트와 연결되어있는 시스템이고 피어간 모델은 P2P로써 각각의 클라이언트가 모두 서버가 될 수 있다.

다중 프로그래밍과 시분할 환경에서, 다수 사용자가 동시에 시스템을 공유한다.

문제점 두가지 : 동시에 공유 메모리에 접근하는 경우 데이터 충돌이 발생할 수 있다.

메인프레임 또는 미니 컴퓨터 : 크기와 휴대성

서버에 연결된 워크스테이션 : 성능과 고속 LAN, Middleware를 통한 단일 시스템통합

휴대용 컴퓨터 : 휴대성

시분할 시스템을 사용하는것이 좋을 때 : 많은이들에게 데이터를 공유하여 빠른 시간내에 사용자 전환이 필요한 경우

대칭적 비대칭적 안나옴

클러스터형 시스템과 다중 처리기 시스템 차이점 : 클러스터형아 시스템은 여러 워크스테이션이나 PC들을 고속LAN과 같은 네트워크로 통합하여 Middleware를 통해 단일 이미지 시스템 및 가용성 인프라를 구축하여서 통합 사용하는 것이고 다중 처리기 시스템은 수많은 CPU를 연결 시켜서 공유 메모리의 데이터를 나누어 처리하는 것이다. 두 컴퓨터가 협력하는데 필요한것 : Middleware, 고속 LAN

17,18, 안나옴

인터럽트의 목적은 무엇이며, 트랩과 인터럽트의 차이점은 무엇인가? 트랩은 사용자 프로그램에 의해 의도적으로 발생할 수 있는가? 그렇다면 그 목적은 무엇인가?

-인터럽트의 목적은 CPU(중앙처리장치)가 연산을 함에 있어서 사이클마다 인터럽트를 확인하는데 이 목적은 Software 인터럽트, Hardware 인터럽트, exception 인터럽트가 있고 보통 인터럽트 신호를 CPU에게 보내서 기존 수행 중이던 작업을 일시적으로 스택에 저장하여 중지시키고 I/O장치의 데이터를 처리하거나 예외 처리 등 인터럽트를 처리한 후 완료가 되면 다시 원래의 수행 중이던 작업으로 되돌아와서 실행 되는 것이 인터럽트이며 인터럽트를 통해서 CPU가 한가지 작업 뿐만이 아닌 사용자가 원하는 여러가지의 작업을 수행하고 즉시 반응할 수 있도록 해준다. 트랩은 Exception Interrupt로 인해 Internal Interrupt가 발생하는 것을 의미한다.

A. 전송을 조율하기 위하여 CPU는 어떻게 장치와 결합 작동하는가?

- CPU와 기억장치, I/O장치들과의 속도 차이로 인한 병목현상을 방지하기 위해서 CPU는 다른 기억장치들과의 연결 통로 사이에 버퍼를 사용하며 각 장치들 마다 컨트롤러를 두어서 예를 들어 입출력장치의 컨트롤러가 입출력에 필요한 데이터를 모두 처리하면 입출력장치 컨트롤러가 데이터를 갖고 있다가 CPU에게 인터럽트를 보내고 처리 완료되어 갖고 있던 데이터를 한번에 전송한다.

-EX) CPU - Device Controller - I/O Device

B. CPU는 메모리 연산이 종료되었음을 어떻게 알 수 있는가?

-앞서 말했듯이 각 기억장치나 외부장치들의 컨트롤러가 장치 드라이버를 통해 제어되고 장치 컨트롤러는 장치가 처리한 데이터를 갖고 있다가 CPU에 인터럽트 신호를 보내면 CPU가 신호를 신호를 받고 메모리 연산이 종료되었음을 알게 된다. 이후 ISR을 통해서 인터럽트를 처리한다.

C. DMA가 데이터를 전송하는 동안 CPU는 다른 프로그램을 실행할 수 있다. 이 프로세스는 사용자 프로그램의 실행을 방해하는가? 그렇다면 어떤 형태의 방해가 일어나는가?

-DMA는 장치의 컨트롤러가 데이터를 모두 처리한 후에 CPU에 인터럽트 신호를 보내는데 데이터의 전송 단위가 크지 않기 때문에 polling방식 보다는 CPU에 부하가 덜하지만 인터럽트 한번에 전송하는 데이터의 크기가 크지 않기 때문에 인터럽트 신호가 마찬가지로 많이 발생하게 되어서CPU가 많은 부하를 받게 된다. 이를 방지하기 위해 DMA 방식은 DMA컨트롤러가 따로 존재하여 CPU가 처리 해주어야 할 인터럽트 신호를 대신하여 CPU가 좀 더 적은 인터럽트를 받게 만들어준다. CPU가 할 일을 DMA가 일부 대신하여 줌(Device 인터럽트 신호를 대신 받고 많은 데이터를 모아서 원하는 데이터의 처리가 모두 완료되었을 때)으로써 CPU가 한번의 인터럽트 신호로 처리 완료된 데이터를 가져갈 수 있다. 사용자 프로그램 실행을 방해 여부는 아마도 CPU가 인터럽트 신호 한번에 많은 양의 데이터를 가져가기 때문에 사용자 프로그램을 실행 시 순간적으로 인터럽트 신호를 처리하기 위해 프로그램이 잠시 느려질 수 있다.

한 프로그램이 다른 프로그램이 사용하는 메모리를 변경하는 것을 막기위해 사용 되는 보호 기법 : 프로세스 인터럽트 불가상태 만들기 => 한 프로그램이 실행 되는데 다른프로그램이 사용하는 메모리를 변경하려하기 전에 해당 프로그램이 인터럽트 불가 상태로 만든 후 작업을 모두 실행하고 다음 프로세스로 CPU사이클을 넘기면 된다. 임계구역설정하기 => 5장 공유메모리 사용시 프로세스간 서로 데이터를 변경할 수 있는 부분에 임계구역(CS)을 설정하여 CS부분은 한가지 프로세스만 동시에 접근할 수 있도록 하는 것

클라이언트 서버 시스템에 비해 피어간 시스템 장점 : 클라이언트 서버 시스템은 1:다 시스템으로써 클라이언트 들은 서버로부터만 통신 가능 하지만 P2P시스템은 모든 클라이언트가 서버가 되서 서로간 통신이 가능하다.

오픈 소스 운영체제 : DRM, CopyLeft, 등등 프린트 마지막장 보기

2장

시스템 호출 목적 : 사용자 프로그램에서 System-Call을 통해서 특권 명령어를 사용할 수 있는OS(Kernel)모드에 서비스를 받게 요청할 수 있다.

프로세스 관리와 연관된 운영체제의 주요 활동 5가지 : 프로세스 생성, 제거 , 프로세스 중단 및 재개 , 프로세스 동기화 , 통신, 교착상태 처리

메모리와 연관된 운영체제의 주요 활동 3가지 : 메모리 자원 할당, 회수 , 회계 : 자원에 대한 사용기록 , 적재할 프로세스 선택

보조 기억장치 관리와 연관된 OS 활동 3가지 : 디스크 공간 관리, 저장 공간 할당, 디스크 스케쥴링

명령어 해독기의 목적? 커널에 포함되지 않는 이유 : 명령에 해석기는 사용자나 명령 파일로 부터 실행하고, 보통 하나 또는 그 이상의 System-Call로 실행된다. 명령어 해석기는 변경될수 있으므로 일반적으로 커널의 일부가 아니다.

커널을 시작하기 위해 셸이 실행해야하는 시스템 호출은? : fork()명령어 : fork명령어로 System-Call을 하여 현재 실행중인 프로세스를 복제하여 실행한다. 이후 exec() 명령어의 System-Call로 복제 된 프로세스를 수정(오버레이 or 지훈이)하여 새로운 프로세스로 실행한다.

시스템 프로그램의 목적 : 유용한 System - Call의 묶음으로써 기본적인 기능을 제공하고 간단한 문제를 해결할 때 사용자가 그들 자신의 프로그램을 사용하지 않아도 되도록 한다.

계층화된 시스템 설계 방식 이점? 단점? : 구현과 디버깅이 간단하고 사용자 입장에서 연산이 무슨 동작을 하는지만 알면 된다(구현되는 방법 몰라도 됨). 하지만 각 계층의 정의하기가 어렵고 각 계층별 통신을 할때 통신 오버헤드가 많이 발생한다. 특히 낮은 Level에서 사용자 프로그램 쪽High Level 쪽으로의 통신도 일어날 수 있기 때문에 불리하다.

운영체제가 사용자를 위해 제공하는 서비스 5개 : 사용자 인터페이스(UI), 입출력 연산 , 통신, 파일 시스템 연산, 오류 탐지

일부 시스템에서 운영 체제를 펌웨어에 저장하는 반면 다른 시스템에서는 디스크에 저장하는 이유는? : 펌웨어는 잘 수정되기 어려워서 PDA 및 휴대폰과 같은 특정 장치의 경우 파일 시스템이 있는 디스크에 OS를 저장해놓을 수 없을 수도 있다. 이럴 경우 펌웨어에 저장하고 OS는 정책에 따라서 변경 될 수 있기 때문에 컴퓨터나 특정 장치와 같은 경우는 OS의 수정이나 업데이트를 간편하고 쉽게 하기 위해 디스크에 저장한다. 펌웨어는 수정하기 어렵다(보통 ROM에 저장)

부트스트랩 프로그램이 실행해야 하는 작업 : 전원이 공급 되면 CPU로 신호가 세트되고 CPU는 ROM에 있는 Bootstrap프로그램(Firmware)을 실행 시킨다. 이때ROM에서 실행하면 시간이 오래 걸리므로 RAM 메모리에 적재하여 실행시킨다. Boothstrap 프로그램이 실행되면 디스크의 'Boot Block'(첫위치)에 있는 OS의 커널을 간단한 프로그램으로부터 단계적으로 높은 Complex를 가진 Kernel로 실행시킨다. 또한 시스템을 진단하고 초기화 시키는 역할을 수행한다.

운영체제가 제공하는 서비스와 기능 : 사용자/프로그래머 입장(서비스), 인터페이스(기능)

인터페이스 : 설계자와 사용자간의 연결고리

OS로 매개변수 전달방법 : 레지스터(레지스터에 매개변수 저장, 갯수 제한) , 스택(시스템 스택에 Push(응용프로그램) Pop(OS)) , 블록(매개변수 많을 때 사용, 메모리 블록에 매개변수 저장하고 블록의 주소를 레지스터를 통해 전달)

통계 프로파일 기법 및 이유 : 회계기법, 사용자 과금을 위해

파일 관리와 관련된 5가지 활동 : 파일 변경, 파일 상태정보, 프로그램 적재 및 실행, 백그라운드 서비스

파일과 장치에 접근하기위해 같은 시스템 호출 인터페이스를 사용할때 장단점 : 시스템 호출 인터페이스는 응용프로그램 인터페이스(API)와는 달리 같은 기능에 대한 명령어가 OS마다 상이하고 이로인해 유지보수가 어렵고 이식성이 떨어진다. 장점으로는 API는 응용프로그램 Library함수를 통해서 시스템 호출을 하지만 시스템 호출 인터페이스는 바로 접근할 수 있기 때문에 통신 오버헤드가 없다.

운영체제가 제공하는 시스템 호출 인터페이스를 사용하여 사용자가 새로운 명령어 해석기를 개발하는 것은 가능한가?

-입출력 장치와 파일들 간에는 유사성이 매우 많기 때문에 많은 운영체제가 이들 둘을 통합된 파일-장치 구조로 결합하였다. 이 경우 같은 시스템 호출들이 파일과 장치에 사용된다. 따라서 시스템 호출은 다르더라도 사용자 인터페이스(UI)를 통해서 새로운 명령어 해석기와 같은 인터페이스를 구축하는 것이 가능하다.

프로세스간 2가지 통신 모델 : Message Passing / Shared Memory : 메시지 패싱 방식은 프로세스에서 커널로 메시지를 보내서 커널이 메시지를 받아 다시 다른 프로세스로 메시지를 보내는데 통신 오버헤드가 발생하고 적은 용량의 메시지를 전달할때는 좋지만 많은 용량을 전송하려면 여러번의 복사라 이루어져야 하므로 비효율적이다. 또한 CPU 멀티코어 갯수에 따른 성능 향상이 가능하다. 공유 메모리 방식은 메모리에 프로세스끼리 공유하는 메모리공간을 할당하는 방법으로써 공유 메모리를 할당할 때 시스템 호출 순간에만 인터럽트가 일어나고 할당된 메모리로 각 프로세스가 데이터를 전송하고 가져온다. 따라서 속도가 빠르다, 하지만 CPU 코어 수 증가시 캐시 일관성 문제로 성능 저하가 올 수 있다.

기법과 정책을 분리하는 것이 바람직한 이유는?

-빌딩 블록의 기본 집합을 구현함으로써 기법과 정책 분리를 추구하는데, 이 블록들은 정책으로부터 거의 자유롭고, 보다 고급의 기법과 정책들이 사용자 생성 커널 모듈이나 사용자 프로그램 자체를 통해 첨가될 수 있기 때문이다. 또한 정책과 기법 분리는 융통성을 위해 아주 중요하다. 정책은 장소나 시간에 따라 바뀔 수 있기 때문이다. 기법-어떻게 할것인가, 정책-무엇을 할것인가

시스템을 설계할 때 마이크로커널 방식을 사용하느 장점은 무엇인가? 마이크로 커널 구조에서 사용자 프로그램과 시스템 서비스가 상호작용하는 방식에 대해 설명하시오. 마이크로커널 방식을 사용할 때의 단점은 무엇인가?

-마이크로 커널은 계층적구조인 Layered 커널(계층별 시스템 호출을 하고 계층적 구조를 가짐으로 인해서 커널의 용량(크기)가 커지고 무거워져서 설계하는데 어려운 면)을 갖고 있는 것과는 달리 이를 해결하기 위해 사용자 모드로 상당히 많은 주요 기능들을 빼낸 커널에 꼭 필요한 부분만 넣은 Microkernel(smallkernel)이 등장하였고 사용자 모듈에서 필요 시 마다 커널로 message-passing기능을 통해서 서비스를 요청하는 방식을 사용한다. 이로써 발생하는 장점은 OS확장이 용이해졌고 이식이 용이하며 높은 신뢰성+보안성을 가진다.(서비스가 잘못되더라도 사용자 프로그램에서 잘못될 확률이 큼) 단점으로는 사용자 모드와 커널 모드간 통신 오버헤드가 발생함으로써 성능이 좋지는 못함.

적재 가능 커널 모듈을 사용하는 장점은?

적재 가능 커널 모듈(Loadable kernel module)을 사용하면 꼭 필요한 것만 커널에 넣어두고 필요시 마다 선택적으로 실행시간 동안 동적으로 적재되어 커널 기능을 확장하기 쉽고 서로 프로그램,시스템간에 통신 오버헤드가 발생하지 않아서 성능 또한 보장된다. 유연하고 효율적인 장점이 있다.

IOS와 Android 차이점과 유사점 : 안드로이드는 Linux커널을 쓰고 그 위에 안드로이드 가상머신과 라이브러리가 있다. 그 위에 App framework를 통해 APP을 구현한다. IOS는 Core Service 와 Core OS 를 분리하였다. 두 OS의 유사점은 휴대폰에 사용된다.

3장

1. 5 : 부모의 value값을 따른다 자식의 pid==0, 부모의 pid>0이기 때문

2. 2\*2\*2=8

3. 병행처리를 위한 3가지 요건 : 공유 메모리 접근제어, 모듈성(프로세스를 서브작업으로 나눔), OS와 하드웨어의 지원?

4. 레지스터 Change

5. fork 연산 수행시 부모 프로세스와 자식 프로세스가 공유하는 상태는 공유 메모리 영역이다.

-Symmentic : CPU가 같은 모든 작업을 나누어 수행할 수 있음 Asymmentic : CPU가 각각의 특정 작업을 수행함 (cpu0 : os , cpu1 : program)

8.장기 스케쥴러 : 자주 일어나지 않는 디스크Pool에서 메모리의 ReadyQueue로 이동시킬 프로세스 선택, 현대(시분할(대화형)시스템)에는 사람이 역할을 대신 수행, Multiprogramming정도를 제어

중기 스케쥴러 : 메모리에서 Disk로 이동시킬 프로세스를 선택(Swapping), 메모리가 Full이어서 디스크로 보낼 프로세스를 선택하는 것(Disk 가상화 사용 가능)

단기 스케쥴러 : 다음에 실행시킬 프로세스를 선택하여 CPU에 할당, CPU 스케쥴러

프로세스간 Context switching시 커널의 작업 : 커널은 현재 프로세스의 PC, 데이터 값을 레지스터 혹은 PCB(메모리)에 저장하고 Waiting Queue로 보내고 Ready Queue에 있는 프로세스중 적절한 프로세스를 실행시킨다.

Init 프로세스를 설명하시오 : Init 프로세스는 Unix 시스템에서 시작 프로세스로써 OS가 시작될 때 처음으로 시작되는 프로그램으로써 최상단 부모 프로세스이다. 백그라운드 프로세스로 동작하며 어떤 프로세스가 부모 프로세스가 종료되어서 고아프로세스로 남아서 실행되고 있을 때 발견하여 부모 프로세스 역할을 대신하여 자식프로세스가 exit되서 terminated상태가 되면 메모리에서 회수시켜준다.

12. 2\*2\*2\*2=16

14. 0, 2603, 2603, 2600

일반 파이프 사용 예와 지명 파이프 사용 예

일반 파이프는 단방향 전송으로써 생성자가 파이프를 생성하여 Write-end만 사용가능하고 소비자는 Read-end만 사용 가능하다. 따라서 일반 파이프는 생성한 프로세스만 접근 가능하다.

Unix : pipe(), windows : CreatePipe()

지명파이프는 양방향 통신으로써 반이중과 전이중 통신으로 나뉜다. 통신하는 프로세스간 특별한 관계일 경우 생성한다. 지명파이프는 파일 형태로 존재하여 부모-자식 관계 필요없이 파일 형태의 지명파이프를 통해 프로세스들이 데이터를 주고 받을 수 있다. (SandBox와 유사)

동기적 통신 : Blocking 송수신 : 동작이 상대편 동작에 영향을 받는다

- 송신(Send) 프로세스는 수신 프로세스나 메일박스가 메시지를 받을 때 까지 Block 상태

- 수신(Receive) 프로세스는 송신 프로세스로부터 수신 메시지가 있을 때 까지 Block 상태

비동기적 통신 : Non-Blocking 송수신

- Non send : 송신 프로세스는 메시지를 보내고 데이터 송신여부만 return받아서 바로 작업 계속 실행(수신여부 필요 X)

- Non receive : 수신프로세스는 유효한 메시지를 받거나 NULL을 받고 바로 return, 작업 수행

4장

다중 쓰레딩이 단일 쓰레딩 보다 더 빠른 이유는 : 다중 쓰레드는 프로세스를 여러개로 나누어 동시에 처리할 수 있기 때문 에 더 빠르다. ex) GUI

사용자 수준 쓰레드와 커널 수준 쓰레드 차이점 : 사용자 수준 쓰레드(onetomany)는 사용자 수준 에서 thread library에 의해 구현되고 Kernel수준 쓰레드는 OS커널에서 직접 지원되고 구현, 관리 된다.

사용자 수준 쓰레드는 Context Switching과 동기화 오버헤드가 적다(사용자 수준은 context switching이 즉시 일어남 커널수준에서 일어나는 Context switching이 없음). 하지만 커널 입장에서는 다중 쓰레드를 인식하지 못하고 단일 쓰레드로 인식하여 하나의 쓰레드가 Block 되게 되면 다른 모든 thread가 동시에 block되어버린다. 반면 커널 수준 쓰레드(onetoone,manytomany)는 Context Switching 과 동기화 오버헤드가 발생하지만 한 쓰레드의 프로세스가 block되어도 다른 thread로 스케쥴링 할 수 있다.

커널 수준 스레드(다중 쓰레딩) 문맥교환 할 때 커널이 수행하는 작업 : CPU Context Switching (CPU 레지스터 값(PC), 스택, 쓰레드 전용 메모리공간(static))

쓰레드 생성시 사용되는 자원 : 스택, CPU 레지스터, 쓰레드 전용 메모리공간/ 프로세스 생성시 : 스택, CPU레지스터, <코드, data, 파일> <<==쓰레드환경에서는 공유

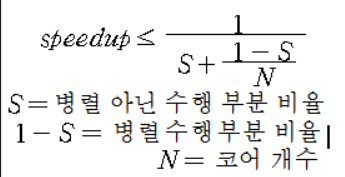
다중 쓰레드는 어떤 쓰레드가 먼저 실행되는지 알 수 없다.

단일 처리기 시스템 프로그래밍에서 다중 커널 스레드를 사용하여 다중 스레드 프로그램을 만들었을 때 이 프로그램의 성능이 더 나은 성능을 보일 수 있는 경우는 ? 속도는 단일 처리기 시스템이라서 더 나아질 수 없지만 프로세스를 여러 쓰레드로 나누어 실행함으로써 단계 별로 실행하는 것 과 같은 상황으로 사용자와의 응답성이 빨라진다.

다중 쓰레드 사이 공유되는 것은 ? : 힙(공용) 메모리, 전역 변수(Data),, file, code

다수의 사용자 수준 스레드를 사용하는 다중 쓰레드 해결책이 단일 처리기 시스템보다 다중 처리기 시스템에서 더 나은 향상을 보일 수 있다.

병렬성이 아닌 동시 수행성 : 병행수행 : 완전한 동시 실행은 아니지만 프로세스들을 동시에 수행하는것과 같게 잘게 분할(context switching)하여서 돌아가며 수행

 <- 병렬실행 프로세스 수행시 얻을 수 있는 성능 향상

4.14 : 입력, 출력에 각각 1개씩 쓰레드 제공, 연산을 위해 2개의 쓰레드 제공

15 : 2\*2+2=6, 2개의 쓰레드

5장

인터럽트가 불가능하게 만들면 시스템 클록은 1초마다 인터럽트를 보내서 클록을 재생시키는데 인터럽트가 자주 불가능하게 만들면 시스템 클록의 인터럽트가 불가되어서 클록에 영향을 미칠 수가 있다.

Spin락, Mutex락 프린트 보기

Busy Waiting이란 무엇인가? : 프로세스가 임계구역에 진입하려 할때 다른 프로세스가 이미 CS구역에서 실행중이어서 진입하지 못하여 해당 프로그램 코드를 대기 루프를 돌면서 기다리는 것이 Busy Waiting이라고 한다. 다른 대기 방식에는 Busy Waiting이 없는 Mutex락 기법이 있는데 이는 TAS를 사용한 상호 배제로써 while대기 루프문 안에 block을 넣어서 프로세스를 잠재우고 CS구역에 있는 타 프로세스가 작업이 완료되어 빠져 나올 때 Release를 호출하여 잠자는 프로세스를 Wake up 시켜서 block되어있던 프로세스를 깨우고 임계구역 진입.

Mutex : Mutual Exclusion 상호배제 락

TAS : TestAndSet 명령어로 while 문 내부의 코드를 한 명령어로써 한번에 임계구역진입가능 여부 검사

스핀락 : 단일 프로세서에서는 좋지 않다 프로세스들을 돌면서(루프 반복 실행) Lock 획득을 기다리는데 이는 CPU사이클을 낭비시킨다. 단일 프로세스는 한번에 한 프로세스만 처리하기 때문에 따로 CS구역을 지정하고 Lock을 확인 할 필요가 없다. 따라서 스핀락으 단일 프로세서 시스템에서는 CPU사이클 낭비, 멀티 코어 프로세서에서 효율적(Context Switcing이 없음)

중간고사 범위 끝