1. **운영체제 소개 개관**

자원의 관리, 활용이 효율적으로 동작 운영체제에 대한 전반적인 개요

운영체제의 정의와 역할

운영체제를 구성하는 기본적 요소인 프로세스 관리자, 메모리 관리자, 장치 관리자, 파일 관리자 역할 학습 목표

운영체제의 개념과 역할을 이해 운영체제의 구성을 이해 운영체제의 유형을 이해

주요 용어

운영체제 커널 모드 사용자 모드 시스템 호출

일괄처리 운영체제 시분할 운영체제 실시간 운영체제 분산 운영체제

* 1. **운영체제란 무엇인가?**
     1. **운영체제의 정의 [3]**

**컴퓨터 시스템 하드웨어(hardware)**

**CPU, 메모리, 저장장치, 입출력장치, 네트워크 장치 저장장치**

**하드 디스크 / SSD**

입출력장치

키보드, 마우스, 프린터 네트워크 장치

**유무선 네트워크 카드 소프트웨어(software)**

응용 소프트웨어 일반적으로 사용하는 앱

웹 브라우저, 워드프로세서, 게임, 그래픽 소프트웨어 시스템 소프트웨어

하드웨어와 밀접한 관계를 갖고서 응용 소프트웨어와 하드웨어 사이의 매개체 역할 운영체제, 컴파일러

운영체제란?

시스템 소프트웨어로 컴퓨터 시스템의 자원을 관리하고 컴퓨터 프로그램이 동작하기 위한 서비스를 제공하는 프로그램들의 모음

* + - 1. **운영체제란 무엇인가? (2) [3]**
    1. **운영체제의 역할 자원 관리**

자원관리 예시

1. 저장장치에서 데이터 읽어 오기
2. 키보드나 마우스 등의 장치 제어
3. 여러 프로그램이 동시에 실행될 때 CPU와 메모리 관리를 효율적으로 함.

컴퓨터 시스템의 자원 관리 => (목적) 컴퓨터 시스템 효율적 운영 하드웨어 자원

CPU, 메모리, 키보드, 마우스, 네트워크 카드

소프트웨어 자원

웹 브라우저, 워드프로세서, 게임 등

데이터

사용자 지원 => 사용자에게 편의성 제공

GUI 형태의 인터페이스 제공 -> 사용자가 원하는 작업 찾아 수행 사용자가 USB 메모리를 입력시 저장 데이터 볼 수 있도록 키보드로 입력한 내용 응용 프로그램에 전달

* + 1. **컴퓨터 시스템과 운영체제 [4]**

운영체제 없던 초기 컴퓨터 시스템

응용 프로그램이 직접 컴퓨터 시스템 자원 제어

사용자가 응용 프로그램을 실행 -> 응용 프로그램이 직접 컴퓨터 시스템 자원 제어 응용 프로그램 개발자 -> 하드웨어에 대한 제어방법 잘 알고 있어야.

하나의 하드웨어를 여러 사용자가 공유

응용 프로그램들끼리 효율적 자원 나눠쓰기 어려움 -> 컴퓨터 시스템 제대로 동작 x

=> 하드웨어와 응용프로그램 사이에 운영체제 두고 하드웨어에 대한 제어는 운영체제만 하도록 함.

응용프로그램 => 운영체제에 부탁 => 운영체제가 대신 하드웨어 제어 하드웨어에 대한 자세한 정보 모르더라도 기능 수행)

=> 그 결과를 응용 프로그램에 제공 (응용프로그램이

여러 응용프로그램이 동시에 수행되더라도 운영체제가 중간에서 충돌 없이 자원을 효율적으로 나눠 줌

* 1. **운영체제란 무엇인가? (3)**
     1. **커널 모드와 사용자 모드 배경**

하드웨어에 대한 제어를 운영체제만 하도록 하기 위한 방법 응용 프로그램이 하드웨어에 대한 직접 접근 강제 막기

시스템 안정성 보장

**커널 모드와 사용자 모드 사용 커널 모드(kernel mode)**

하드웨어를 직접 제어할 수 있는 CPU의 명령어를 사용할 수 있는 모드 운영체제만 동작

**사용자 모드(user mode)**

**CPU 명령어 사용 X (하드웨어 직접 제어 X)**

**응용 프로그램은 사용자 모드에서만 동작 커널(kernel)**

응용 프로그램과 하드웨어 수준의 처리 사이의 가교 역할 특징

커널 모드에서만 동작 운영체제의 핵심 요소

유형

**일체형 커널 (monolithic kerne)**

* **운영체제의 모든 서비스가 커널 내에 포함 - 모든 요소가 커널 내부에 존재 장점**

여러 서비스간 효율적인 상호작용 단점

**하나의 요소라도 오류가 발생하면 시스템 전체에 장애 마이크로커널(microkernel)**

* **운영체제 요소의 대부분 커널 외부로 분리 - 커널 내부에는 최소한의 요소만 남겨 놓음**

**커널 내부: (1) 메모리 관리, (2) 프로세스 간 통신(InterProcess Communication: IPC), (3) 멀티 태스킹 등 최소 한의 요소만**

커널 외부: (1) 파일 시스템 (2) 장치 드라이버 (3) 네트워크 프로토콜 등 커널 모드가 아닌 사용자 모드에서 동작하도록 분리됨

장치 드라이버(Device Driver)?

: 운영체제와 하드웨어 장치 간의 통신을 가능하게 하는 소프트웨어.

운영체제가 하드웨어를 직접 제어하는 대신, 장치 드라이버를 통해 하드웨어를 제어하고 데이터를 주고 받음.

장점

1. **운영체제 확장 용이**

새로운 서비스 추가 용이

1. **안정성, 유지보수 용이**

커널 외부 요소 문제 발생하더라도, 커널 자체에는 영향 X

단점

성능 저하: 커널 외부 운영체제 요소들 사이에 데이터 전달이 필요한 경우 프로세스 간 통신(IPC)을 통해야만 함.

시스템 호출

- 응용 프로그램이 하드웨어에 대한 제어가 필요한 경우 "시스템 호출"이용 커널 모드로 변경 => 운영체제의 커널 동작 => 하드웨어 제어

* 1. **운영체제의 구성**

**운영체제의 구성 요소(x4) : 프메장파[8] 유형**

프로세스 관리자 메모리 관리자 장치 관리자 파일 관리자

각 서브시스템의 역할

**(1) 자신 자원 모니터링 (2) 정해 둔 기준에 따라 자원 배분/회수**

* + 1. **프로세스 관리자 프로세스 (process)**

실행 중인 프로그램 역할

프로세스 생성, 삭제

프로스스에 CPU를 할당하기 위한 스케줄 결정 각 프로세스 상태 관리, 상태 전이 처리

- 운영체제에 서비스를 요청

- 프로세스는 매 순간 준비, 실행, 대기 등 다양한 상태 중 하나 - 프로세스 관리자는 CPU 할당/회수 등 상황에 따라 프로 세스의 상태 전이시킴

* + 1. **메모리 관리자**

역할: 메모리(주기억장치) 공간에 대한 요구가 발생하면 이에 대한 유효성 확인

메모리 할당 및 회수 메모리 공간 보호

* + - 1. 한 프로세스에 할당한 메모리 공간을 다른 프로세스가 접근하여 사용하지 못하도록 관리
      2. 운영체제가 직접 사용하는 메모리 공간에 어떤 응용 프로그램의 프로세스도 접근하지 못하도록 관리
    1. **장치 관리자 역할**

컴퓨터 시스템의 모든 장치 관리 (저장, 입출력, 네트워크) 효율적 장치 관리를 위한 스케줄링 포함 다양한 기법 활용

* + 1. **파일 관리자 역할**

파일 생, 열, 수, 닫, 삭 파일 저장장치 공간 관리 파일 접근 제한 관리

* 1. **운영체제의 유형 [10]**

**운영체제의 대표적 유형(X4) : 일시실분 유형**

**일괄처리 운영체제(batch processing)**

# 시분할(time-sharing) 실시간(real-time) 분산(distributed)

* + 1. **일괄처리(batch processing) 운영체제 작업을 모아서 순서대로 처리**

장점

사람(오퍼레이터)이 하던 일을 프로그램이 빠르게 처리 => 전체적 작업 처리속도 향상 단점

한 번에 한 프로그램씩 수행 => 컴퓨터 시스템 전체적인 자원 충분히 활용X

나중에 들어온 작업은 앞서 기다리던 작업들이 모두 끝날 때까지 아무런 상호작용 없이 기다려야만 했음.

* + 1. **시분할(time-sharing) 운영체제 = 대화형(interactive) 운영체제**

각 사용자 프로그램 한 번에 조금씩 수행 => 마치 여러 프로그램이 동시에 실행되는 것과 비슷한 효과 장점

응답시간(요청 시점부터 반응이 시작되는 시점까지의 소요시간) 이 일괄처리 운영체제보다 크게 단축

각자 터미널로 동시에 컴퓨터에 접속한 여러 사용자 -> 컴퓨터에 명령 직접 내리고, 수 있게 됨

특징

그에 대한 응답도 빠른 시간 안에 받을

"실행이 완료되지 않은 여러 프로그램을 동시에 관리"

내 생각: 어딘가에 각 프로그램의 진행사항을 저장해두는 공간이 있겠네?

* + 1. **실시간(real-time) 운영체제**

원하는 시간 내에 프로그램의 결과를 얻을 수 있는 운영체제 우선순위가 높은 작업을 우선 처리할 수 있는 기법 활용 예시

미사일 제어, 증권거래 관리 시스템

* + 1. **분산(distributed) 운영체제**

2개 이상의 컴퓨터 시스템이 "네트워크"로 서로 연결되어 "서로의 자원을 이용"하는 "분산 시스템을 관리"하기 위한 운영체제 요약 [12]

1. **운영체제는 컴퓨터 시스템의 자원을 관리하고 컴퓨터 프로그램이 동작하기 위한 서비스를 제공하는 프로그램들의 모음으로 대표 적인 시스템 소프트웨어이다.**
2. **커널 모드는 하드웨어를 직접 제어할 수 있는 CPU의 명령어를 사용할 수 있는 모드이고, 사용자 모드는 하드웨어를 직접 제어할 수 있는 CPU의 명령어를 사용할 수 없는 모드이다.**
3. **응용프로그램은 사용자 모드에서 동작하기에, 체제에 요청해야 한다.**

하드웨어에 대한 제어가 필요한 경우 "시스템 호출"을 통해 필요한 서비스를 운영

1. **운영체제의 주요 구성요소에는 프로세스 관리자, 메모리 관리자, 장치 관리자, 파일 관리자가 있다.**
2. **운영체제의 대표적인 유형으로는 일괄처리 운영체제, 시분할 운영체제, 실시간 운영체제, 분산 운영체제가 있다.**
3. **프로세스와 쓰레드(1) 개관**

개관

프로그램 실행 -> 운영체제로부터 자원 할당 받은 "프로세스"(실행 중인 프로그램)가 동작 시작 프로세스의 동작

CPU가 그 프로세스의 명령어들을 실행하는 것

CPU도 기억장치나 입출력장치와 마찬가지로 프로세스가 동작하는 데 필요한 자원 프로세스 내에서의 다중처리를 위해 한 프로세스 안에 여러 개의 쓰레드를 두기도 함

학습 목표

1. **프로세스의 개념을 이해한다.**
2. **쓰레드의 개념을 이해한다.**
3. **프로세스와 쓰레드의 관계를 이해한다. 주요 용어**

프로세스

**프로세스 제어 블록(PCB) 프로세스 상태**

디스패치 쓰레드

* + 1. **프로세스의 개요[15] 프로세스(process)란?**

실행 중인 프로그램을 의미

윈도우 운영체제 프로세스 상태

**CPU, 메모리, 디스크, 네트워크, GPU 사용량 표시 프로그램과 달리 "동작"을 하는 "능동적인 개체"**

프로그램?

프로그램 자체

1) 디스크 내 파일로 존재하는, 2) 동작을 하지 않는 정적이며 수동적인 개체 프로그램을 실행시키려면?

운영체제로부터 프로그램이 동작하는 데 필요한 CPU, 메모리, 입출력장치, 파일 등의 자원을 할당받아 동작을 시작 프로세스의 유형

사용자 프로세스

사용자가 실행시킨 프로그램 스풀링

시스템 작업을 위한 시스템 프로세스 프로세스의 동작이란?

CPU가 그 프로세스의 명령들을 실행

CPU도 메모리나 입출력장치와 마찬가지로 프로세스가 동작하는 데 필요한 자원

운영체제는 실행할 준비가 된 프로세스들이 적절히 CPU를 배정받아 효율적으로 작업을 처리할 수 있도록 관리

운영체제

* + - 1. **프로세스 생성, 종료를 위한 다양한 작업**
      2. **프로세스 실행 스케줄링 작업**
      3. **각 프로세스 상태 관리, 전이**
    1. **프로세스의 구성 [16]**

프로세스의 구성 메모리 구조

프로그램 실행에 직접적으로 필요한 코드와 데이터 구분

코드 영역

해당 프로세스의 프로그램 자체

코드 영역에서 명령을 하나씩 꺼내 CPU가 해당 명령을 처리 데이터 영역

프로세스가 사용하는 상수나 변수의 값, 서브프로그램의 호출상태 등 프로그램 실행 시 필요한 데이터를 가진다. 구분

정적 데이터 영역 스택 영역

힙 영역

**프로세스 제어 블록(PCB, Process Control Block)**

운영체제가 프로세스를 관리하기 위해 필요한 정보들로 이루어짐

프로세스 관리를 위해 운영체제는 각 프로세스마다 프로세스 제어블록(PCB)를 두고서 여기에 해당 프로세스의 정보를 보 관한다.

프로세스 제어 블록의 주요 항목

# 프로세스 번호(Process IDentification : PID)

**프로세스의 구분 기준이 되는 식별자(identification) 프로세스 상태**

**실행, 준비 등 프로세스의 현재 상태 프로그램 카운터 (Program Counter: PC)**

**프로세스 수행을 위해 다음에 실행할 명령의 주소 레지스터(register)**

CPU의 레지스터 정보를 저장. 실행상태에 다른 상태로 전이되는 경우 나중에 다시 실행상태로 전이될 때 복구시켜 프 로세스의 정확한 수행을 계속 할 수 있도록 함.

메모리 관리 정보

**가상주소와 실주소의 사상(mapping) 정보, 기준 레지스터(base register)와 경계 레지스터(bound register) 등 의 정보**

프로세스 우선순위

운영체제가 스케줄링 시 어떤 프로세스를 선택할 것인가를 결정하는 데 필요한 프로세스의 우선순위 정보 회계정보

CPU 사용시간, 프로세스의 시스템 존재시간, 메모리 사용량, 보조기억장치 사용량, 그리고 기타 시스템 프로그램의 사 용 실태 등

운영체제는 보통 여러 프로세스를 동시에 관리하며 번갈아 실행(시분할 운영체제) 실행 중이던 프로세스의 여러 정보를 그 프 로세스의 프로세스 제어 블록에 저장한 후 나중에 이 프로세스를 다시 실행하게 되면, 프로세스 제어 블록에 저장된 정보를 이 용하여 이어서 실행

* + 1. **프로세스 상태 관리 [18] 프로세스의 상태(X5) : 생준실대종 생성상태**
* **처음 작업이 시스템에 주어진 상태**
* **운영체제 해당 작업에 대한 PCB 생성, 작업 큐에 넣음**

**- 프로세스 번호(PID) 결정**

-프로세스 메모리 구조도 생성되면 준비상태로 전이 준비상태

프로세스의 실행 준비가 된 상태로 CPU 할당을 기다리는 상태

**프로세스가 준비 큐(ready queue)에 머물고 있다가 스케줄러에 의해 선택되면 CPU를 할당받으며 실행상태로 전이 디스패치(dispatch)**

준비 큐에 있는 프로세스에 CPU를 할당하는 과정 실행상태

CPU가 프로세스의 명령들을 처리하는 상태

프로세스가 처리되다가 만약 스케줄러가 준비 큐에서 다른 프로세스를 선택하게 되면? 실행상태의 프로세스는 CPU를 회수당하며 준비상태로 전이된다.

스케줄링 알고리즘에 따라

1. **CPU 할당시간이 만료된 경우**
2. **더 높은 우선순위의 프로세스가 준비 큐에 들어오는 경우 실행상태의 프로세스가 대기상태로 전이하는 경우**
3. **입출력(I/O)을 요구하는 작업을 만남**
4. **페이지 교환을 요구하는 작업을 만남 대기상태**

프로세스가 I/O 작업이 끝날 때까지 또는 특정 자원을 할당받을 때까지 보류되는 상태 재개 조건이 만족되면 준비상태로 전이됨

재개 조건

* 1. **I/O 작업이 끝났다거나**
  2. **자원을 할당받았다는 이벤트가 발생하면 만족**

종료상태

프로세스가 더 이상 실행되지 않도록 끝난 상태

프로세스에 할당되었던 각종 자원은 운영체제가 회수하게 됨 종료 상태로 전이되는 경우

1. **실행상태의 프로세스가 모든 처리를 완료**
2. **운영체제가 에러 발생을 감지하는 등의 이유로 프로세스를 강제로 종료시킨 경우**
   * 1. **부모 프로세스와 자식 프로세스 프로세스를 생성하는 방법**

사용자가 프로그램 직접 실행

한 프로세스가 다른 프로세스를 생성 프로세스 생성 시스템 호출

**프로세스가 다른 프로세스를 생성하기 위한 방법 부모 프로세스(parent process)**

**시스템 호출을 하는 프로세스 자식 프로세스(child process)**

시스템 호출을 통해 새로 생성된 프로세스

프로세스 생성 시스템 호출방법은 운영체제마다 다를 수 있다.

**UNIX와 Linux fork() 시스템 호출**

자식 프로세스는 부모 프로세스의 복제본

부모 프로세스의 메모리 구조를 복사하여 자식 프로세스의 메모리 구조로 이용 프로세스 제어블록도 프로세스 번호를 제외하고 그에 상응하게 만들어짐

부모 프로세스와 자식 프로세스 모두 프로그램 카운터가 가리키는 곳, 다음에 수행될 위치는? fork()의 다음 명령. (프로그램 카운터가 가리키는 곳)

자식 프로세스가 부모 프로세스와는 다른 프로그램을 실행시키기 원한다면? exec() 시스템 호출을 이용하여 원하는 프로그램 실행시킬 수 있다.

자식 프로세스는 복제되었던 메모리 구조와 프로세스 제어 블록의 내용을 새로운 프로그램의 내용으로 대체하게 된다.

# Windows CreateProcess()

처음부터 새로운 프로그램으로 자식 프로세스를 생성 (linux와의 차이점)

**Linux, Unix와의 공통점**

생성된 자식 프로세스의 프로세스 번호가 부모 프로세스에 전달되는 것은 동일하다. 프로세스가 종료되는 방법

프로세스의 마지막 명령이 실행을 마쳐 모든 처리를 완료하고 정상적으로 종료되는 경우 부모 프로세스에 의해 자식 프로세스가 강제로 종료되는 경우

부모 프로세스는 자식프로세스 번호를 이용하여 프로세스 종료 시스템 호출을 통해 자식 프로세스를 종료시킬 수 있다. 부모 프로세스가 종료되는 경우 운영체제에 의해 그 자식 프로세스가 모두 종료될 수 있음

자식 프로세스가 종료되면?

실행 결과를 부모 프로세스에게 돌려준다.

자식 프로세스의 각종 자원 : 즉시 회수되더라도

자식 프로세스의 PCB : 부모 프로세스가 결과를 받을 때까지 사라지지 않고 유지됨

1. **프로세스와 쓰레드 (2)**

**2.2. 쓰레드 프로세스**

하나의 프로그램을 실행하기 위한 기본 단위 자원 소유의 단위인 동시에 디스패칭의 단위 전통적인 프로세스

하나의 메모리 구조를 갖고 그 안의 코드 영역에 대해 하나의 제어흐름을 갖는다.

하나의 프로세스는 코드 영역, 정적 데이터 영역, 스택 영역, 힙 영역을 하나씩 가지며, 프로세스 제어 블록도 하나를 가진다. 프로그램 카운터(PC)를 포함한 레지스터의 값과 파일 등 할당받은 자원을 하나씩 갖는다.

전통적인 프로세스가 프로세스 내에서 다중처리가 불가능한 이유

프로세스는 매번 프로그램 카운터가 가리키는 명령을 처리하는데, 하나의 프로세스는 하나의 프로그램 카운터만 유지하므 로 제어 흐름을 하나만 갖는다.

**2.2. 쓰레드(2) 쓰레드(thread)**

프로세스 내에서의 다중 처리를 위해 제안된 개념

자원 소유의 단위는 프로세스로 두고, 디스패칭의 단위는 쓰레드로 둔다. 특징

하나의 프로세스 내에는 하나 이상의 쓰레드를 두어 제어 흐름이 하나 이상 될 수 있도록 한다.

(b)는 하나의 프로세스 내에 여러 개의 쓰레드가 있고 각 쓰레드마다 프로그램 카운터(PC)가 있어 쓰레드별로 디스패칭이 가능하다.

프로세스에서 '실행'의 개념만 분리

각 쓰레드는 실행에 필요한 최소한의 정보만 가지며, 나머지 정보는 프로세스에 두고서 다른 쓰레드와 공유한다. 각 쓰레드는 프로그램 카운터를 포함한 레지스터의 값과 스택 영역을 갖는다.

각 쓰레드마다 별도로 점유하는 영역

**2.2. 쓰레드 (3)**

프로그램 카운터

디스패치되어 처리할 명령의 주소를 가짐 레지스터

실행과 관련된 정보를 가짐 스택 영역

실행 시 서브프로그램의 호출상태와 관련된 정보를 저장 상태정보

쓰레드마다 현재 상태가 다를 수 있음

**하나의 프로세스가 다중 쓰레드(멀티 쓰레드, multithread)로 구성된 경우 장점**

컴퓨터 시스템에 여러 개의 CPU가 있는 경우 / 하나의 CPU에 여러 개의 코어(core)가 있는 경우 다중 쓰레드를 병렬로 처리할 수 있다.

만약 다중 쓰레드 각각의 역할이 계산을 위한 쓰레드, 사용자 입력을 위한 쓰레드, 백업을 위한 쓰레드 등 처리속도가 서롣 ㅏ른 작업 을 전담하도록 되어 있는 경우라면

속도가 느린 작업 때문에 다른 작업들이 기다릴 필요 없이 효율적으로 처리 가능하다.

요약

1. **프로세스는 실행 중인 프로그램을 의미하며, CPU, 메모리, 파일, 입출력장치 등 실행에 필요한 자원이 할당된다.**
2. **프로세스 제어 블록은 프로세스를 명시해 주는 다양한 내용을 포함하고 있다.**
3. **프로세스는 생성, 준비, 실행, 대기, 종료의 다섯 상태 중 하나로 존재하며, CPU의 스케줄링, 입출력 대기 등에 따라 준비, 실행, 대기 등으로 상태가 변화되며 동작한다.**
4. **쓰레드는 프로세스에서 실행의 개념만 분리한 것으로 디스패칭의 단위이다.**
5. **하나의 프로세스 내에는 하나 이상의 쓰레드가 있을 수 있어, 다중 쓰레드를 생성하여 프로세스 내에서 다중처리를 할 수 있다.**
6. **프로세스 스케줄링 개관**

운영체제 : 프로세스가 CPU를 적절히 배정받아 작업을 효율적으로 처리할 수 있도록 관리 학습 목표

1. **프로세스 스케줄링의 개요와 정책을 이해한다.**
2. **스케줄링 성능평가 기준을 이해한다.**
3. **다양한 스케줄링 기법을 이해한다. 주요 용어**

비선점 스케줄링 정책 선점 스케줄링 정책 문맥 교환

**대기시간 반환시간 FCFS**

# SJF SRT RR HRN

다단계 피드백 큐

* 1. **프로세스 스케줄링 프로세스 스케줄링**

여러 개의 프로세스가 주어졌을 때, 어떤 순서대로 프로세스를 처리할지 운영체제가 결정하는 것

* + 1. **스케줄링 단계[28]**

운영체제마다 스케줄링 단계 수가 다를 수 있음. (상위, 중간 단계 없이 하위 단계만 존재할 수 있음) 상위단계 스케줄링 = 장기 스케줄링

* + - 1. **시스템에 들어와 작업 큐(queue)에 있는 작업을 선택하여 2) 프로세스를 생성한 후 3) 프로세스 준비 큐에 전달 시스템 자원 효율적 이용이 기준. 다음 작업 균형있게 선택하도록 작업 순서 결정**

**입출력(I/O) 중심 작업 연산 중심 작업**

하위단계 스케줄링 = 단기 스케줄링

1. **준비 큐 프로세스를 선택하여사용 가능한 CPU를 할당 2) dispatch된 프로세스는 실행상태가 되어 처리 하위단계 스케줄링의 수행 주체는?**

**디스패처(dispatcher) 특징**

단기 스케줄링은 빈번하게 이뤄지므로 디스패처는 늘 메모리에 상주 중간단계스케줄링 = 중기 스케줄링

프로세스를 일시적으로 메모리에서 제거하여 중지시킴 중지된 프로세스에 다시 메모리를 할당하여 활성화시킴

시스템에 대한 단기적 부하 조절 (시스템 과부하시 어떤 프로세스 제거할지 결정)

* 1. **프로세스 스케줄링 (2)**
     1. **스케줄링의 목표 [29]**

기본적인 목표 공정성

모든 프로세스가 적정 수준에서 CPU 작업을 할 수 있게 하는 것 균형성

시스템의 자원들이 충분히 활용될 수 있게 하는 것 운영체제별 목표

일괄처리 운영체제

처리량의 극대화, 반환시간의 최소화, CPU 활용의 극대화 시분할 운영체제

빠른 응답시간, 과다한 대기시간 방지 실시간 운영체제

처리기한을 맞추는 것 용어 정의

**처리량 (throughput)**

**주어진 시간에 처리한 프로세스 수 반환시간 (turnaround time)**

**프로세스 생성 시점부터 종료 시점까지의 소요시간 응답시간 (response time)**

**요청한 시점부터 반응이 시작되는 시점까지의 소요시간 대기시간 (waiting time)**

프로세스가 종료될 때까지 준비 큐에서 기다린 시간의 합

* 1. **프로세스 스케줄링 (3)**
     1. **스케줄링 정책**

스케줄링 목표들은 각각 서로 상반된 경우가 있어서 모두를 한 번에 충족시키는 정책은 세우기가 어렵다. 처리량 극대화

방법

단위시간당 완료되는 프로세스 수가 많아지도록 수행시간이 짧은 프로세스를 우선 처리 오버헤드 줄이기 위해 인터럽트 없이 프로세스를 수행

반대급부

(처리량은 극대화되었지만) 수행시간이 긴 프로세스는 대기시간, 반환시간이 길어진다. 이후 설명은 모두 하위단계 스케줄링으로 가정

1. **선점 스케줄링 정책 (preemptive scheduling policy)**

실행 중인 프로세스에 인터럽트(interrupt)를 걸고 다른 프로세스에 CPU를 할당할 수 있는 스케줄링 방식 높은 우선순위의 프로세스를 우선 처리

특성

결과 예측 가능 : 실시간 시스템에서 선점 스케줄링 정책을 이용하면 필요한 프로세스가 CPU를 선점 시분할 시스템에서도 빠른 응답시간을 유지하기 위해 필요

단점

**문맥 교환(context switching)**

1) CPU가 현재 실행하고 있는 프로세스의 문맥을 프로세스 제어 블록(PCB)에 저장하고, 2) 다른 프로세스의 프로세 스 제어 블록으로부터 문맥을 복원하는 작업

**문맥 교환에 따른 오버헤드(overhead) 발생 : 문맥 교환에 사용되는 일정부분의 시간, 자원 고려**

1. **비선점 스케줄링 정책 (nonpreemptive scheduling policy)**

실행 중인 프로세스를 바로 준비상태로 전이시킬 수 없는 스케줄링 방식

CPU 할당 받아 실행중인 프로세스는, 대기 또는 종료상태로 전이될 때까지 계속 실행상태 장점

강제적인 문맥 교환이 없어 오버헤드가 발생하지 않는다. 단점

오랜 시간이 걸리는 프로세스가 실행 중이라면 실행시간이 짧은 프로세스가 오래 기다리는 경우가 발생할 수 있음 프로세스 스케줄링 (2)

* 1. **프로세스 스케줄링 (4)**
     1. **스케줄링의 평가 기준[31]**

일반적인 스케줄링 알고리즘 성능 평가 기준 평균대기시간

각 프로세스가 수행이 완료될 때까지 준비 큐에서 기다리는 시간의 합의 평균값 평균반환시간

각 프로세스가 생성된 시점 (여기서는 준비 큐에 들어온 시점과 동일한 것으로 가정)부터 수행이 완료된 시점까지의 소요 시간의 평균값

예제

프로세스 A가 준비 큐에 들어온 시각이 0, 큐에서 나와 수행을 시작한 시각이 2, 다시 큐로 돌아가지 않고 수행을 완료한 시간이 4라면?

A의 대기시간 2-0 = 2

A의 반환시간 4-0 = 4

주의

원래 반환시간에는 프로세스 생성 시간과 대기상태에서의 시간도 모두 포함된다.

**단, 여기서는**

**1) 프로세스 생성 시간을**

준비 큐에 들어온 시점으로 보고 2) 입출력 인터럽트 등으로 대기상태로 전이되는 경우는 없는 것으로 가정한다. => 이 경 우 한 프로세스의 반환시간은 대기시간과 실행상태의 시간의 합과 같다.

실행 상태의 시간은 결국 수행에 필요한 CPU 사이클 시간이므로, 반환시간에서 CPU 사이클 시간을 빼면 대기시간이 된 다.

* 1. **스케줄링 알고리즘**

준비 큐에 머물고 있는 프로세스는 스케줄러에 의해 선택되어 디스패치된다.

# FCFS 스케줄링 (First Come First Served)

* + - 1. **프로세스는 준비 큐에 도착한 순서에 따라 디스패치된다. 2) 비선점 방식**

한 프로세스가 CPU를 차지하면 그 프로세스의 수행이 완료된 후에 그 다음 프로세스가 CPU를 차지하고 수행된다. 예제

특징

겉으로는 공정한 것처럼 보임 단점

* + - * 1. **실행시간이 짧은 작업이 긴 작업을 기다리게 된다.**
        2. **중요한 프로세스가 나중에 수행될 수도 있다.**
        3. **스케줄링 알고리즘은 프로세스들의 도착순서에 따라 평균 반환시간이 크게 변하는 단점도 있다. 만약 A, B, C, D 순서가 아닌 C, D, B, A 순으로 입력되었을 때 평균 대기시간과 평균 반환시간은?**

예제2 프로세스들의 도착시간이 다른 예시

주로 다른 방법과 결합하여 쓰인다. 우선순위가 같은 경우에 FCFS를 적용한다.

# SJF 스케줄링 [35] (Shortest Job First)

(준비 큐에서 기다리는 프로세스 중) 1) 예상 실행시간이 가장 짧은 것 먼저 디스패치하여 실행 2) 비선점 방식 실행할 프로세스의 CPU 사이클이 미리 주어져야만 적용 가능

예제 단점

실행예정시간의 길이를 사용자의 추정치에 의존 -> 실제로는 먼저 처리할 작업의 CPU 시간을 예상할 수 없다.

비선점 방식이기에 새로 들어온 짧은 작업이 실행 중인 긴 작업을 기다리게 되기도 하기에 시분할 운영체제나 실시간 운영체 제에는 적합하지 않다.

* 1. **스케줄링 알고리즘 (2) [36]**

# SRT 스케줄링 (Shortest Remaining Time)

* + - 1. **SJF 알고리즘의 선점 방식 2) 준비 큐에서 기다리고 있는 프로세스 중 남은 실행시간이 가장 짧다고 예상되는 것을 먼저 디스 패치하여 실행**

새로 들어온 프로세스가 즉시 디스패치 되는 경우?

새롭게 준비 큐에 들어온 프로세스의 예상 실행시간이 현재 실행상태에 있는 프로세스의 남은 실행시간보다 짧은 경우 조건

SJF 알고리즘 처럼 실행할 프로세스의 CPU 사이클이 미리 주어져야만 적용 가능 예제

**SJF 스케줄링과 SRT 스케줄링 비교 [38]**

**비선점 vs 선점**

SRT가 SJF보다 평균 대기시간이나 평균 반환시간에서 효율적 오버헤드 발생: 1) 프로세스 실행시간 추적 2) 선점을 위한문맥교환

1. **실행 중인 프로세스가 거의 끝날 무렵 실행시간 추정치가 매우 작은 프로세스가 새로 입력되는 경우**
2. **실행 중인 프로세스가 아직 많은 처리과정을 거쳐야 하는데, 이보다 실행시간 추정치가 약간 더 작은 프로세스가 입력 되는 경우**
   * 1. **RR 스케줄링 (Round Robin) 스케줄링**
        1. **프로세스 도착 순서대로 디스패치 2) but 정해진 시간 할당량 (또는 시간 간격)에 의해 실행 제한 3) 선점형**

디스패치된 프로세스가 시간 할당량 안에 종료하지 못하면 해당 프로세스는 준비상태로 전이되어 준비 큐의 마지막에 들어간다. 예제[39]

**가정: 시간 할당량 4**

가정2: (시각 4에 프로세스 C도 도착하고 프로세스 A도 시간 할당량이 끝나는데, 이 경우 새로 도착한 프로세스가 준비 큐에 먼저 들어가는 것으로 임의로 가정한다.)

정답 유도 장점

시분할 운영체제에 적합하다.

적절한 시간 할당량을 사용하면 프로세스들이 CPU를 독점하지 않고 공평하게 이용될 수 있게 한다. 시간 할당량이 0.1초이고 10개의 프로세스가 활성화되어 있다면?

모든 프로세스는 1초 이내의 응답시간을 갖게 된다. 시간 할당량에 따라 RR 스케줄링 성능이 달라진다.

시간 할당량이 너무 큰 경우

**FCFS와 동일**

시간 할당량이 너무 작은 경우

너무 많은 문맥 교환으로 인한 오버헤드 적절한 시간 할당량

문맥 교환에 걸리는 시간보다 수백 배에서 수천 배 정도 커서 오버헤드 비율이 크지 않아야 하며 프로세스가 한 번에 처리할(사용자에게 피드백을 줄 수 있는) 양을 처리할 수 있는 정도가 되어야 함

* 1. **스케줄링 알고리즘 (4) [41]**

# HRN 스케줄링 (또는 HRRN, Highest Response Ratio Next)

준비 큐 프로세스 중 1) 응답비율이 가장 큰 것 먼저 디스패치하여 실행 2) 비선점 방식 응답비율 계산 공식

응답비율은 예상실행시간이 짧을수록, 그리고 대기시간이 길수록 커진다. 예시

모든 프로세스가 동시에 준비 큐에 들어온다면? 그 중 예상실행시간이 가장 짧은 프로세스 선택

예상실행시간이 모두 동일한 프로세스이지만 준비 큐에 들어온 시각이 모두 다르다면? 그 중 대기시간이 가장 긴 프로세스가 선택

조건

**SJF, SRT 스케줄링처럼 실행할 프로세스의 CPU 사이클이 미리 주어져야만 적용 가능**

예제 특징

**SJF 스케줄링의 단점 보완**

SJF 스케줄링 : 예상실행시간이 긴 프로세스가 준비 큐에 먼저 들어왔더라도 이후에 예상 실행시간이 짧은 프로세스가 계 속해서 준비 큐에 들어오면 우선순위에서 밀림

HRN 스케줄링 : 예상실행시간이 긴 프로세스라 하더라도 대기시간이 길어지면 응답비율도 커져서 실행시간이 짧은 프로 세스가 나중에 들어오더라도 우선순위에서 밀리지 않을 수 있다.

* 1. **스케줄링 알고리즘 (5) [43]**
     1. **다단계 피드백 큐 스케줄링**
        1. **입출력 중심인 프로세스와 연산 중심인 프로세스의 특성에 따라 서로 다른 시간 할당량을 부여 2) 선점 방식 스케줄링**

n개의 단계가 있는 경우

각 단계마다 하나씩의 준비 큐가 존재 단계 k는 단계 k+1에 피드백을 줌

단계가 커질수록 시간 할당량은 커지는 형태 상세 절차

**1단계**

* + - * 1. **단계 1의 준비 큐에 프로세스들이 들어간다.**
        2. **단계 1의 시간 할당량만큼 실행된다.**
        3. **시간 할당량보다 짧은 시간에 프로세스가 완료되면? 종료 상태로 전이**
        4. **입출력 같은 이벤트가 발생하면?**

CPU를 양보하고 대기상태로 갔다가 다시 준비상태가 될 때는 현재와 동일한 단계 (즉, 단계 1)의 준비 큐에 배치된다.

* + - * 1. **시간 할당량을 다 썼지만 프로세스가 종료되지 못했다면? 다음 단계(즉, 단계2)의 준비 큐로 이동 배치 된다.**

마지막 단계 n에서도 단계 n의 시간 할당량만큼 실행 후 종료되지 못한 경우 계속하여 동일한 단계 n의 큐에 배치된다

단계 n만 보면 RR 스케줄링 방식처럼 동작하게 된다.

주의

[그림 3-10]에 CPU가 여러 번 나오지만 이는 단계를 구분하기 위한 표현일 뿐 실제 CPU는 하나로 가정한다. 즉, 하나의 단계에서 디스패치가 일어나면 다른 단계에서는 디스패치를 할 수 없다.

단계 k의 준비 큐에 있는 프로세스가 디스패치 되려면?

단계 1부터 단계 k-1까지 모든 준비 큐가 비어 있어야만 한다.

Q. 그럼 끊임없이 다단계 피드백 큐에 프로세스가 들어오면 어떻게 되는거지? 하나의 단계에서 디스패치가 일어나면 다른 단계에 서는 디스패치를 할 수 없다.

즉, 단계가 커질수록 프로세스가 디스패치되는 빈도는 줄어들 수 있다.

대신 단계가 커질수록 시간 할당량도 커지므로 프로세스가 한 번 디스패치되면 더 오랜 시간 실행할 수 있다. 연산 위주의 프로세스

시간 할당량 안에 종료되지 못하여 점점 단계가 커지게 되고, 입출력 위주의 프로세스는 시간 할당량보다 짧은 실행 이 유지되어 작은 단계를 유지하게 된다.

**요약 [46]**

1. **프로세서의 스케줄링을 위해 상위 단계, 하위단계 및 중간단계 스케줄링이 사용된다.**
2. **선점 스케줄링 정책은 실행 중인 프로세스에 인터럽트를 걸고 다른 프로세스에 CPU를 할당할 수 있는 스케줄링 방식이고, 비선 점 스케줄링 정책은 실행 중인 프로세스를 바로 준비상태로 전이시킬 수 없는 스케줄링 방식이다.**
3. **FCFS 스케줄링은 준비 큐에 도착한 순서에 따라 디스패치하는 비선점 방식의 스케줄링 알고리즘이다.**
4. **SJF 스케줄링은 준비 큐에서 기다리는 프로세스 중 실행시간이 가장 짧다고 예상되는 것을 먼저 디스패치하는 비선점 방식의 스 케줄링 알고리즘이다.**
5. **SRT 스케줄링은 준비 큐에서 기다리는 프로세스 중 남은 실행시간이 가장 짧다고 예상되는 것을 먼저 디스패치하는 선점 방식의 알고리즘이다.**
6. **RR 스케줄링은 프로세스가 도착한 순서대로 프로세스를 디스패치하지만 정해진 시간 할당량에 의해 실행을 제한하는 선점 방식 의 스케줄링 알고리즘이다.**
7. **HRN 스케줄링은 준비 큐에서 기다리는 프로세스 중 응답비율이 가장 큰 것을 먼저 디스패치하여 실행하는 비선점 방식의 스케 줄링 알고리즘이다.**
8. **HRN 스케줄링의 응답비율은 예상실행시간이 짧을수록, 그리고 대기시간이 길수록 커진다.**
9. **다단계 피드백 큐 스케줄링은 입출력 위주의 프로세스가 연산 위주의 프로세스보다 우선권을 갖도록 하는 선점 방식의 알고리즘 이다.**