프로세스

- 주 기억 장치에 수록되어 있으면서 실행 / 대기 중인 프로그램

- PCB(Process Control Block)을 지닌 프로그램

- 프로그램 카운터를 지닌 프로그램

- 능동적 개체(Entity)로 순차적으로 수행하는 프로그램

- 하나의 프로세스는 cpu가 수행하는 작업 단위

----------------------------------

* 컴퓨터에서 연속적으로 실행되고 있는 컴퓨터 프로그램
* 스케줄링의 대상이 되는 작업(task)이라는 용어와 같은 의미로 사용
* 여러 개의 프로세서를 사용하는 것을 멀티프로세싱이라고 하며, 같은 시간에 여러 개의 프로그램을 띄우는 시분할 방식을 멀티태스킹이라고 함
* 프로세스 관리는 운영체제의 중요한 부분

프로그램 : 하드 디스크 등에 저장되어 있는 실행 코드

프로세스 : 프로그램을 구동하여 프로그램 자체와 프로그램의 상태가 메모리 상에서 실행되는 작업 단위를 지칭

ex) 하나의 프로그램을 여러 번 구동하면 여러 개의 프로세스가 메모리 상에서 실행됨

프로세스상태

// 이미지 추가

1) 생성(Create) : 프로세스가 생성되는 중

2) 실행(Running) : 프로세스가 CPU를 차지하여 명령어들이 실행되고 있음

3) 준비(Ready) : 프로세스가 CPU를 사용하고 있지는 않지만, 언제든지 사용할 수 있는 상태

4) 대기(Waiting) / 보류(Block) : 프로세스가 입출력 완료, 시그널 수신 등 어떤 사건을 기다리고 있는 상태

5) 종료(Terminated) : 프로세스의 실행이 종료

프로세스의 상태 전이

하나의 프로그램이 실행되면 그 프로그램에 대응되는 프로세스가 생성되어 준비 리스트의 끝에 들어감. 준비 리스트 상의 다른 프로세스들이 CPU를 할당 받아 준비 리스트를 떠나면, 그 프로세스는 점차 준비 리스트의 앞으로 나가게 되고 언젠가 CPU를 사용할 수 있게 됨

1) 디스패치(Dispatch) : 준비 리스트의 맨 앞에 있던 프로세스가 CPU를 점유하게 되는 것(준비상태 -> 실행 상태) 표시 : dispatch ( processname ) : ready -> running

2) 보류(Block) : 실행 상태의 프로세스가 허가된 시간을 다 쓰기 전에 입출력 동작을 필요로 하는 경우 프로세스는 CPU를 스스로 반납하고 보류 상태로 넘어감 표시 : block ( processname ) : running -> blocked

3) 깨움(Wakeup) : 입출력 작업 종료 등 기다리던 사건이 일어났을 때 보류 상태에서 준비 상태로 넘어가는 과정 표시 : dispatch ( processname ) : block -> ready

4) 시간제한(Timeout) : 운영체제는 프로세스가 프로세서를 계속 독점해서 사용하지 못하게 하기 위해 clock interrupt를 두어서 프로세스가 일정 시간 동안만(시분할 시스템의 time slice) 프로세서를 점유할 수 있게 함 표시 : timeout ( processname ) : running -> ready

프로세스 구성 요소

코드 영역 : 프로그램의 코드 자체, 프로그램은 실행되기 전에 주기억장치에 CPU가 해석할 수 있는 이진 코드 상태로 주기억장치에 수록되는 영역

데이터 영역 : 프로그램의 전역 변수(Global Variable)나 정적 변수(Static Variable)의 할당

스택 영역 : 지역 변수(Local Variable) 할당과 함수 호출 시 전달되는 인수(Argument) 할당

힙 영역 : 동적 변수 할당

프로세스 제어 블록(PCB)

- 정의 : 프로세스에 관한 모든 정보를 수록하고 있는 장소(데이터베이스)

- 내용 : 프로세스의 현재 상태(실행, 준비완료, 대기 등), 프로세스의 고유 이름(Identifier), 프로세스의 우선 순위, 프로세스가 적재된 기억장치의 주소를 가지는 포인터, 할당된 자원(장치 등)을 가리키는 포인터, 중앙처리장치의 각종 레지스터 상태를 저장하기 위한 공간

- 모든 프로세스는 각기 고유의 PCB 지님

- 스케줄러 내에 있는 트레픽 제어기가 프로세스의 상태를 파악함

프로세스 스케줄링

* 목적
* 공정성 : 자원 배분의 공정성
* 처리 능력의 최대화 : 자원의 효율적 사용과 처리 능력의 극대화
* 응답 시간의 최소화 : 응답시간 및 처리 시간의 최소화
* 예측 가능
* 오버헤드의 최소화
* 자원 사용의 균형 유지 : 특정 프로세스에게 과다한 자원 할당 회피
* 응답과 이용 간의 균형 유지
* 실행의 무한한 지연을 피할 것
* 우선순위제의 실시
* 주요 자원들을 차지하고 있는 프로세스에게 우선권을 부여
* 좀 더 바람직한 동작을 보이는 프로세스에게 더 좋은 서비스를 제공
* 과중한 부하를 완만히 감소함

프로세스 스케줄링 : 단계별 분류

중앙처리 장치 스케줄링의 세가지 단계

1) 상위 단계 스케줄링(High level scheduling)

작업 / 승인(Job / Admission) 스케줄링이라고도 불림

어떤 작업에게 시스템의 자원들을 차지할 수 있도록 할 것인가를 결정

2) 중간 단계 스케줄링(Intermediate level scheduling)

원활한 시스템 작동과 시스템 전체의 성능 향상을 위하여 시스템 부하(Load) 내에서 짧은 순간에 프로세스들에 대한 일시적인 활동의 중단 및 재개를 수행함

3) 하위 단계 스케줄링(Low level scheduling)

어떤 준비완료 프로세스에게 중앙처리장치를 할당할 것인가를 결정하고 실제로 중앙처리장치를 이 프로세스에게 할당해 줌

프로세스 스케줄링 : 방법 / 환경별 분류

1) 선점 / 비선점 스케줄링

- 비선점 스케줄링 : 하나의 프로세스에 중앙처리장치가 할당되면 그 프로세스의 수행이 끝날 때까지 중앙처리장치는 그 프로세스로부터 빠져나올 수 없음, 응답시간을 쉽게 예측할 수 있음

- 선점 스케줄링 : 하나의 프로세스가 중앙처리장치를 차지하고 있을 때 다른 프로세스가 현재 수행 중인 프로세스를 중지시키고 자신이 중앙처리장치를 차지할 수 있음. 높은 우선 순위를 가진 프로세스들이 빠른 처리를 요구하는 시스템에 유리함. 문맥 교환 등으로 인한 오버헤드가 발생됨. 많은 프로세스들이 준비 상태로 존재해야만 효과적

2) 우선순위 스케줄링

각 프로세스에게 우선순위를 부여하여 우선순위가 높은 순서대로 처리하는 방법

- 정적 우선순위 기법 : 처음 프로세스 생성시 우선 순위 부여. 상대적으로 오버헤드는 적으나 주위 여건의 변화에 적응하지 않고 우선순위를 바꾸지 않음

- 동적 우선순위 기법 : 실행 시 필요에 따라 우선순위 재구성. 복잡하고 오버헤드가 많지만 시스템의 응답도를 증가시켜 효율성을 높일 수 있음

3) 기한부 스케줄링

작업들이 명시된 시간이나 기한내에 완료되도록 계획. 시간 내 결과가 완성되면 효용성이 높으나 그렇지 못하면 쓸모가 없음. 따라서 실시간 시스템과 같은 제한된 응답 분야에서는 매우 유효함

- 경성 실시간 시스템 : 정한 시간 내에 완료할 수 있도록 해주는 강한 형태의 실시간 시스템

- 연성 실시간 시스템 : 시간적 제한이 다소 약한 형태의 실시간 시스템

4) 다중 프로세서 스케줄링

프로세서가 여러 개의 경우의 스케줄링

프로세스 스케줄링 알고리즘

1) FCFS(First Come First Served)

비선점 스케줄링 방법

프로세스들은 대기 큐에 도착한 순서에 따라 중앙처리장치를 할당

빠른 응답을 요하는 대화식 방식에는 적합하지 않음

호위 효과 : 첫 번째 프로세스가 끝날 때까지 매우 긴 시간을 기다리게 되는 것

2) SJF(Shortest Job First)

비선점 방식

수행시간이 가장 짧은 것을 먼저 수행하게 함

빠른 응답을 요하는 대화식 방식 또는 시분할 방식에는 적합하지 않음

3) Priority

우선순위가 각 프로세스에게 주어지며, 중앙처리장치는 가장 높은 우선순위를 가진 프로세스로 할당

무한 대기와 기아 현상 문제 존재, 해결책으로 에이징 존재

에이징 : 오랫동안 시스템에서 대기하는 프로세스들의 우선순위를 점진적으로 증가시키는 방법

4) Round Robin

시분할 시스템을 위하여 고안된 선점 스케줄링 방식

각 프로세스는 같은 크기의 중앙처리장치 시간을 할당 받음

할당시간내에 마치지 못하면 준비 리스트의 가장 뒤에 저장됨

대화식 사용자들에게 알맞은 응답시간으로 보장해주어야 하는 시분할 방식의 시스템에 효과적인 방식

5) SRT(Shortest Remaining Time)

SJF 기법에 선점 방식을 도입한 방법

시분할 시스템에서 유용

새로 도착한 프로세스를 포함하여 처리가 완료되는데 가장 짧은 시간이 소요되는 프로세스를 먼저 수행

6) Multi Level Queue

작업들을 여러 그룹으로 나누어 여러 개의 큐를 이용하는 방법

전면 작업 : 대화식 작업

후면 작업 : 일관 처리 작업

7) Multi Level Feedback Queue

짧은 작업에 유리, 입출력 장치를 효과적으로 이용하기 위하여 입출력 위주 작업들에게 우선권을 주며 가능한 빨리 작업의 특성을 알고 그것에 맞게 그 작업을 스케줄링 함

8) HRN(High Response ratio Next)

긴 작업과 짧은 작업 간의 지나친 불평등을 어느정도 보완한 기법

한 작업이 중앙처리장치를 차지하면 그 작업은 완성될 때 까지 실행하며, 대기시간이 고려되어 긴 작업과 짧은 작업 간의 불평등을 어느 정도 완화시킴

쓰레드(Thread)

어떠한 프로그램 내에서, 특히 프로세스 내에서 실행되는 흐름의 단위

일반적으로 한 프로그램은 하나의 스레드를 가지고 있지만, 프로그램 환경에 따라 둘 이상의 스레드를 동시에 실행 할 수 있고, 이를 멀티쓰레드라고 함

쓰레드의 종류

사용자 레벨 쓰레드(User – Level Thread) : 커널 영역의 상위에서 지원되며 일반적으로 사용자 레벨의 라이브러리를 통해 구현, 라이브러리는 스레드의 생성 및 스케줄링 등에 관한 관리 기능 제공. 동일한 메모리 영역에서 스레드가 생성 및 관리되므로 속도가 빠르지만, 여러 개의 사용자 스레드 중 하나의 스레드가 시스템 호출 등으로 중단되면 나머지 모든 스레드 역시 중단되는 단점 존재(커널이 프로세스 내부의 스레드를 인식하지 못하여 해당 프로세스를 대기 상태로 전환시키기 때문)

커널 레벨 쓰레드(Kernel – Level Thread) : 운영체제가 지원하는 쓰레드 기능으로 구현되며, 커널이 쓰레드의 스케줄링 등 관리. 스레드가 시스템 호출 등으로 중단되더라도, 커널은 프로세스 내의 다른 스레드를 중단시키지 않고 계속 실행시킴. 다중 처리기 환경에서 커널은 여러 개의 스레드를 각각 다른 처리기에 할당할 수 있으나 사용자 스레드에 비해 생성 및 관리하는 것이 느림

스레드 데이터

- 기본 데이터 : 자신만의 고유한 스레드 ID, 프로그램 카운터, 레지스터 집합, 스택을 가지며 코드, 데이터, 파일 등 기타 자원은 프로세스 내의 다른 스레드와 공유

- 스레드 특정 데이터 : 하나의 스레드에만 연관된 데이터

스레드의 특성

- 서로 독립적

- 실행 / 종료 순서는 예측 할 수 없음

- 수행을 위해 스케줄링 되고 결과들은 프로세스에게 전달됨

- 프로그램에 있는 쓰레드의 수는 다른 쓰레드에게 알려지지 않음

- 쓰레드는 프로그램의 외부에서는 보이지 않음

- 쓰레드는 서로 독립적이지만, 한 쓰레드가 취한 행동은 프로세스에 있는 다른 쓰레드에 영향을 미침

- 쓰레드는 프로세스의 일부분이기 때문에 프로세스의 자원들을 공유하지만 그 자신의 처리시간과 스택, 레지스터들이 할당

- 한 프로세스가 시스템 콜을 통해 종료되면, 모든 스레드들도 종료됨

- 중량 프로세스 : 하나의 쓰레드를 가진 프로세스

- 경량 프로세스 : 두개 이상의 쓰레드를 가진 프로세스

프로세서

중앙처리장치(CPU)는 명령어를 해석하는 컴퓨터의 한 부분

프로세스 간 동기화 및 통신

병행 프로세스(Concurrent Process)

두 개 이상의 프로세스가 동시에 수행 중 임

독립적 병행 프로세스(프로세스들이 서로 관련 없이 독립적으로 수행함)

협력적 병행 프로세스(다른 프로세스들과의 협력을 통해서 기능을 수행 / 비동기적 수행)

병행 처리를 위하여 제한된 자원을 공유하기 위하여 상호작용이 필요함

프로세스들을 동기화하지 않으면 교착상태, 임계영역 문제, 결과를 예측할 수 없는 상황 등 여러 문제들이 발생하기 때문에 동기화가 필요함

동기화 : 2개 이상의 프로세스에 대한 처리 순서를 결정하는 것

병행 처리의 문제점을 해결하기 위한 사항

한 순간에 하나의 프로세스가 공유 자원을 상호 배타적으로 사용 가능해야 함

한 기능(함수)를 공유해 수행하는 두 프로세스 간의 동기화 문제

자료 교환을 위한 메시지 전달 방식 등의 통신 문제

프로세스들의 실행 순서와는 무관하게 항상 같은 결과를 얻을 수 있어야 하는 문제

교착상태 문제

프로그래밍 언어를 통한 병행 처리 문제

올바른 실행을 검증하는 문제

임계구역

2개 이상의 프로세스가 동시에 접근하면 안되는 공유 자원이 있는 코드 영역

어떤 프로세스가 공유 자원을 접근하고 있는 동안 그 프로세스는 임계구역에 있다고 함

임계구역에 접근한 프로세스는 상호배제를 보장받아야 함

상호배제 : 한 프로세스가 임계영역에서 실행 중 일 때, 다른 어떤 프로세스도 임계영역에서 실행될 수 없도록 하는 것, Mutex Lock을 이용하여 임계구역에 들어가기 전에 Lock 획득, 빠져 나올 때는 Lock 반환

임계구역 문제 해결책을 위한 충족 요건 : 상호배제, 진행, 한정된 대기

하드웨어에 의한 동기화

- 임계구역 문제는 공유변수가 변경되는 동안 인터럽트 발생을 허용하지 않도록 설정함

프로세스 간 통신(Inter-Process Communication, IPC)

- 프로세스들 사이에 서로 데이터를 주고받는 행위 또는 그에 대한 방법이나 경로

- 마이크로 커널과 나노 커널의 디자인 프로세스에 매우 중요함

주요 IPC 방식

1) 파일 : 대부분의 운영 체제

2) 신호 : 대부분의 운영체제, 윈도우와 같은 일부 시스템은 C 런타임 라이브러리에서만 신호를 제공하며 IPC 방식으로 사용하는 것을 지원하지는 않음

3) 소켓 : 대부분의 운영체제

4) 메시지 큐 : 대부분의 운영체제

5) 파이프 : 모든 POSIX 시스템, 윈도우

6) 지명 파이프 : 모든 POSIX 시스템, 윈도우

7) 세마포어 : 모든 POSIX 시스템, 윈도우

8) 공유 메모리 : 모든 POSIX 시스템, 윈도우

9) 메시지 전달(비공유) : MPI 패러다임, 자바 RMI, CORBA, MSMQ, 메일 슬롯, QNX 등에 쓰임

10 ) 메모리맵 파일 : 모든 POSIX 시스템, 윈도우

-> Memory mapped file(MMF)운영체제에서 파일을 다루는 방법 중 하나

+) 컨텍스트 스위칭