1. 스레드

1) 스레드

- 어떠한 프로그램 내에서, 특히 프로세스 내에서 실행되는 흐름의 단위
- 일반적으로 한 프로그램은 하나의 스레드를 가지고 있지만 프로그램 환경에 따라 둘 이상의 스레드를 동시에 실행할 수 있고, 이를 멀티 스레드라고 함

2) 스레드의 종류

- 사용자 레벨 스레드(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 스레드(Kernel - Level Thread)

  운영체제가 지원하는 스레드 기능으로 구현되며, 커널이 스레드의 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

3) 스레드의 특성

- 서로 독립적
- 실행 / 종료 순서는 예측 할 수 없음
- 수행을 위해 스케줄링 되고 결과들은 프로세스에게 전달됨
- 프로그램에 있는 스레드의 수는 다른 스레드에게 알려지지 않음
- 스레드는 프로그램의 외부에서는 보이지 않음
- 스레드는 서로 독립적이지만, 한 스레드가 취한 행동은 프로세스에 있는 다른 스레드에 영향을 미침
- 스레드는 프로세스의 일부분이기 때문에 프로세스의 자원들을 공유하지만 그 자신의 처리시간과 스택, 레지스터들이 할당
- 한 프로세스가 시스템 콜을 통해 종료되면, 모든 스레드들도 종료됨
- 중량 프로세스 : 하나의 스레드를 가진 프로세스
- 경량 프로세스 : 두개 이상의 스레드를 가진 프로세스

2. 멀티 스레드

1) 멀티 스레드

- 여러 개의 스레드가 하나의 프로세서에서 실행되는 상태

2) 장점 및 단점

- 장점

  응답성, 자원공유, 경제성, 멀티프로세서 활용

- 단점

  - 다중 스레드는 캐시나 변환 색인 버퍼와 같은 하드웨어 리소스를 공유할 때 서로를 간섭할 수 있음
  - 하나의 스레드만 실행 중인 경우 싱글 스레드의 실행 시간이 개선되지 않고 오히려 지연될 수 있음
  - 멀티 스레딩의 하드웨어 지원을 위해 응용프로그램과 운영체제 둘 다 충분한 변화가 필요함
  - 스레드 스케줄링은 멀티스레딩의 주요 문제이기도 함

3) 멀티 스레드 모델

- 다대일(Many to One) : 여러 개의 사용자 수준 스레드들이 하나의 커널 스레드(프로세스)로 매핑되는 방식.

  사용자 수준에서 스레드 관리가 이루어짐

- 일대일(One to One) : 사용자 스레드를 각각 하나의 커널 스레드로 매핑시키는 모델

- 다대다(Many to Many) : 여러 개의 사용자 스레드를 여러 개의 커널 스레드로 매핑 시키는 모델

3. 프로세스

1) 프로세스

- 주 기억 장치에 수록되어 있으면서 실행 / 대기 중인 프로그램
- PCB(Process Control Block)와 프로그램 카운터를 지닌 프로그램
- 컴퓨터에서 연속적으로 실행되고 있는 컴퓨터 프로그램
- 프로세스 관리는 운영체제의 중요한 부분
- 문맥 교환(Context Switch) : 하나의 프로세스가 CPU를 사용중인 상태에서 다른 프로세스가 CPU를 사용하도록 하기 위해, 이전의 프로세스의 상태(문맥)을 보관하고 새로운 프로세스의 상태를 적재하는 작업

2) 프로세스 vs 프로그램 vs 프로세서

- 프로세스 : 프로그램을 구동하여 프로그램 자체와 프로그램의 상태가 메모리 상에서 실행되는 작업 단위
- 프로그램 : 하드 디스크 등에 저장되어 있는 실행 코드
- 프로세서 : 마이크로프로세서(microprocessor) 또는 초소형 연산처리장치는 컴퓨터의 중앙처리 장치(CPU)

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

3) 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

4) 프로세스 제어 블록(PCB)

- 프로세스에 관한 모든 정보를 수록하고 있는 장소(데이터베이스)

- 모든 프로세스는 각기 고유의 PCB 지님

5) 프로세스 상태

- 생성(Create), 실행(Running), 준비(Ready), 대기(Waiting) / 보류(Block), 종료(Terminated)

6) 프로세스의 상태 전이

- 디스패치(Dispatch), 보류(Block), 깨움(Wakeup), 시간제한(Timeout)

7) 프로세스 스케줄링

- 목적

공정성, 처리 능력의 최대화, 응답 시간의 최소화, 예측 가능, 오버헤드의 최소화, 자원 사용의 균형 유지, 응답과 이용 간의 균형 유지, 실행의 무한한 지연을 피함, 우선 순위제의 실시, 주요 자원들을 차지하고 있는 프로세스에게 우선권 부여, 과중한 부하 완만히 감소

- 분류

단계별 분류	방법 / 환경 별 분류
상위 단계 스케줄링(High level scheduling) 중간 단계 스케줄링(Intermediate level scheduling) 하위 단계 스케줄링(Low level scheduling)	선점 / 비 선점 스케줄링 우선순위 스케줄링(정적 / 동적) 기한부 스케줄링 다중 프로세서 스케줄링

- 알고리즘

FCFS(First Come First Served)	비 선점 방법으로 프로세스가 대기 큐에 도착한 순서에 따라 중앙처리장치를 할당, 빠른 응답을 요하는 대화식 방식에는 적합하지 않음
SJF(Shortest Job First)	비 선점 방식으로 수행시간이 가장 짧은 것을 먼저 수행하게 함. 빠른 응답을 요하는 대화식 방식 또는 시분할 방식에는 적합하지 않음
Priority	우선순위가 각 프로세스에게 주어지며, 중앙처리장치는 가장 높은 우선순위를 가진 프로세스로 할당. 무한 대기와 기아 현상 문제 존재, 해결책으로 에이징 존재(우선순위 점진적 증가)
Round Robin	시분할 시스템을 위하여 고안된 선점 스케줄링 방식으로 각 프로세스는 같은 크기의 중앙처리장치 시간을 할당 받음. 할당시간내에 마치지 못하면 준비 리스트의 가장 뒤에 저장됨 대화식 사용자들에게 알맞은 응답시간으로 보장해 주어야 하는 시분할 방식의 시스템에 효과적인 방식
SRT(Shortest Remaining Time)	SJF 기법에 선점 방식을 도입한 방법으로 시분할 시스템에서 유용. 새로 도착한 프로세스를 포함하여 처리가 완료되는데 가장 짧은 시간이 소요되는 프로세스를 먼저 수행
Multi Level Queue	작업들을 여러 그룹으로 나누어 여러 개의 큐를 이용하는 방법.
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
HRN(High Response ratio Next)	긴 작업과 짧은 작업 간의 지나친 불평등을 어느정도 보완한 기법. 한 작업이 중앙처리장치를 차지하면 그 작업은 완성될 때까지 실행하며, 대기시간이 고려되어 긴 작업과 짧은 작업 간의 불평등을 어느 정도 완화시킴

3. 프로세스 간 동기화 및 통신

1) 병행 프로세스(Concurrent Process)

- 두 개 이상의 프로세스가 동시에 수행 중인 상태
- 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

2) 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

3) 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

4) 상호배제

- 한 프로세스가 임계영역에서 실행 중 일 때, 다른 어떤 프로세스도 임계영역에서 실행될 수 없도록 하는 것,

5) 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

  파일, 신호, 소켓, 메시지 큐, 파이프, 지명 파이프, 세마포어, 공유 메모리, 메시지, 메모리맵 파일

6) 세마포어

- 운영체제(또는 프로그래밍) 내에서 공유 자원에 대한 접속을 제한하기 위해서 사용되는 신호
- 프로세스 동기화 도구
- 일반적으로 세마포어로 사용되는 변수 S는 정수형 변수로써, 초기화를 제외하고는 단지 두 개의 표준

  연산으로만 접근할 수 있음
- S : 정수 값을 가지는 변수, P와 V라는 명령에 의해서만 접근할 수 있음

  P : 임계 구역에 들어가기 전에 수행됨

  V : 임계 구역에서 나올 때 수행됨

  한 프로세스(스레드)에서 세마포어 값을 변경하는 동안 다른 프로세스가 동시에 이 값을 변경해서는 안됨

- 종류
  - 계수 세마포어 : 초기값은 가능한 자원의 수로 정해지며, 세마포어 값의 범위는 정해져 있지 않음
  - 이진 세마포어 : 세마포어 값으로 0 또는 1을 가짐. 계수 세마포어보다 간단히 구현할 수 있으며, 하드웨어가 지원하는 기능을 이용하여 구현하기도 함

- 약점

  - 고급 언어에서 동기화를 제공해야 함
  - P함수와 V함수의 동작이 독립적이기 때문에 잘못 사용하는 경우 문제가 발생함
  - P - 임계 구역 - P : 현재 프로세스가 임계 구역에서 빠져나갈 수 없고 다른 프로세스가 임계 구역에 들어갈 수 없으므로 교착상태(Deadlock) 발생
  - V - 임계 구역 - P : 2개 이상의 프로세스가 동시에 임계 구역에 들어갈 수 있으므로 상호 배제를 보장할 수 없음

7) 모니터

- 프로세스 또는 스레드를 동기화 하는 방법 중 하나로, 그 방법으로 활용하기 위해 구현된 기능 또는 모듈
- 주로 고급 언어에서 이 기능이 지원되며, 한번에 하나의 프로세스만 모니터에서 활동하도록 보장해줌
- 세마포어는 동기화 함수의 제약 조건을 고려해야 하는 반면, 모니터는 프로시저를 호출하여 간단히 해결
- 동작

  어떤 공유 데이터에 대해 모니터를 지정해 놓으면, 프로세스는 그 데이터를 접근하기 위해 모니터에 들어가야함. 즉, 모니터 내부에 들어간 프로세스에게만 공유 데이터를 접근할 수 있는 기능을 제공하는 것이고, 프로세스가 모니터에 들어가고자 할 때 다른 프로세스가 모니터 내부에 있다면 입장 큐에서 기다려야 함

8) 메시지

- 프로세스 간의 통신 및 동기화에 적합한 비교적 단순한 매커니즘
- 송신 측 프로세스와 수신 측 프로세스 간에 교환될 수 있는 정보(데이터, 실행 명령)의 집합
- 많은 다중 프로그래밍 운영체제들이 몇 가지 종류의 프로세스 간 통신을 지원하기 위하여 메시지 메커니즘을 채택하고 있고, 분산 운영 체제에서는 메시지 메커니즘이 보편화 되어있음
- 형식 : 송신 측 및 수신 측 식별자, 형식, 길이, 데이터 등으로 구성
- 고려해야 할 사항 : 네이밍, 복사, 버퍼링, 길이 문제

4. 교착 상태

1) 교착상태(Deadlock)

- 하나 또는 그 이상의 프로세스가 발생될 수 없는 어떤 특정 사건을 기다리고 있는 상태
- 특정 프로세스가 특정한 자원을 위하여 무한정 기다려도 도저히 해결할 수 없는 상태
- 컴퓨터 시스템의 효율을 급격히 떨어뜨리는 문제점을 발생
- 두 개 이상의 작업이 서로 상대방의 작업이 끝나기 만을 기다리고 있기 때문에 결과적으로 아무것도 완료되지 못하는 상태

2) 교착 상태의 조건

- 상호배제(Mutual Exclusion) : 프로세스들이 필요로 하는 자원에 대해 배타적인 통제권을 요구함
- 점유대기(Hold and Wait) : 프로세스가 할당된 자원을 가진 상태에서 다른 자원을 기다림
- 비 선점(No preemption) : 프로세스가 어떤 자원의 사용을 끝날 때까지 그 자원을 뺏을 수 없음
- 순환대기(Circular Wait) : 각 프로세스는 순환적으로 다음 프로세스가 요구하는 자원을 가지고 있음
- 이 조건 중에서 한 가지라도 만족하지 않으면 교착 상태는 발생하지 않고, 이 중 순환대기 조건은 점유대기 조건과 비 선점 조건을 만족해야 성립하는 조건이므로 위의 조건들은 서로 완전히 독립적이지는 않음

3) 교착 상태 관리

- 예방

  - 상호배제 조건의 제거 : 교착 상태는 두 개 이상의 프로세스가 공유가능한 자원을 사용할 때 발생하는 것이므로 공유 불가능한, 즉 상호 배제 조건을 제거하면 교착 상태를 해결할 수 있음
  - 점유와 대기 조건의 제거 : 한 프로세스에 수행되기 전에 모든 자원을 할당 시키고 나서 점유하지 않을 때에는 다른 프로세스가 자원을 요구하도록 하는 방법. 자원 과다 사용으로 인한 효율성, 프로세스가 요구하는 자원을 파악하는 것에 대한 비용, 자원에 대한 내용을 저장 및 복원하기 위한 비용, 기아 상태, 무한 대기 등의 문제점이 존재
  - 비 선점 조건의 제거 : 비 선점 프로세스에 대해 선점 가능한 프로토콜 제작
  - 환형 대기 조건의 제거 : 자원 유형에 따라 순서를 매김
  - 이 해결 방법들은 자원 사용의 효율성이 떨어지고 비용이 많이 드는 문제점이 있음

- 회피

  - 자원이 어떻게 요청될지에 대한 추가정보를 제공하도록 요구하는 것으로, 시스템에 환형 대기가 발생하지 않도록 자원 할당 상태를 검사함
  - 자원 할당 그래프 알고리즘 : 그래프를 그려서 교착 상태 여부를 판단하는 것
  - 은행원 알고리즘 : 안전 상태 알고리즘으로, 여러 개의 프로세스가 있을 때 그 프로세스가 어떤 특정 순서대로 자원을 요청할 때 아무런 문제없이 자원을 할당 받을 수 있는 경우

    단점 : 사전에 필요한 자원의 최대량을 제시해야 하나 파악이 어려움

- 교착 상태의 무시

  예방 혹은 회피기법을 프로그래밍해서 넣으면 성능에 큰 문제를 미치기 때문에 교착 상태의 발생 확률이 비교적 낮은 경우 별다른 조치를 취하지 않음

- 교착 상태의 발견

  감시 및 발견을 하는 탐지 알고리즘으로 교착 상태 발생을 체크하는 방식, 성능에 큰 영향을 미칠 수 있음