**Ch7) Deadlocks**

* 한 쌍의 프로세스가 각각 공유 자원을 점유하고 내주지 않을 때 발생하는 문제(교착 상태)이다.

1. **Deadlock Characterization**

* Mutual exclusion
* hold and wait
* no preemption
* circular wait

1. **Handling Deadlock**
2. Deadlock prevention – 데드락이 발생할 수 없게 끔 소프트웨어 설계(개발자 역할)
3. Deadlock avoidance – 데드락에 빠질 가능성이 있는지 없는지 운영체제가 검사 후 가능성이 없을 경우만 공유 자원을 넘겨줌(운영체제 역할)
4. Deadlock detection and recovery – 교착 상태의 프로세스를 한 개씩 중지 후 데드락이 빠지기 전 상태로 복구
5. Ignore – 무시
6. **Safe state and unsafe state (avoidance)**

* Safe state -> no deadlock
* Unsafe stae -> possibility of deadlock
* Avoidance – unsafe state에 들어가지 않도록 보장하는 것이 핵심

1. **Banker’s Algorithm (대표적인 avoidance 알고리즘)**

* Process의 최대 자원 요구량을 파악하기 힘들다
* Process 수가 동적으로 변화함
* 결국 deadlock 문제는 원천적으로 해결이 불가능하다
* 그럼에도 banker’s alg를 배우는 이유는 알고리즘 자체에 의의가 있기 때문이다.
* 사실상 운영체제에서 deadlock 문제에 대해서는 손을 놓고 있는 실정이다.

**Ch8) Memory Management Strategies**

* 제한적 물리적 메모리의 효율적인 관리가 필요

1. **데이터가 언제 메모리에 할당되는가?**
2. Compile time - compile시에 메모리 주소 결정
3. Load time – load 할 때 메모리 주소 결정
4. Execution time – 실행할 때 메모리 주소 결정, 주소값을 계산해주는 연산을 해야해서 오버헤드가 커질 수 있지만 연산이 하드웨어 로직에 포함된다면 그렇지 않다. -> MMU(memory manage unit)
5. **Virtual memory(VM) – 가상메모리**

* Logical address에서 physical address로 translation하는 부분이 virtual memory management의 핵심

1. **Dynamic Linking – DLL(Dynamic Link Library)**
2. **Contiguous Allocation**

* External Fragmentation(외부 단편화) 문제 발생-모리에 충분한 공간이 있음에도 불구하고 partition 때문에 적재 불가능할 경우가 생김 -> Page개념 등장
* Internal Fragmentation(내부 단편화) – 내부에서 빈 공간 발생

1. **Paging**

* process를 더 작은 단위로 쪼개서 적재
* Logical memory(page) -> page table(mapping) -> physical memory(frame)
* 외부단편화 극복, 내부단편화 존재 -> segmentation

1. **Page Table**

* 주소변환을 위한 자료구조로 물리적 메모리에 위치
* 프로세스 마다 하나씩 존재
* TLB(hardware cache) – 자주 참조되는 페이지에 대한 주소 변환 정보 저장
* Page table은 생각보다 크기가 크기 때문에 cpu구현을 통해 사용하지는 않고 page table의 일부만을 MMU에 넣어서 CPU의 cache메모리 형태로 사용, TLB라는 cache 메모리를 따로 두고 사용하여 오버헤드를 줄인다.
* TLB hit -> TLB cache에 존재, TLB miss -> TLB cache에 존재X

**Ch9) Virtual Memory Management**

1. **Demand Paging**

* 하드디스크에 있는 것 중 필요한 것만 physical memory에 올린다.
* Temporal locality, Spatial locality에 기반해 작동

1. Valid, invalid Bit
2. **Page Fault**

* 요청한 page가 메모리에 있어야 참조가 되는데, 메모리에는 없고 하드디스크에 있는 상황

1. **Free frame이 없는 상태라면?**

* 앞으로 안 쓰일 것(victim)을 하드디스크에 내린다 -> page replacement

1. **Page Replacement**

* Page replacement algorithms
* Want lowest page-fault rate

1. FIFO Alg
2. Optimal Alg – 미래 데이터 확인
3. LRU Alg – 과거 데이터 확인

**Ch10) File System(사용자 입장)**

**Ch11) Implementing File System(개발자 입장)**

* On-disk structure
* In-memory structure

1. **On-disk structure**

* 어떻게 파일이 하드디스크에 저장되어 있는가?

1. Master Boot Record(MBR)
   * Boot loader: 운영체제를 메모리에 올려놓기
2. Partition
3. Boot block – 운영체제 정보 저장
4. Super block – file system metadata – 파일시스템의 전반적인 정보 저장
5. Bitmap block – freespace management – 여유공간 관리
6. I-nodes – file metadata – 파일이 생성될 때 마다 하나씩 생성
7. Root directory block – 가장 상위 폴더
8. Files and directory block – data block
9. **Allocation Method**

* Disk block을 어떻게 할당할 것인가?(file 크기에 따라)

1. Contiguous allocation
   * 연속적 할당
   * 단점: 외부 단편화
2. Linked allocation
   * Linked list로 할당
   * 단점: 성능 하락, 신뢰도 문제(bad sector)
3. Indexed allocation
   * Linked list의 모든 포인터를 인덱스 테이블에 저장(유닉스에서 i-node라고 부르는 이유)
4. **Block Size Performance vs Efficiency**

* Block size가 커지면 더 많은 정보를 한꺼번에 가져올 수 있다.
* 하지만 일정 수준이 넘어서면 내부단편화가 일어난다.
* Trade-off, 현재 적절한 타협점의 block size는 4KB

**Ch12) Secondary – Storage Architecture**

1. **Disks**

* Platters, surfaces, tracks, sectors, cylinders, arm, heads
* 운영체제에서 data를 관리하는 단위가 디스크에 읽고 쓰는 단위인 sector(512B)가 아니라 4KB인 disk block 단위이다.
* 같은 cylinder방향으로 데이터를 저장하는 것이 효율적
* Disk performance

1. Seek: disk arm이 cylinder를 찾아가는 것
2. Rotation: platter가 회전하여 sector를 찾아가는 것
3. Transfer
4. **Disk Scheduling**

* Seek time <-> seek distance <-> arm의 이동거리
* Minimize seek time
* Seek time이 가장 길기 때문에 이 seek time을 줄여주는 것이 disk scheduling의 핵심이다.

1. FCFS – 순서대로
2. SSTF(Shortest seek time first)
   * SJF scheduling과 유사
   * 가장 효율적인 방식, window와 linux에서 사용
3. SCAN
   * Elevator algorithm
   * 일반적으로는 SSTF를 사용하 지만 특수한 환경에서는 SCAN이나 C-SCAN 방식 사용
   * Q에 있는 양끝점이 아닌 disk에 있는 양끝 점 방문
4. C-SCAN
   * SCAN과 유사하지만 한쪽 방향으로만!
5. C-LOOK
   * C-SCAN의 개량 버전
   * 양끝 방문X, track번호의 최대치 최소치 확인 필요

* 대용량 데이터를 로드해야 할때는 SCAN과 C-SCAN방식 사용

1. **RAID**

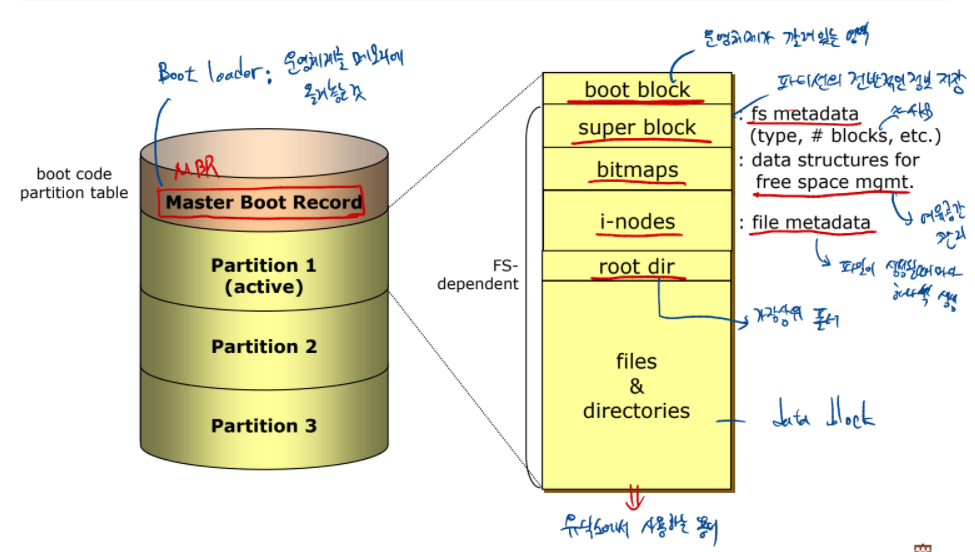
* Redundant(중복된) Array of Inexpensive Disks
* 하나의 비싸고 큰 용량의 디스크보다 여러 개의 싸고 작은 용량의 디스크
* Reliability(안정성) 향상 – mirroring(shadowing), parity, error-correcting codes
* Performance(성능) 향상 – 병렬방식으로 – bit striping, block striping
* Striping – 연속된 데이터를 여러 개의 디스크에 RR 방식(병렬)으로 저장하는 기술
* RAID 번호를 맞추는 것 보다는 어떠한 방식으로 RAID를 구성했는지 맞추는 방식으로 문제가 나올 예정이다.

기말

1. 다음 용어를 설명하시오
2. **Deadlock** – 두 개 이상의 process가 서로 상대방의 작업이 끝나기 만을 기다리면서 공유 자원을 얻지 못해서 다음 처리를 하지 못하는 상태
3. **Polling and Interrupt**
   * Polling: cpu가 device에게 지속적으로 이벤트 수행 여부에 대해 물어보는 방식
   * Interrupt: Device가 CPU에게 이벤트 발생 여부를 알려주는 방식
4. **DMA(Direct Memory Access)**
   * CPU의 간섭없이 memory와 I/O 장치 간의 데이터 전송하는 방식
5. **Virtual Memory**
   * Demand paging기법을 사용하여 실제 메모리 크기보다 더 큰 메모리를 활용할 수 있게 해주는 것.
6. **Consider the following snapshot of a system. (Banker’s alg)**



1. **Draw on-disk structure of Unix “File System” and describe each part**



1. **Explain the difference between contiguous alloc and linked alloc on disk space.**

* Contiguous allocation: 각 파일을 연속적으로 block에 할당한다. 외부 단편화가 발생할 수 있다는 단점이 있다.
* Linked allocation: 각 파일을 linked list로 할당. 외부 단편화를 해결할 수 있지만, 성능이 떨어지고 신뢰도 문제가 있다.

1. **Describe what is the RAID(Redundant Array of Independent Disks) and give a detailed description of the following RAID level 4**

* Block-level striping의 performance 향상 방식과 Parity의 Reliability 방식을 사용한 방법이다.

1. **Describe what is difference between Direct I/O and Memory Mapped I/O.**

* Direct I/O: main memory address와 다르게 독립적 I/O port address를 둔다.
* Memory mapped I/O: main memory와 I/O port의 address를 통합해서 관리한다.

1. **Draw the I/O System Layers and describe detailed operation.**

