파일 할당

File Allocation

양희재 교수 (hjyang@ks.ac.kr) / 경성대학교 컴퓨터공학과

파일 할당

- 컴퓨터 시스템 자원 관리
  - CPU: 프로세스 관리 (CPU 스케쥴링, 프로세스 동기화)
  - 주기억장치: 메인 메모리 관리 (페이징, 가상 메모리)
  - 보조기억장치: 파일 시스템

- 보조기억장치 (하드 디스크)
  - 하드디스크: track (cylinder), sector
  - Sector size = 512 bytes, *cf.* Block size
  - 블록 단위의 읽기/쓰기 (block device)
  - 디스크 = *pool of free blocks*
  - 각각의 파일에 대해 free block 을 어떻게 할당해줄까?

파일 할당

- 파일 할당
  - 연속 할당 (Contiguous Allocation)
  - 연결 할당 (Linked Allocation)
  - 색인 할당 (Indexed Allocation)

연속 할당

- Contiguous Allocation
  - 각 파일에 대해 디스크 상의 연속된 블록을 할당
  - 장점: 디스크 헤더의 이동 최소화 = 빠른 i/o 성능
  - 옛날 IBM VM/CMS 에서 사용
  - 동영상, 음악, VOD 등에 적합
  - 순서적으로 읽을 수도 있고 (sequential access 순차접근)
  - 특정 부분을 바로 읽을 수도 있다 (direct access 직접접근)
  - 단점?

연속 할당

- Contiguous Allocation
  - 파일이 삭제되면 hole 생성
  - 파일 생성/삭제 반복되면 곳곳에 흩어지는 holes
  - 새로운 파일을 어느 hole 에 둘 것인가? ☞ 외부 단편화!
    - First-, Best-, Worst-fit
  - 단점: 외부 단편화로 인한 디스크 공간 낭비
    - Compaction 할 수 있지만 시간 오래 걸린다 (초창기 MS-DOS)
  - 또 다른 단점?

연속 할당

- Contiguous Allocation
  - 파일 생성 당시 이 파일의 크기를 알 수 없다 ☞ 파일을 어느 hole 에?
  - 파일의 크기가 계속 증가할 수 있다 (log file) ☞ 기존의 hole 배치로는 불가!
  - 어떻게 해결할 수 있을까?

연결 할당

- Linked Allocation
  - 파일 = *linked list* of data blocks
  - 파일 디렉토리(directory)는 제일 처음 블록 가리킨다.
  - 각 블록은 포인터 저장을 위한 4바이트 또는 이상 소모

- 새로운 파일 만들기
  - 비어있는 임의의 블록을 첫 블록으로
  - 파일이 커지면 다른 블록을 할당 받고 연결
  - 외부 단편화 없음!

연결 할당

- 단점
  - 순서대로 읽기 = sequential access
    - Direct access 불가
  - 포인터 저장 위해 4바이트 이상 손실
  - 낮은 신뢰성: 포인터 끊어지면 이하 접근 불가
  - 느린 속도: 헤더의 움직임

연결 할당

- 개선: FAT 파일 시스템
  - *File Allocation Table* 파일 시스템
  - MS-DOS, OS/2, Windows 등에서 사용
  - 포인터들만 모은 테이블 (FAT) 을 별도 블록에 저장
  - FAT 손실 시 복구 위해 이중 저장
  - Direct access 도 가능!
  - FAT 는 일반적으로 메모리 캐싱

색인 할당

- Indexed Allocation
  - 파일 당 (한 개의) 인덱스 블록 – *데이터 블록 외에*
  - 인덱스 블록은 포인터의 모음
  - 디렉토리는 인덱스 블록을 가리킨다.
  - Unix/Linux 등에서 사용

- 장점
  - Direct access 가능
  - 외부 단편화 없음

색인 할당

- 단점
  - 인덱스 블록 할당에 따른 저장공간 손실
    - 예: 1바이트 파일 위해 데이터 1블록 + 인덱스 1블록

- 파일의 최대 크기
  - 예제: 1블록 = 512바이트 = 4바이트 x 128개 인덱스
    - 128 * 512바이트 = 64KB
  - 예제: 1블록 = 1KB = 4바이트 x 256개 인덱스
    - 256 * 1KB = 256KB
  - 해결 방법: Linked, Multilevel index, Combined 등

디스크 스케쥴링

Disk Scheduling

디스크 스케쥴링

- 디스크 접근 시간
  - Seek time + rotational delay + transfer time
  - 탐색시간 (seek time) 이 가장 크다.
- 다중 프로그래밍 환경
  - *디스크 큐(disk queue)*에는 많은 요청(request)들이 쌓여있다.
  - 요청들을 어떻게 처리하면 탐색시간을 줄일 수 있을까?
- 디스크 스케쥴링 알고리즘
  - FCFS (First-Come First-Served)
  - ???

FCFS Scheduling

- First-Come First-Served
  - Simple and fair

- 예제
  - 200 cylinder disk, 0 .. 199
  - Disk queue: 98 183 37 122 14 124 65 67
  - Head is currently at cylinder 53
  - Total head movement = **640 cylinders**
  - *Is FCFS efficient?*

SSTF Scheduling

- Shortest-Seek-Time-First
  - Select the request with the minimum seek time from the current head position
- 예제
  - 200 cylinder disk, 0 .. 199
  - Disk queue: 98 183 37 122 14 124 65 67
  - Head is currently at cylinder 53
  - Total head movement = **236 cylinders**
- 문제점
  - Starvation
  - *Is SSTF optimal? No!* (e.g., 53 → 37 → … = 208 cyl)

SCAN Scheduling

- Scan disk
  - The head continuously scans back and forth across the disk
- 예제
  - 200 cylinder disk, 0 .. 199
  - Disk queue: 98 183 37 122 14 124 65 67
  - Head is currently at cylinder 53 *(moving toward 0)*
  - Total head movement = **53+183 cylinders** *(less time)*
- 토론
  - Assume a uniform distribution of requests for cylinders
  - *Circular SCAN is necessary!*

SCAN Variants

- C-SCAN
  - Treats the cylinders as a cicular list that wraps around from the final cylinder to the first one
- LOOK
  - The head goes only as far as the final request in each direction
  - *Look* for a request before continuing to move in a given direction
- C-LOOK
  - LOOK version of C-SCAN

Elevator Algorithm

- SCAN and variants
  - SCAN & C-SCAN
  - LOOK & C-LOOK
  - The head behaves just like an elevator in a building, first servicing all the requests going up, and then reversing to service requests the other way