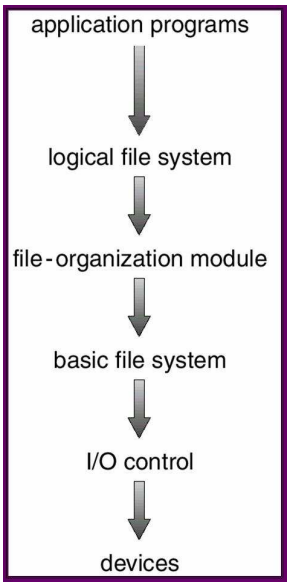
Slide 0

3부 저장장치 관리
(12장. 파일시스템 구현)

파일시스템 구조

Slide 1

- 디스크의 특징
  - 재기록이 가능
  - 임의의 블록에 직접접근 가능
- 디스크 입출력
  - 입출력은 효율성을 위해 블록 단위로 실행
  - 블록은 하나 이상의 섹터로 구성
  - 섹터는 보통 32B ~4KB 까지 구성되며 보통 512B
- 파일 시스템 구조
  - 계층적으로 구성됨



파일시스템 구조 (계속)

Slide 2

- 입출력 제어(I/O control)
  - 장치 구동기(device driver) + 인터럽트 처리기
  - 주기억장치와 디스크 사이의 정보전송
  - 장치 구동기
    - 상위 수준의 명령을 하드웨어 제어기 수준의 명령으로 구성
    - 하드웨어 제어기를 직접 제어하여 명령을 내리고 데이터를 이동
    - 데이터 이동시 인터럽트 방식으로 동작 (인터럽트 처리기에서 처리)
- 기본 파일 시스템(basic file system)
  - 디스크에 있는 정보를 읽거나 기록하기 위해 장치구동기를 동작
  - 블록 위치는 보통 장치번호, 실린더번호, 트랙번호, 섹터번호로 구성됨
- 파일 구성 모듈(file-organization module)
  - 파일의 논리적 블록과 물리적 블록에대한 정보를 가짐
  - 논리적 블록주소를 물리적 블록주소로 변환 기본 파일 시스템에 제공
  - 보통 논리적 블록 주소는 0(혹은 1)에서 N번지 존재
  - 보통 물리적 블록 주소는 논리적 주소와 다름으로 변환 필요
  - 가용 공간을 관리하며 요청시 할당함

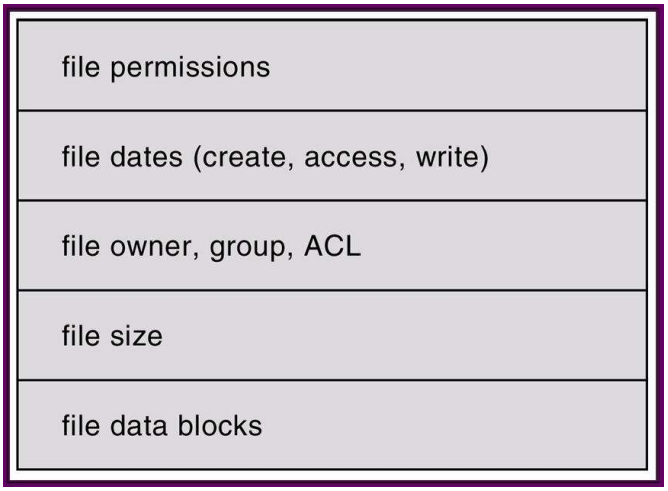
파일시스템 구조 (계속)

Slide 3

- 논리적 파일 시스템(logical file system)
  - 메타데이터를 관리함
  - 메타데이터란 파일 데이터를 제외한 파일 시스템에 대한 모든 정보
  - 파일 구성 모듈을 위하여 디렉토리 구조를 관리함
  - FCB(file control block)을 이용하여 파일을 관리
  - FCB
    - 파일에 대한 정보
    - 소유자
    - 허용(permissions)
    - 파일 내용의 위치
    - 보호
    - 보안
- 대부분의 운영체제는 하나 이상의 파일 시스템을 지원
  - 윈도우: FAT, FAT32, NTFS등
  - Unix: UFS(Unix File System)
  - Linux: ext2, ext3

파일시스템 구조 (계속)

Slide 4



파일 시스템 구현

Slide 5

- 개요
  - 파일 시스템 구현을 위해 운영체제는 디스크에 많은 정보를 관리
  - 디스크에 유지되는 정보
    - 부트 제어 블록(boot control block)
      - 운영체제를 부팅하기 위해 필요한 정보를 여기에 유지
      - 일반적으로 파티션의 첫번째 블록에 저장
      - 모든 파티션마다 있을 필요는 없음
    - 파티션 제어 블록(partition control block)
      - 파티션에대한 정보를 유지하는 블록으로 다음과 같은 정보를 유지
      - 파티션의 크기(블록수), 블록의 크기, 빈 블록수, 빈 블록 포인터, 빈 FCB수, FCB 포인터
    - 디렉토리 구조
    - 유닉스에서는 FCB를 inode로 부름

파일 시스템 구현 (계속)

Slide 6

- 주기억장치에 유지되는 정보
  - 파티션 테이블: 마운트된 각 파티션에 대한 정보를 유지하는 테이블
  - 최근에 접근된 디렉토리 구조: 디렉토리에대한 소프트웨어 캐시
  - 시스템 전체 오픈 파일 테이블(SWOFT: system-wide open-file table)
    →오픈한 모든 파일의 FCB 복사본을 유지하는 테이블
  - 프로세스별 오픈 파일 테이블(PPOFT: per-process open-file table)
    →SWOFT의 항을 가리키는 포인터를 유지
- 파일 생성
  - 새 FCB를 할당하고 해당 디렉토리를 주기억장치로 읽어 갱신후 디스크 저장
  - 유닉스는 디렉토리를 파일과 동일하게 처리
    (FCB내에 파일인지 디렉토리인지 나타내는 field가 있음)

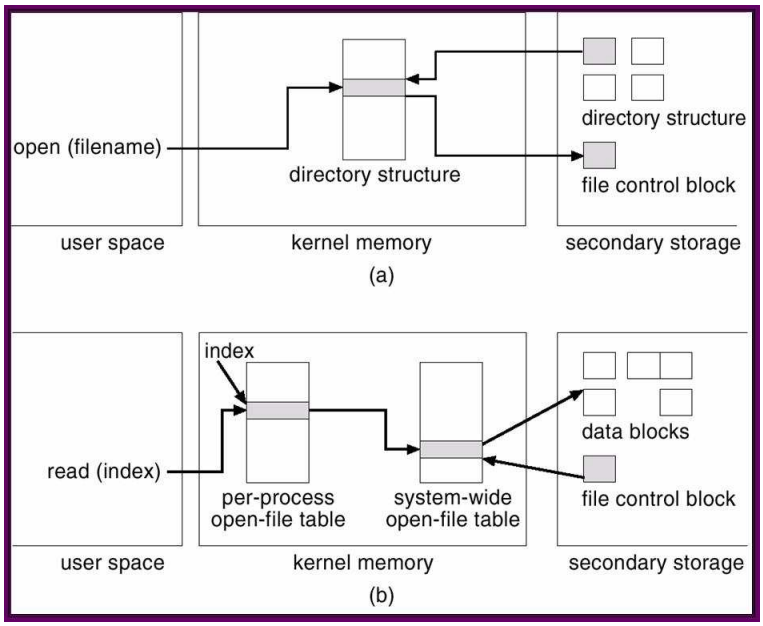
파일 시스템 구현 (계속)

Slide 7

- 파일 열기
  - 열기시 파일이름을 파일시스템에 넘겨 줌
  - 디렉토리를 검색해 해당 파일을 찾음
    (디렉토리 구조의 일부가 주기억장치에 캐시됨)
  - 해당 파일의 FCB가 SWOFT에 복사됨
    (SWOFT에는 해당 파일을 열기한 프로세스의 수도 저장됨)
  - PPOFT에 새 항이 추가됨
  - 이 항에는 SWOFT의 FCB에 대한 포인터등이 있음
    (또한 현재 파일 위치에 대한 포인터도 포함됨)
  - 열기는 PPOFT에 추가된 항에 대한 포인터를 반환
  - 모든 파일관련 동작은 이 포인터를 사용해 이루어짐
    (유닉스에서는 file descriptor라 하고 윈도우에서는 file handle이라함)
- 파일 닫기
  - PPOFT의 해당 항을 삭제
  - SWOFT의 열기 계수를 하나 감소
    (열기 계수가 0이면 파일 정보를 디스크에 쓰고 SWOFT 항을 제거)

파일 시스템 구현 (계속)

Slide 8



파일 시스템 구현 (계속)

Slide 9

- 파티션과 마운팅
  - 파티션
    - 하나의 디스크가 여러개의 파티션으로 분할 가능 (윈도우에서 C: D:)
    - 하나의 파티션에 여러개의 디스크가 사용됨 (RAID 디스크)
  - 파티션이
    - 파일 시스템을 포함한 경우 (cooked disk)
    - 포함하지 않은 경우 (raw disk)
  - raw disk를 사용하는 경우
    - 유닉스의 스왑공간
    - 데이터베이스에서 자체적인 포맷을 사용할 때
  - 부트로더 (부트 파티션에 있는 부팅 프로그램)
    - 부팅시에는 장치 구동기가 실행되기 전이므로 자체적인 포맷을 사용
    - 여러개의 파일시스템 및 운영체제를 인식하면 다중 부팅이 가능함
    - 각 파티션별로 다른 파일시스템과 다른 운영체제를 설치 다중부팅 가능
    - 운영체제 커널을 포함하는 루트파티션은 자동으로 마운트되며 다른 파티션은 자동 혹은 수동으로 마운트됨

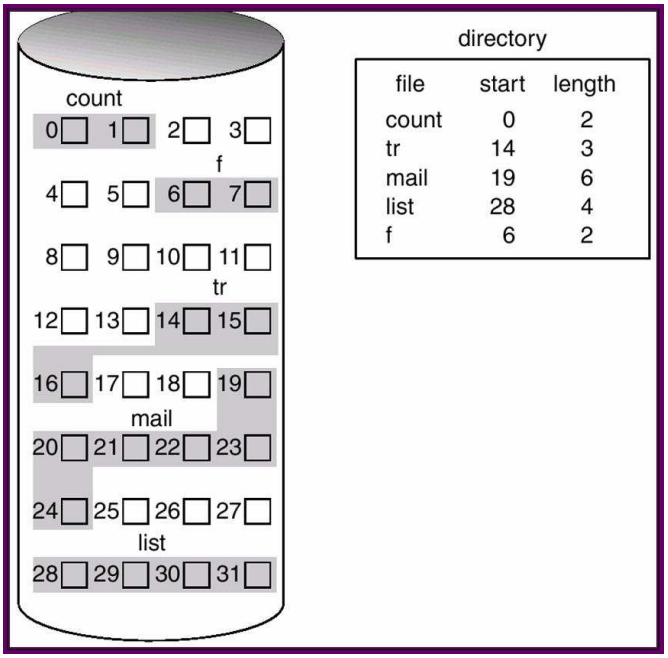
할당 방법

Slide 10

- 연속 할당
  - 방법
    - 각 파일을 연속된 공간에 할당 (주기억장치에서와 유사)
    - 빈 공간을 찾기위해 홀 정보를 유지해야함
    - 최초적합, 최적적합, 최악적합 사용 가능
  - 장점
    - 파일 접근 시간이 빠름
    - 파일에 대한 순차접근과 직접접근이 가능
    - 디렉토리에는 시작위치와 크기만 유지하면 됨
  - 단점
    - 외부 단편화 발생
    - 파일 생성시 파일 크기를 알 수 없음
    - 파일 복사시에는 문제 안됨

할당 방법 (계속)

Slide 11



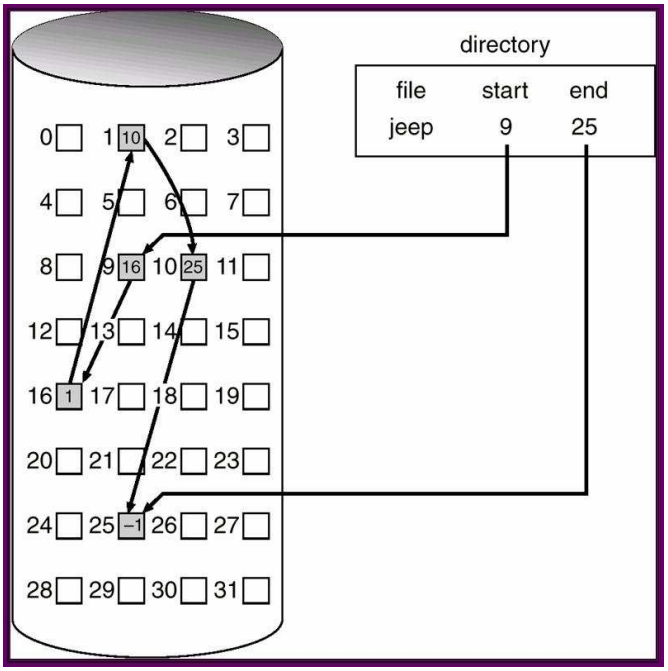
할당 방법 (계속)

Slide 12

- 연결 할당
  - 방법
    - 디스크 블록의 연결형태로 할당
    - 디렉토리에는 파일의 첫 블록과 마지막 블록에대한 포인터만 유지
    - 각 블록에는 다음 블록을 가리키는 포인터가 있음
  - 장점
    - 외부 단편화가 발생하지 않음
    - 파일이 동적으로 커져도 문제 발생하지 않음
  - 단점
    - 파일에 대한 순차접근만 가능 (연결 리스트를 따라서..)
    - 블록에 포인터를 저장하여야 함 (예, 512바이트 한 블록에서 4바이트)
      (영향을 줄이기 위하여 블록을 몇개 묶어서 클러스터 단위로 할당)
      (클러스터 방법은 내부 단편화 발생 시킴)
    - 신뢰도가 떨어짐 (포인터 손상으로 파일 자체가 손상됨)
      (이중 연결 리스트로 강화 가능하나 저장 오버헤드가 커짐)

할당 방법 (계속)

Slide 13



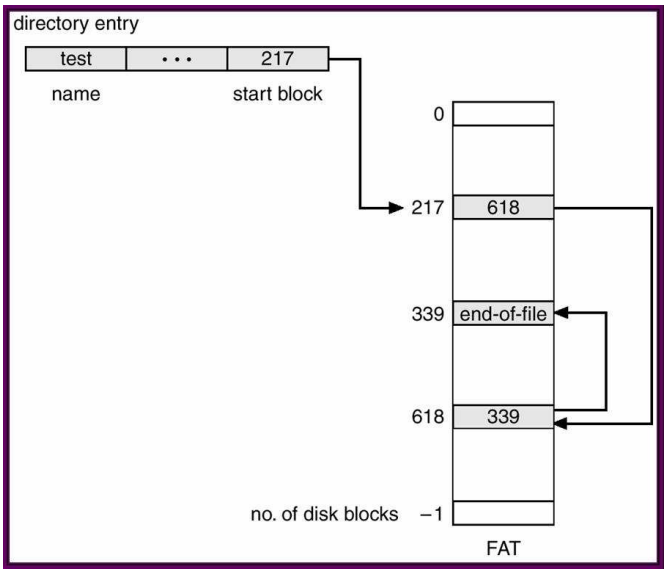
할당 방법 (계속)

Slide 14

- FAT (File Allocation Table)
  - 연결 할당의 변형으로 MS-DOS 와 OS/2에서 사용
  - 파티션의 시작 섹션에 테이블을 유지
  - 테이블은 각 블럭당 한 항목이 있음
  - 각 항목은 블록 번호로 색인됨
  - 디렉토리 항목은 파일의 첫번째 블록번호를 가지고 있음
  - 블록번호로 색인된 테이블에 그 파일의 다음 블럭에대한 번호가 있음
  - 파일의 마지막 블록에는 파일의 끝을 나타내는 값이 있음
  - 사용되지 않는 블록에 대한 테이블 값은 0
  - 새로운 파일 생성시 테이블에서 0인 항목을 찾아서 할당

할당 방법 (계속)

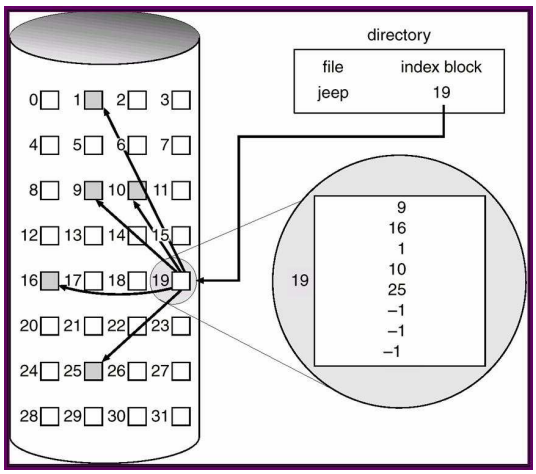
Slide 15



할당 방법 (계속)

Slide 16

- 색인 할당
  - 디스크의 한 블록에 파일의 나머지 블록에 대한 포인터를 유지
  - 장점은 연결 할당의 문제점인 직접접근 불가능 문제를 해결
  - 단점은 연결 할당에 비해 공간 낭비가 심함



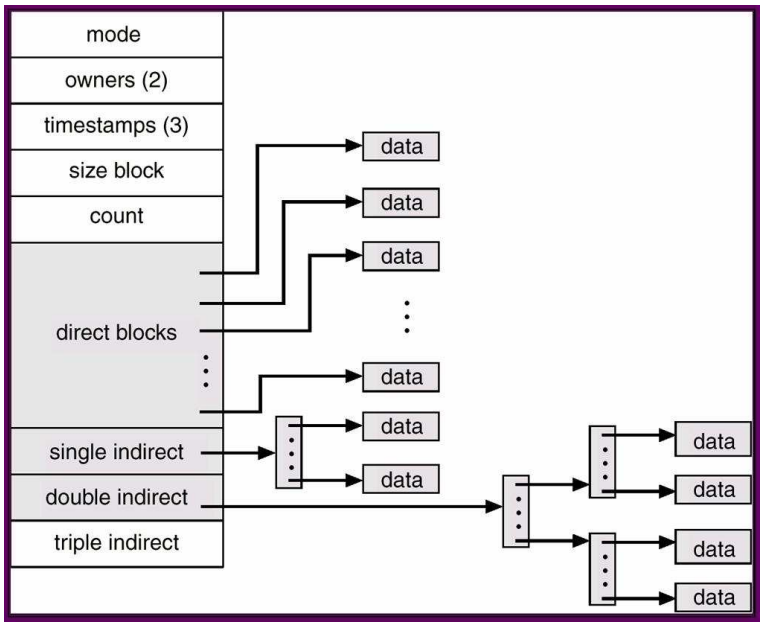
할당 방법 (계속)

Slide 17

- 색인 블록 할당 방법
  - 연결 기법(linked scheme)
    - 처음에는 하나의 색인 블록 할당
    - 필요시 여러개의 색인 블록을 연결 리스트로 사용
  - 다단계 색인(multilevel index)
    - 색인블록을 다른 색인 블록에 연결
    - 4096바이트 블록에서 4B 포인터는 1024개의 색인 가능
    - 2단계 색인에서는 (1K × 1K × 4KB = 4GB) 최대 4G 파일생성
  - 결합 기법 (combined scheme)
    - 유닉스에서 사용하는 방법
    - 파일의 inode에 15개의 포인터를 유지
    - 15개중 12개는 직접 데이터 블록을 가리킴
      (블록이 4K라면 12 × 4K = 48K, 즉 48K 이하의 파일은 직접 접근)
    - 13번째는 단일 간접, 14번째는 이중 간접, 15번째는 삼중 간접으로

할당 방법 (계속)

Slide 18



할당 방법 (계속)

Slide 19

- 성능
  - 디스크 공간할당 방법의 성능은
    - 저장 공간 효율
    - 접근속도측면에서 고려해야함
  - 접근속도 측면
    - 연속할당이 가장 우수
    - 파일의 크기 변화에 대처하기 어렵고 가용공간 할당이 어려움
  - 파일 j번째 블록을 읽기위해 읽는 블록의 수
    - 연속 할당: 1
    - 연결 할당: $j-1$
    - 색인 할당: $i+1$, i는 j블록을 읽기위해 읽는 색인 블록의 수
  - 할당 방법별 장점을 활용하기 위해 여러 방법을 혼용하는 경우도 있음

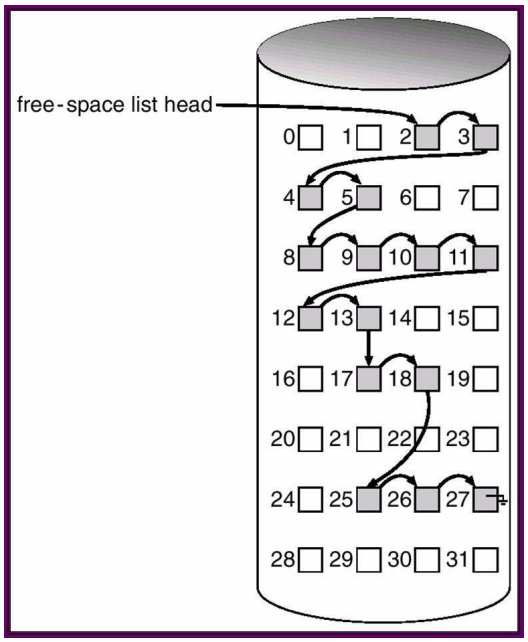
가용 공간 관리

Slide 20

- 비트벡터
  - 각 블록을 1비트로하여 할당여부 표현(1이면 빈곳)
  - 장점: 첫번째 빈 블록을 찾기가 쉬움
  - 단점
    - 모든 비트벡터를 주기억장치에 유지해야만 효율적
    - 디스크 크기가 큰 경우 주기억장치 부담이 큼
- 연결 리스트
  - 빈 블록을 연결리스트 형태로 유지
  - 첫 빈블록에 대한 포인터를 운영체제가 주기억장치에 유지
- 그룹핑
  - 첫 빈 블록에 n개의 가용 블록 주소를 저장 (주소 블록)
  - n개의 주소중 $n-1$개는 실제 빈 블록을 가리키고 n번째 주소는 다른 주소 블록에 대한 번지

가용 공간 관리 (계속)

Slide 21



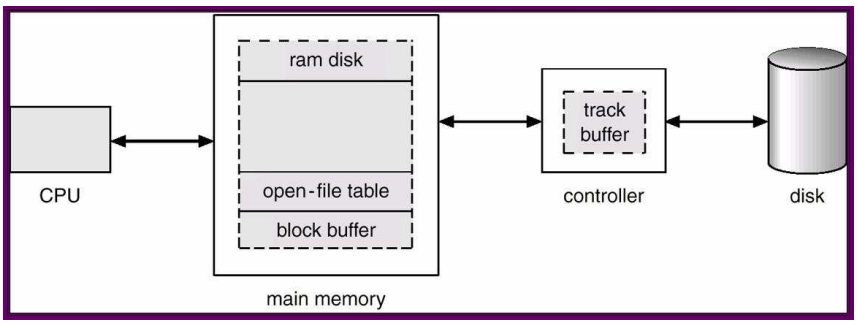
기타

Slide 22

- 램 디스크 와 디스크 캐시
  - 램 디스크
    - 주기억장치의 일부분을 디스크처럼 사용함
    - 램 디스크 장치 구동기는 모든 표준 디스크 명령을 받아들여 램상에서 명령 실행
    - 사용자는 램 디스크와 디스크의 차이를 느끼지 못함
    - 속도는 당연히 램 디스크가 디스크보다 빠름
    - 단, 전원이 나가면 램 디스크상의 모든 정보는 지워짐
    - 램 디스크는 사용자가 램을 디스크처럼 사용하고자 사용 (사용자가 제어함)
  - 디스크 캐시
    - 디스크 캐시는 운영체제가 디스크와의 빠른 동작을 위해 사용하는 메모리 (운영체제가 제어함)

기타 (계속)

Slide 23



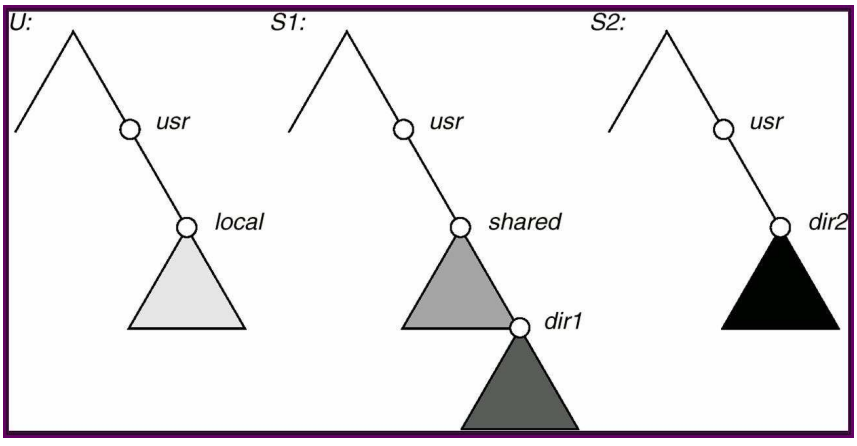
NFS(Network File System)

Slide 24

- NFS 소개
  - NFS란?
    - SunOS 나 Solaris에서 제공하는 근거리/원거리 파일 접근 시스템
    - UDP/IP 프로토콜 사용
    - 파일 시스템 사이에 투명한 공유를 허용
    - 클라이언트-서버 관계를 기반으로 함
    - 이기종의 기계, 운영체제 및 네트워크 환경에 독립적임
    - XDR(External Data Representation) 프로토콜의 최상위에 있는 RPC(Remote Procedure Call) 사용
  - NFS 마운트
    - 원격 기계의 특정 디렉토리를 투명하게 접근하기 위하여 마운트
    - 클라이언트와 서버간의 논리적 연결을 설정
    - 지역 파일 시스템의 디렉토리 상에 마운트됨
    - 마운트 요구는 RPC를 통해서 서버로 전달됨
    - 서버는 자신의 파일 시스템에서 NFS 허용 리스트를 관리함 (export list라하며 디렉토리별 기계이름, 사용자, 허용동작등이 설정됨)

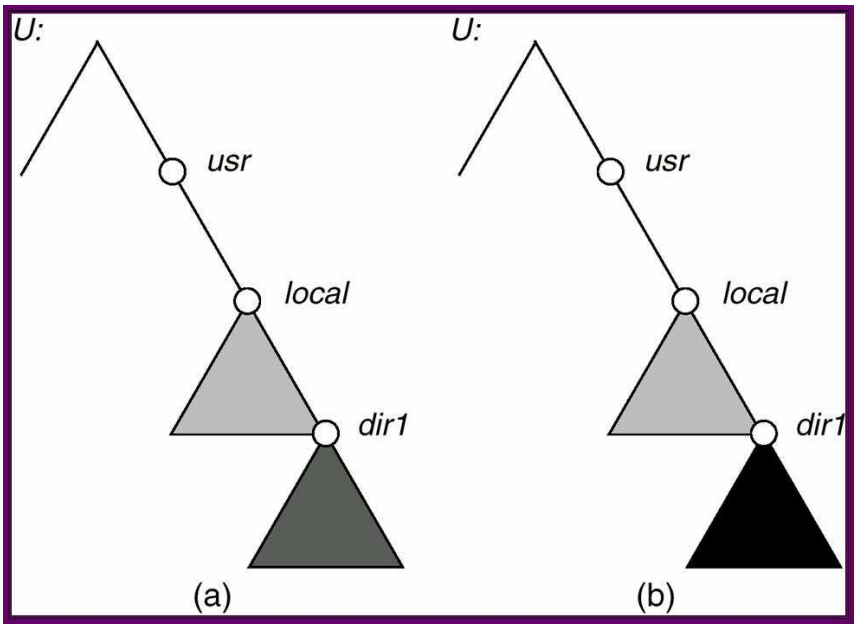
NFS(Network File System) (계속)

Slide 25



NFS(Network File System) (계속)

Slide 26



NFS(Network File System) (계속)

Slide 27

- NFS 프로토콜
  - 원격 파일 처리를 위해 몇개의 RPC를 제공
    - 디렉토리내의 파일 검색
    - 디렉토리 항목 읽기
    - 링크와 디렉토리 조작
    - 파일속성의 접근
    - 파일의 읽기와 기록
  - 수정된 데이터는 클라이언트에게 결과가 리턴 되기전 서버상에 기록
- NFS 구조
  - 시스템 호출 인터페이스: 보통 open, read, write, close 호출
  - VFS(Virtual File System) 인터페이스
    - 지역 파일 시스템과 원격 파일 시스템을 구분하고
    - 여러 종류의 지역 파일시스템을 투명하게 접근하게 함
    - 원격 파일 요청의 경우 NFS서비스를 호출함
  - NFS 서비스 계층
    - NFS 프로토콜을 구현한 것으로 VFS의 요청을 실행하고 결과를 리턴

NFS(Network File System) (계속)

Slide 28