파일시스템

1. 개요
2. 이미 존재하는 OS의 파일 시스템을 사용하지 않고 왜 독자적인 가상의 파일 시스템을 만들어 사용하는지에 대한 명확한 이해 필요
3. 이미 OS레벨에서는 이에 대한 이슈를 충분히 알고 있으므로 파일 처리에 대응되는 chunk 처리 인터페이스를 충분히 준비해 주고 있음

Ex> D3DXLoadSurfaceFromFile, D3DXLoadSurfaceFromFileInMemory

Ex> AddFontResourceEx, AddFontMemResourceEx

1. 대용량 게임 어플리케이션이라는 특성상 하드디스크 IO 시간을 줄이기 위한 어쩔 수 없는 선택

* 하드디스크의 arm 이동 거리 최소화
* 인코딩/압축으로 파일 크기를 줄일 수 있음

1. 클라이언트 보안 이슈

* 적의 손에 있는 인질의 생명력을 조금 더 높이기 위해 black-box 형태로 만들어 놓음
* 작정하고 달려드는 해커는 어쩔 수 없지만 양민들에게는 충분한 진입 장벽이 됨

1. 목표 및 과정
2. 파일 시스템 요구사항 파악
3. 설계
4. 자동화 인터페이스

**[ 원리만 알면 그냥 있는거 가져다 쓰세요^^ ]**

1. 파일 시스템 요구사항 파악
2. 파일 시스템을 사용하는 어플 개발자 입장에서는 있는지도 모를 정도로 사용자 환경을 OS 파일 시스템 사용과 동일하게 구성
3. 파일 데이터는 언제든 깨질 수 있으므로 chunk가 깨져도 복구 할 수 있는 수단 필요
4. 외부에서의 파일 삽입/참조, chunk에서의 파일 삭제/참조 기능 필요. 옵션으로 일괄 갱신 추가
5. 단편화 현상을 해결하기 위한 chunk 최적화 기능 필요(로드밸런싱 고려)
6. 예외 규칙 지정
7. Background loading
8. 설계
9. 하나의 파일 block은 header + body 형태
10. Block에 포함되어야 할 정보 : 블록 상태, 블록 크기, 헤더 크기
11. Header에 포함되어야 할 정보 : 파일의 상대 경로, 파일의 원본 크기/저장된 크기, time stamp
12. 파일 경로 정의

* 절대 경로를 사용할 경우 개발환경이 바뀌면 처리가 불가능

Ex> C:\Project\에서 개발된 파일을 D:\Test\ 경로에서 사용할 수 없음

* 따라서 대상 폴더의 root directory를 기준으로 한 상대경로 기반으로 구축
* 경로 3종세트(OS 제공의 working, module에 root directory라는 개념을 추가) 확보

5> block이 모여 chunk를 만들고, chunk가 모여 file system을 구성

1. 자동화 인터페이스

MkCore - Data management - File system 참조

MkCore - Main – Singleton - MkCore\_MkFileManager 참조

Background loading은,

MkCore - Main – Singleton - MkCore\_MkBackgroundLoader 참조

MkCore - Main – System class - MkCore\_MkBaseLoadingTarget 참조

예제는,

MkCore - Data management - Data node - MkCore\_MkDataNodeLoadingTarget 참조

툴은, MkFilePacker, MkChunkNavigator 참조