네오플

1. 자기소개

안녕하십니까 역배에 올인하는 남자 이 용 석 이라고 합니다.

이런 수식어를 붙인 이유는 친구들과의 내기 때문입니다.

3년전 네오플이라는 꿈이 생겼을 때

친구들은 너나 할 것 없이 못간다에 베팅했지만 저는 꿈에 배팅했습니다.

그리고 고등학생때까지 운동을 하며 얻은 승부욕과 끈기를 바탕으로 3년동안 꾸준히 노력

했고 그 결과 이자리에 있는 것 같습니다.

역배 대박을 위해 오늘 면접 최선을 다해보도록 하겠습니다.

감사합니다.

1. 게임프로그래머가 되고 싶은 이유

재수를 할 때 좋은 성적이라는 목표는 있었지만 구체적인 목표가 없어서

고민하던 시기에 고등학생때 같이 운동을 했던 형의 추천으로 게임공학과를 알게되었습니다.

형과 대화를 하며 "내가 좋아하는 게임을 내가 개발해본다?"라는 생각이 너무 매력적으로 다가왔고 크레이지 아케이드를 모작한 작품을 보며 "와 나도 저런거 만들어보고싶다"라고 생각해서

게임공학과에 진학했고 처음 팔라독 모작 개발에 참여해보니 상상대로 신기하기도 하고 성취감을 많이 느껴서 게임 프로그래머의 길을 선택했습니다.

1. 서버프로그래머가 되고 싶은 이유

싱글플레이 위주의 게임을 개발하다가 3학년때 처음 서버프로그래밍을 접하게 되었는데

친구들과 같이 게임을 하는 것을 좋아하는 제가 다른사람이 접속해서 같이 게임을 할 수 있는 기능을 구현하는 것이 너무 신기했고 재밌었습니다.

그래서 서버 개발자의 길을 택하게 되었습니다.

1. 프로그래밍할 때 나의 단점은 무엇인가?

작업을 하다가 흐름이 끊기고 다음에 다시 작업을 시작할 때 이전의 생각들을 잊어버리는 것이 가장 큰 단점이였습니다.

* 어떻게 해결하셨나요?

메모하는 습관을 들이기 시작했습니다.

작업 종료 30분 전에 이 때까지 했던 생각들에 대해 포스트잇에 정리해서 붙이기 시작했고 팀원들도 이런 습관이 좋아보였는지 화이트보드를 사서 벽에 붙여놓고 작업내용을 메모하면서 해결했습니다.

커밋 회수도 많아지고 개발 속도가 훨씬 빨라졌기 때문에 좋은 습관인 것 같습니다.

1. 사회생활은 어떤 편인가요?

사회생활은 항장 잘하는 편이라고 생각합니다.

* 왜 그렇게 생각하시나요?

사람들과 어울리는 것을 좋아하는 성격이라 학교생활 할 때는 학생회도 오래하며 원만한 관계를 유지했었습니다.

그리고 저는 어릴 때부터 알바를 많이 했는데 일하는 곳의 사장님들이 항상 저를 이쁘게 봐주셨기 때문에 사회생활은 잘하는 편이라고 생각합니다.

* 사례가 있나요?

최근에 학교근처 카페에서 일을 하며 만난 매니저님이 가장 생각이 많이나는데

남들이 꺼려하는 일도 웃으며 열심히 하는 모습을 좋게 보고있다고 하시며 카페 브랜드가 커지면 같이 일하자는 제안을 받았었던 기억이 가장 임펙트가 크게 다가왔던 것 같습니다.

1. 협업 시 가장 힘들었던 점

작업량의 차이로 발생하는 문제들을 해결하는 과정이 힘들었습니다.

저희는 사무실을 대여해서 같이 작업했기 때문에 서로의 작업량에 대해 잘 알고있었는데

작업량이 적던 친구가 거짓말로 작업을 회피하려는 모습을 보인적이 있었습니다.

거짓말에 너무 화가났지만 무작정 들이박는 것은 팀 분위기에 좋지않다고 생각해서

“쉬고 싶은 마음은 알겠지만 할 건 하고 쉬었으면 좋겠다, 내 작업이 끝나면 너꺼 도와줄

테니 속도좀 내보자” 라며 대화로 풀어가려 노력했고 이런 경험을 하면서 책임감을 많이 배웠

습니다.

졸업작품 때는 힘든 점이였지만 앞으로 더 큰 프로젝트를 진행할 때는 장점이 될것이라 생각합니다.

기술 면접 C/C++

* C와 c++의 차이를 아시나요?

그리고 객체를 이용한 출력, 함수를 이용한 출력이 다르고 표준 입출력 라이브러리가 다릅니다

C++은 namespace 개념을 사용개념을 사용합니다. C를 배울 때는 흔히 식별자의 이름은 같을 수 없다고 배웁니다. 하지만 C++에서는 식별자의 이름이 같을 수 있습니다.

* 포인터란 무엇인가요?

어떤 변수의 주소값을 저장하는 변수 입니다.

* 포인터와 래퍼런스의 차이점에 대해 아시나요?

래퍼런스는 포인터와 다르게 NULL 값을 허용하지 않습니다.

그래서 매개변수로 사용 될 때 포인터에 대해서는 NULL값에 대한 체크를 해야합니다.

그리고 래퍼런스는 선언과 동시에 초기화 해야 하며 값을 바꿀 수 없습니다.

* 각 각 언제 사용하나요?

매개변수에 NULL을 넘겨주거나 리턴값이 NULL이 될 수 있다면 포인터를 사용합니다.

나머지 경우엔 가능한 참조자를 쓰는 것이 좋습니다.

* 왜 좋나요?

가독성과 포인터를 사용하다 잘못된 메모리에 접근하는 상황을 막기 위해서 입니다.

* 인스턴스란 무엇인가요?

객체에 메모리가 할당되어 실제로 활용되는 실체를 말합니다.

* Null과 nullptr의 차이

NULL은 0이라는 상수로 취급되어 처리됩니다. 반면

Nullptr은 포인터 취급을 받는다는 차이점이 있습니다.

(매개변수로 쓸 경우 NULL은 int로 취급됩니다.)

* Abstract 키워드에 대해 아시나요?

추상 클래스로 만드는 키워드이며 파생 클레스에게 특정 함수를 반드시 구현하라고 지시하는 것입니다.

* 추상 클래스에 대해 설명해보세요

클래스 내부에 하나이상의 순수 가상함수를 포함하거나 abstract로 정의된 경우를 말합니다.

동작이 정의되지 않은 순수 가상 함수를 포함하기 때문에 인스턴스를 생성할 수 없다는 특징이 있습니다.

* 추상 클래스는 어떤 경우에 사용할 수 없나요?

변수 또는 멤버변수, 함수로 전달되는 인수 타입, 함수의 반환 타입, 명시적 타입 변환의 용도로 사용할 수 없습니다.

* L-value 와 R-value

L-value는 표현식 이후에도 사라지지 않고 지속되는 객체

R-value는 표현식 이후에 사라지는 임시 객체 입니다.

* Auto 자료형

초기화 값에 따라 데이터 타입을 정해주는 키워드입니다.

생성 시 변수를 초기활 할 때만 작동하며, 초기화 값을 사용하지 않고 생성된 변수는 기능을 사용하지 못합니다.

그리고 매개변수와 함께 사용할 수 없습니다.

* Volatile 자료형

해당 변수의 컴파일러 최적화를 제한하는 용도로 사용됩니다.

* 어떤 경우 사용되나요?

일반적으로 멀티스레드 환경에서 많이 사용되는 걸로 알고있지만 메모리 접근 순서를 보장하지 않아서 cpu가 실행시간에 메모리접근을 재배치할 수 있기 때문에 사용하면 안된다고 알고있습니다.

* Atomic 자료형

메모리 접근 순서를 보장하여 DataRace 문제를 해결할 수 있는 타입이다. 느리다는 단점

* 스마트 포인터

프로그래머가 메모리를 해제 하지 않았을 경우를 대비하여 사용이 끝난 메모리를 안전하게 자동으로 해제해 주는 역할을 하며 Unique\_ptr, Shared\_ptr, weak\_ptr이 있습니다.

Unique\_ptr은 하나의 스마트포인터만이 특정객체를 소유할 수 있도록 합니다.

그렇기 때문에 복사나 대입과 같은 공유를 허용하지 않는 스마트 포인터 입니다.

Shared\_ptr은 참조 카운팅을 기반으로 동작하는 스마트 포인터이며 하나의 객체를 다른 Shared\_ptr과 공유할 수 있습니다.

하지만 두객체가 서로를 참조하여 참조 카운트가 0이 되지못해 스스로 메모리 해제가 불가능한 상황을 말하는 순환참조가 발생할 수 있다는 단점이 있습니다.

이를 해결한 것이 Weak\_ptr인데 sharted\_ptr의 객체만 참조할 뿐, shared\_ptr의 참조 카운트를 증가시키지 않는 방식입니다.

* 물어보면 대답

Shared\_ptr에 강한 참조 카운트와 약한 참조 카운트가 있는데 객체 생명 주기에 관여하는 강한 참조 카운트를 올리지 않을 뿐 약한 참조 카운트는 증가시킵니다.

* Constexpr 과 const 차이점

가장 큰 차이점은 const는 런타임 상수이고, constexpr은 컴파일 시간 상수식입니다.

그렇기 때문에 Constexpr은 잘못 사용시 컴파일 타임에 오류가 발생한다는 장점이 있습니다.

* Define 과 const 차이점

Define은 타입지정을 하지 않아도 되고 디버깅시에 확인이 불가능 하며

Const는 디버깅시에 확인이 가능하다는 차이가 있습니다.

* 함수호출 규약인 Cdecl 과 stdcall, fastcall

Cdecl은 함수 호출용으로 스택에 쌓은 파라미터를 호출한 함수에서 제거하고,

Stdcall은 호출 당한 함수 내부에서 스택 포인터 연산을 합니다.

즉, 스택 포인터의 연산 위치의 차이가 존재합니다.

* 호출된 쪽에서 스택을 정리하는 all은 가변인자를 허용하지 않고

호출자가 정리하는 cdcel은 가변인자를 허용합니다.

* 클래스와 구조체 차이점

기본접근지정자의 차이가 존재합니다.

구조체는 기본접근 지정자가 public이고 class에서는 private이라는 차이점이 있습니다.

* 템플릿

매개변수의 타입에 따라 함수나 클래스를 생성하는 매커니즘을 의미합니다.

템플릿을 사용하면 타입마다 별도의 함수나 클래스를 만들지 않고, 여러 타입에서 동작할 수 있는 함수나 클래스를 작성할 수 있습니다.

* 오버로딩과 오버라이딩의 차이점

오버로딩은 함수명이 같고 배개 변수의 타입이나 개수가 다른 함수를 허용하는 것 이고

오버라이딩은 부모 클래스에서 정의한 함수를 자식 클래스에서 재정의하는 것을 허용하는 것입니다.

* Inline 함수는 무엇입니까?

컴파일러가 함수를 호출하는 대신 그에 대응하는 함수 코드를 대체한다는 것을 의미하며 함수 호출없이 삽입된 함수 코드를 그 자리에서 처리하므로 함수 실행을 위해 다른 주소를 점프했다가 되돌아올 필요가 없어 속도면에서 유리합니다.

* Malloc, calloc, new 차이점

Malloc은 라이브러리가 제공하는 함수이지만 new는 언어가 제공하는 연산자 입니다.

Malloc은 기본적으로 사이즈를 매개변수로 받고, 반환타입이 void\*형 이므로 sizeof와 캐스트연산자의 도움을 받아야 하지만 new는 타입을 지정하면 됩니다.

그리고 New는 자동으로 생성자를 호출하여 초기화를 해 줄 수 있지만 malloc은 불가능합니다.

* Calloc

Calloc은 malloc과 비슷한 기능을 가지고 있지만 calloc은 malloc과 다르개 할당된 공간의 값을 모두 0으로 바꿉니다. 그러므로 초기화할 필요가 있을 경우 calloc을 사용하면 됩니다.

* 언제 malloc을 사용하고 new를 사용하나요?

재할당이 빈번하게 일어나는 경우 동적할당된 메모리 크기를 변경시켜주는 reallloc이 가능한 malloc을 사용하는 것이 옳다고 생각합니다.

* Realloc의 단점은 없나요?

새로운 영역을 할당 받더라도 기존의 내용을 잃어버리는 일은 없지만 할당한 메모리에 여러 개의 포인터가 공유하고 있다면 사용해서는 안됩니다.

왜냐하면 메모리가 부족해 확장을 하지 못하고 새로운 공간으로 옮긴다면 nullptr을 반환하고 원래 공간에 남아있는 주소를 잃어버리게 되기 때문입니다.(댕글링 포인터)

* 초기화 리스트

생성자가 길어지는 것을 줄일 수 있고 대입 연산자를 호출하지 않아 성능적으로 우수합니다.

그리고 const나 reference 변수와 같이 초기값이 필요한 변수를 초기화 할 수 있는 좋은 방법입니다.

* 가상함수

Virtual 키워드를 붙여 자식클래스에서 부모 클래스의 멤버 함수를 재정의할 수 있는 함수이며

자신을 호출하는 객체의 동적타입에 따라 실제 호출할 함수가 결정 됩니다.

* 부모 클래스의 소멸자를 가상으로 만들어주는 이유

다운캐스팅 한 경우 객체 소멸시 부모 클래스의 소멸자는 호출되지만 자식 클래스의 소멸자는 호출되지 않는 문제가 있기 때문입니다.

* 가상 함수 테이블

가상함수를 가지는 클래스에 생성되며 가상 함수 주소들이 배열 형태로 존재하는 테이블이다.

동작 시 호출할 함수의 목록을 가상함수 테이블에 미리 저장하고 실행 중에 객체와 해당 객체의 함수 주소를 찾는 방법으로 동작합니다.

* 절차지향형 프로그래밍(c) vs 객체지향형 프로그래밍(c++)

절차지향 프로그래밍이란 순차적인 처리가 중요시되며 프로그램 전체가 유기적으로 연결되도록 만드는 프로그래밍 기법입니다.

컴퓨터의 처리구조와 유사해 실행속도가 빠르다는 장점이 있지만 유지보수가 어렵고 실행 순서가 바뀌면 동일한 결과를 보장하기 어렵다는 단점이 있습니다.

객체지향 프로그래밍이란 데이터를 객체 단위로 나누어 그 객체들 가느이 상호작용을 통해 로직을 구성하는 프로그래밍 방식입니다.

장점으로는 상속을 통한 확장으로 코드 재사용에 용이하고 유지보수 시에 수정할 객체 내부의 매서드만 수정하면 된다는 장점이 있지만 느린 처리속도와 객체가 많으면 용량이 커진다는 단점이 있습니다.

* Call by Value 와 Call by Reference 차이

Call by value는 인자로 받은 값을 복사하여 처리 하기 때문에 원래 값이 보존된다는 장점이 있지만 복사를 하기 때문에 메모리 사용량이 늘어난다는 단점이 있고

Call by Reference는 복사하지 않고 직접 참조하기 때문에 빠르지만 원래 값이 영향을 받을 수 있다는 단점이 있다.

* Call by pointer

Call by Reference와 같은 방식이지만 nullptr 검사를 해야한다는 단점이 있다.

* 객체지향 언어 – OOP 특징

1. 추상화

* 객체들의 공통적인 속성과 기능을 중심으로 추상화 한다.

1. 캡슐화

* 실제로 구현되는 부분을 외부에 드러나지 않도록 정보를 은닉할 수 있다.
* 객체가 독립적으로 역할을 할 수 있도록 데이터와 기능을 하나로 묶어 관리한다.
* 데이터를 보이지 않고 외부와 상호작용을 할 때는 메소드를 이용하여 통신을 한다.

1. 상속성

* 부모 클래스가 자식 클래스에게 속성과 기능을 물려주어 속성과 기능을 동일하게 사용하는 것
* 기존 코드를 재활용 해서 사용함으로써 객체지향 방법의 중요한 기능 중 하나에 속한다.

1. 다형성

동일한 요청에 의해 서로 다른 방식으로 응답하는 것

* 클래스 4대 속성

1. 멤버 변수의 기본접근지정자를 private으로 두어 외부로부터 원치 않는 데이터변겨을 막는 은닉화
2. 데이터화 함수를 하나로 묶어 사용하는 캡슐화
3. 부모 클래스의 기능을 자식클래스에게 물려주는 상속성
4. 각기 다른 객체가 동일한 메시지를 통해 각기 다른 기능을 수행하는 다형성

* 정적 라이브러리 lib / 동적 라이브러리 dll

정적 라이브러리는 컴파일 시 실행파일에 포함되고, 동적 라이브러리는 실행 시 실행 파일에 포함됩니다.

정적 라이브러리는 실행 파일에 함수의 코드가 복사되어 실행 파일의 크기가 커지지만 컴파일 후에는 lib 파일이 없어도 실행 가능하고,

동적 라이브러리는 라이브러리의 기능을 필요할 때만 참조하여 기능을 호출할 수 있도록 만든 파일이며 런타임에 함수를 호출 합니다.

* 장단점에 대해 아시나요?

DLL은 라이브러리를 한 번 구축해 놓으면 다시 만들 필요가 없어 개발 속도가 빨라지고

소스 유출을 막을 수 있다는 장점이 있지만 DLL 파일이 손상되거나 없어진다면 프로그램 실행에 큰 영향을 줄 수 있다는 단점이 있습니다.

LIB는 속도가 빠르고 작성이 간단하다는 장점이 있지만 실행 파일의 크기가 커지고 함수의 재사용이 코드레벨에서만 가능하다는 단점이 있습니다.

* 공통점은요?

코드 작성시 둘 다 헤더파일과 .lib 파일을 포함시켜야 한다.

* Static 멤버함수에 const를 붙일 수 없는 이유에 대해 아시나요?

Const 인스턴스는 this포인터에 영향을 받는데 static 멤버함수에서는 this포인터를 사용할 수 없기 때문입니다.

* 콜백함수란 무엇인가요?

다른 함수를 매개변수로 사용되는 함수, 어떤 이벤트에 의해 호출되는 함수

* 빈클래스의 크기는 몇인가요?

클래스는 비어 있어도 생성이 된다.

객체를 구분할 수 있는 것은 시작 주소인데 아무것도 존재하지 않아 0바이트를 유지하게 되면 다른 것과 구별할 수 없기 때문에 최소단위인 1바이트를 할당합니다.

* 가상함수가 존재하는 클래스도 똑같나요?

가상함수만 존재한다면 포인터의 크기가 나오는데 가상함수 포인터가 추가되기 때문입니다.

* 전방선언을 하는 이유는 무엇인가요?

헤더파일에 헤더파일을 추가하게 되면 컴파일 시간이 느려지기 때문에 사용합니다.

* 단점은 없나요?

전방선언한 클래스가 있다는 것만 알려주는 방식이기에 정적할당과 상속이 불가능합니다.

전방선언한 클래스가 있다는 것만 알려주는 방식이기에 정적할당과 상속은 불가능 합니다.

* 디자인 패턴

1. MVC패턴

* Model/View/Controller로 역할을 구분해서 설계하는 디자인 패턴이며 역할에 따라 확시랗게 분리하여 유지보스를 용이하게 하고 프로그램의 확장성과 유연성을 높일수 있다는 장점이 있는 패턴입니다.

1. Mediator 패턴

여러 객체가 서로 상호작용이 많은 경우 유지보수나 확장에 불리합니다.

이런상황의 객체들을 캡슐화 함으로써 결합력을 낮추는 가능을 하는 패턴입니다.

하지만 특정 로직에 맞춰져있기 때문에 재사용이 힘들다는 단점이 있습니다.

* 포폴에 CMediator라는 클래스가 있는데 Mediator 패턴을 활용하여 구현한 것이 맞습니까?

사실 처음 구현하고 사용했을 때는 Mediator 패턴이라고 생각했는데 객체간의 상호작용이

많이 없다는 점에서 Mediator패턴을 사용했다기 보다는 유사한 기능을 하는 객체들을 관리하는 하나의 클래스를 만들었다. 라고 보는 것이 맞다고 생각합니다.

1. 싱글톤 패턴

최초 한번만 메모리를 할당하고 그 메모리에 인스턴스를 만들어 사용하는 디자인 패턴이다.

그렇기 때문에 메모리 낭비를 방지할 수 있고 전역이기 때문에 다른 클래스의 인스턴스들이 데이터를 공유하기 쉽다는 장점이 있지만

싱글톤 인스턴스가 너무 많은 일을 하거나 많은 데이터를 공유시킬 경우 다른 클래스간에 결합도가 높아지게 되어 유지보수가 어려워진다는 단점이있고

멀티스레드 환경에서 동기화 문제가 발생할 수 있다.

* 멀티스레드 환경에서 어떻게 해결했나요 ?

DeltaTime을 업데이트하는 클래스를 싱글톤으로 사용하였기 때문에 매 프레임 연산을 해줘야 했습니다.

그래서 lock을 사용하는 방법이 아니라 멀티스레드가 돌기전에 미리 인스턴스를 호출하여 만들어 버리는 방식으로 해결했습니다.

* 상태패턴과 FSM 패턴에 대해 아시나요?

상태패턴은 같은 함수라도 상태에 따라 다른 결과를 내는 패턴입니다.

기초클래스에서는 행동들을 인터페이스화 시키고 실제 구현은 유도클래스에 정의해 사용하는 방식입니다.

FSM패턴은 특정 객체의 상태를 관리하기 위한 패턴으로 이벤트를 받고 그에 따라 현재상태에서 다음 상태로의 전이가 이뤄지는 방식입니다.

* 캐스팅 연산자

|  |  |  |
| --- | --- | --- |
|  | static\_cast (컴파일 타임) | dynamic\_cast(런타임 타입) |
| 특징 | - 논리적으로 변경 가능한 경우에만 변경 허용  - 컴파일타임에 형변환이 가능한지 검사한다. | - 런타임타임에서 안정성을 검사한다. |
| 차이점 | - 사용자가 정의한 클래스가 아닌 C++내부에서 이미정의된 기본 자료형간의 형변환을 하고 싶을 때 사용 가능  - 부모 클래스에서 자식 클래스로 형변환 허용  - 자식 클래스에서 부모 클래스로 형변환 허용 | - 기본 자료형으로 형변환 불가능  - 자식 클래스에서 부모 클래스 형변환 허용  - 하나 이상의 가상함수를 가진 다형성 클래스에 한해서 부모 클래스에서 자식 클래스로 형변환을 허용한다.  - 업케스트 허용  - 다운케스트 제약 |
|  | parent\* \_parent = new child();  child\* \_child = static\_cast<child\*>(\_parent);허용  parent\* \_parent = new parent();  child\* \_child = static\_cast<child\*>(\_parent);허용 | parent\* \_parent = new child();  child\* \_child = static\_cast<child\*>(\_parent);불허  parent\* \_parent = new parent();  child\* \_child = static\_cast<child\*>(\_parent);불허  child\* \_ child = new child();  parent\* \_parent = static\_cast<parent\*>(\_child);허용 |

- const\_cast: const의 성질을 제거하고 싶을 때 사용합니다. const성향을 제거했지만 실제 데이터가 가지는 메모리 배열이 바뀌거나 하는게 아니기 때문에 처음부터 읽기전용으로 만들어진 메모리를 억지로 접근해서 바꾸려면 에러가 발생합니다. char\*를 매개변수로 받는 함수를 정의하는 경우가 종종 있는데 그렇게 되면 const char\* 타입의 리터럴 문자열은 전달하지 못하게 됩니다. 이때 사용하는 경우가 많습니다.

- reinterpret\_cast: 포인터/참조와 관련된 형변환만 지원합니다. 거의 C스타일의 형변환 수준으로 말 그대로 재해석 한다고 볼수 있습니다.

* Const

변수는 상황에 따라 그 값이 변경가능하지만 상수인 const로 지정을 하면 그 값을 절대로 바꿀수 없다. 변수와 달리 처음 상수를 정의할 때 값을 지정해 주어야 한다.

const int\* c c는 상수만을 가리키는 포인터라는 의미이다. C가 가리키는 대상은 변경이 가능하지만 가리키는 값은 변경할 수 없다.

int\* const c = &a; 포인터가 가리키는 위치는 변경할 수 없고 가리키는값은 변경이 가능하다.

* **동적 바인딩**

부모 클래스형 포인터로 자식 클래스를 가리키는 업캐스팅 상황에사 오버라이딩 된 부모 쿨래스의 메소드를 호출하게 되면 실행시간에 동적으로 자식 클래스의 메소드를 호출하는 것을 의미합니다.

* **템플릿 메타프로그래밍**

템플릿을 사용하는 프로그래밍 기법으로 컴파일러에게 프로그램 코드를 생성하도록 하는 방식이다. 이러한 기법은 컴파일 시점에 많은 것을 결정하도록 하여, 실행 시점 계산을 줄여줍니다.

STL

* 시퀀스 컨테이너, 연관 컨테이너, 어뎁터

(자료의 구조로 구분을 하면…)

시퀀스 컨테이너는 자료를 입력한 순서대로 저장하기 때문에 저장 검색 알고리즘이라고 불립니다. 많지 않은 양의 자료/검색속도가 중요하지 않은 경우에 사용되며 vector, list, string, dequue등이 이에 해당합니다.

연관 컨테이너는 일정한 규칙에 따라 자료를 조직화하여 저장하는 것을 말합니다. 자료를 정렬하여 저장하기 때문에 검색에 유리하고 많은 양의 자료/빠른 검색이 중요할 때 사용합니다. 대표적으로 map, set이 그에 해당합니다.

어뎁터 컨테이너는 시퀀스 컨테이너를 변형시켜 스택, 큐, 우선순위 큐 형태로 저장하는 것을 말합니다.

(메모리 상에서 자료를 구성하는 형태로 구분을 하면)

contuquius-Memory연속메모리: 동적할당된 하나의 메모리 단위에 데이터 요소를 저장

node-based노드기반: 동적할당된 하나의 메모리 단위에 하나의 요소만 저장합니다. 포인터로 이를 연경합니다. 자료 추가 삭제에 유리하며 순차적 접근만 가능해 랜덤 접근이 불가능 합니다.

* 컨테이너가 뭔가요?

같은 타입의 여러 객체를 저장하는 일종의 집합이라고 할 수 있습니다. 컨테이너는 클래스 템플릿으로 컨테이너 변변수 선언할 때 컨테이너에 포함할 요소의 타입을 명시할 수 있습니다.

* 이터레이터는요?

반복자는 컨테이너에 저장된 원소를 순회하고 접근하는 일반화된 방법을 제공합니다. 반복자는 컨테이너와 알고리즘이 하나로 동작하게 묶어주는 인터페이스 역할을 합니다. 이 반복자 덕에 알고리즘은 특정 컨테이너에 종속적이지 않고 독립적이면서도 언제든지 컨테이너와 결합하여 동작 할 수 있습니다.

* 컨테이너들의 사용용도에 대해 아시나요?

|  |  |  |  |
| --- | --- | --- | --- |
|  | array (불가변배열) | vector (가변배열) | list |
| 장점 | - 적은 양의 자료에 유리 | - 적은 양의 자료에 유리  - 크기 변경 가능  - 순차 접근 가능  - 랜덤 엑세스 가능 | - 중간 삽입/삭제 가능  - 크기 변경 가능  - 적은 양의 자료에 유리  - 순차 접근 가능 |
| 단점 | - 크기 변경 불가 | - 중간 삽입 삭제 불가능  - 검색 느림  - 많은 양의 자료 불리 | - 많은 양의 자료에 불리  - 랜덤 엑세스 불가능  - 검색 느림 |
| 특징 |  | - reserve제대로 안하고 실시간 puch\_back하면 안된다.  - clear가 제대로된 clear가 아님  - 범위외 참조하지 말 것 | - 기능적으로 list가 성능적으로 forward\_list가 가볍고 좋음 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | map | set |  |
| 장점 | - 많은 양의 자료에 유리  - 검색 속도가 빠름 | - 많은 양의 자료에 유리  - 검색 속도가 빠름 |  |
| 단점 | - 적은 양엔 오버헤드로 인해 손해 | - 적은 양엔 오버헤드로 인해 손해 |  |
| 특징 | - 이진탐색트리기반(레드블랙트리)  - 자동정렬  - key와 value로 pair형태  - 값을 넣을 때 최대한 insert 사용 | - 이진탐색트리기반  - 자동정렬  - key가 곧 value |  |

- 랜던엑세스의 같은 말: 임의 접근, 랜덤 접근, 비순차적 접근, 직접 접근

1. Unordered\_map

키값으로 정렬이 불필요하고 대용량 데이터를 처리할 때 사용합니다.

저는 유저들의 id를 키 값으로 하여 필요한 경우에만 id검색을 통해 사용했습니다.

1. 덱

사용해보진 않았지만 벡터의 단점을 보완한 컨테이너라고 알고있습니다.

벡터는 메모리가 부족하면 새로운 메모리를 할당해 그곳으로 복사를 하게 되지만 덱은 새로운 메모리를 추가하는 형식을 가지기 대문입니다.

하지만 중간에 삽입 삭제가 필요하면 벡터에 비해 성능이 떨어지고 연속된 공간이 아니기 때문에 포인터로 연산이 불리합니다.

* 선형구조와 비선형 구조로 나눠주세요

선형 구조 : 리스트, 스택, 큐 덱

비선형 구조 : 트리, 그래프

* Vector

벡터는 동적으로 확장 축소가 가능하며 배열처럼 연속적인 메모리 공간에 저장하기 때문에 개별 원소에 대한 접근 속도가 빠르고(+ 지역성이 높아 캐시 효율이 좋다) index로의 접근이 가능합니다.

하지만 확장 시 재할당 비용이 크기 때문에 적절한 크기의 reserve로 인한 메모리 확보가 중요합니다.

* 중간에서 삽입/삭제가 이뤄지면 왜 느린가요?

연속적인 배열 형태이기 때문에 중간에서 삽입삭제가 일어나면 한칸씩 당겨야 하는 상황이 발생하기 때문입니다.

* 확장시 비용이 왜 큰가요?

확장시에 기존 벡터의 용량에 1.5~2배 정도 확장된 메모리를 새로 할당하여 기존 벡터를 옮겨 담는 작업을 수행하기 때문입니다. 이를 위해 resize나 reserve를 사용합니다.

* Resize와 reserve는 뭔가요?

Reserve 는 메모리만 할당하고 초기화 하지 않으며 size에는 영향을 주지 않습니다.

Resize는 벡터에 적절한 수의 요소를 삽입,삭제하여 주어진 크기로 만드는 역할을 하며

사용시 초기화 합니다.

* Emplace\_back과 push\_back의 차이는 뭔가요?

Push\_back을 사용할 때는 vector에 값을 넣기 위해 바깥에서 임시 객체의 생성자 / 소멸자가 호출되고 vector내부에 복사되는 과정에서 또 객체가 한번 더 생성되지만

Emplace\_back은 내부에서 한번만 객체가 생성되므로 대부분의 경우 emplace\_back이 성능적으로 좋기 때문에 사용하였습니다.

* Vector array 차이

벡터는 순차 데이터 저장 컨테이너이고 배열은 인덱스 기반의 고정 크기 컨테이너입니다.

즉, Vector는 동적으로 크기가 변경가능하지만 배열은 불가능합니다.

* 벡터와 덱의 차이

덱은 늘어나야 할 크기만큼 할당되기 때문에 Vector에 비해 확장 비용이 작다는 장점이 있지만

테이블 전체가 연속된 메모리 공간이 아니므로 원소들 간의 포인터 연산이 불가능 하다는 단점이 있습니다.

* List란?

중간에 데이터 삽입이나 삭제가 자주 일어나고 데이터를 랜덤하게 접근하는 경우가 많지 않을 경우 유용하게 사용되는 컨테이너입니다.

컨테이너의 어느 위치에서도 삽입삭제가 상수시간이라는 장점이 있지만 인덱스에 의한 접근을 지원하지 않는다는 단점이 있습니다.

* Set이란?

고유한 key 값을 저장하는 연관 컨테이너이며 인덱스 접근을 허용하지 않지만 search와 삽입삭제에서 log n의 시간복잡도를 가집니다.

* Map이란?

고유한 key와 key의 Value값을 저장하는 연관 컨테이너이며 인덱스 접근을 허용하지 않지만 search와 삽입삭제에서 log n의 시간복잡도를 가집니다.

* Map이 내부적으로 어떤 구조로 되어있는지 아시나요?

균형이진탐색트리인 레드블랙트리로 구현되어 있습니다. 이진탐색 트리에서 최악의 경우 트리의 높이만큼 탐색 시간이 걸리는 상황을 해결하기 위해 균형잡힌 트리가 만들어지도록

5가지 조건을 가지고 있는데 각 노드는 빨간색, 검정색 중 하나이다.

루트노드는 검정색이여야 한다, 모든 리프 노드는 블랙이다(NIL노드), 레드노드의 자식은 블랙이다, 리프노드에서 루트 노드까지 가는 경로의 블랙 노드의 개수는 같다.

이 조건을 만족함으로써 트리의 높이를 log n에 바운드되도록해줍니다.

* Unoredered\_map map 차이점

맵은 균형 이진 트리를 기반으로 동작하며 언오더드는 해쉬 테이블 기반으로 동작합니다.

그래서 map은 key값에 따라 정렬되지만 언오더드는 key의 hash 값에 따라 저장되며 요소를 자동정렬하지 않습니다.

* Unordered\_map

해싱된 키를 가지고 배열의 인덱스로 접근하기 때문에 삽입, 삭제 검색이 매우 빠르다는 장점이 있지만 해시 충돌이 발생하면 성능이 떨어집니다.

* 해시충돌이 일어나는 이유는 뭔가요?

Key는 무한하지만 bucket의 사이즈는 유한하기도하고 key가 서로 다르더라도 동일 값을 반환할 수 있기 때문입니다.

* 해시 충돌을 최소화 하는 방법에 대해 아시나요 ?

버킷의 사이즈가 충분히 크거나, 버킷의 크기를 소수로 설계하는 것입니다.

잘은 모르지만 소수가 어떤 계산으로부터 나올 확률이 적기 때문으로 알고있습니다.

* 해시 충돌의 해결법에 대해서 아시나요?

체이닝과 개방주소법가 있습니다.

체이닝이란 value가 저장되는 공간을 가변 배열로 구현하여 충돌이 발생해 동일한 인덱스가 나오면 해당 인덱스에 추가로 이어서 저장하는 방법입니다.

개방주소법은 충돌 발생시 비어있는 인덱스에 value를 할당하는 방법입니다.

비어있는 인덱스를 탐색하는 방법으로는 다음인덱스를 확인하며 찾아가는 선형탐사, 선형탐사와 비슷하지만 폭을 이차식을통해 계산하는 제곱탐사, 또 다른 해시 함수를 사용해 증가 폭을 구하고 증가폭만큼 인덱스를 증가시키며 빈곳을 탐색하는 이중해싱이 있습니다.

* Queue

큐는 선입선출 방식의 자료 구조이며, 한쪽 끝에서 삽입 작업을, 다른 쪽 끝에서 삭제작업을 진행하는 구조입니다.

* 이진 탐색트리에 대해 아시나요?

이진탐색트리는 데이터 삽입, 삭제, 탐색 등이 자주 발생하는 경우에 효율적인 구조로, 이진 트리이면서 같은 값을 갖는 노드가 있어야 합니다.

왼쪽 서브 트리에 있는 모든 데이터는 현재 노드의 값보다 작고, 오른쪽 서브 트리에 있는 모든 노드의 데이터는 현재 노드의 값보다 크다는 특징이 있습니다.

OS & CS

* 가상 메모리란 무엇인가요?

메인 메모리의 부족한 용량을 보완하기 위해 각 프로그램에 실제 메모리 주소가 아닌 가상의 메모리 주소를 할당하는 방식입니다.

* 페이징 기법에 대해 아시나요?

페이징 기법이란 프로세스가 사용하는 메모리 공간을 잘게 나누어 비연속적으로 실제 메모리에 할당하는 메모리 관리 기법입니다.

* 특징에 대해 말해보세요

프레임 단위의 비연속적인 메모리 할당으로 동적메모리 할당에 따른 문제가 없습니다.

따라서 외부 단편화가 없습니다.

* 왜 외부단편화가 없나요?

프로세스를 작은 크기로 나누어 크기에 맞게 메모리를 할당하기 때문입니다.

* 단편화에 대해 설명해 보세요

내부 단편화와 외부 단편화가 있는데 내부 단편화는 메모리 공간이 할당된 프로그램의 크기보다 커서 사용되지 않고 남는 공간을 말하며,

외부 단편화는 메모리 공간이 할당될 프로그램의 크기보다 작아서 할당하지 못하고 비어 있는 모든 공간을 의미 합니다.

* 페이지 폴트란 무엇인가요?

가상메모리공간에는 존재하지만 실제 메모리에 없는 데이터나 코드에 접근을 시도하였을 경우 발생하는 현상입니다.

* 해결법은 뭐가 있나요?

자세히는 모르지만 페이지 폴트 처리기를 실행하여 원하는 페이지를 찾는다고 알고있습니다.

* 데드락에 대해 설명해 보세요

데드락이란 둘 이상의 프로세스들이 자원을 점유한 상태에서 서로 다른 프로세스가 점유하고 있는 자원을 요구하며 무한정 기다리는 현상을 말합니다.

한정된 자원을 여러 곳에서 사용하려고 할 때 발생한다.

* 데드락의 발생 조건에 대해 아시나요?

자원은 한 번에 한 프로세스 만이 사용할 수 있어야 한다는 상호배제,

최소한 하나의 자원을 점유하고 있으면서 다른 프로세스에 할당되어 사용하고 있는 자원을 추가로 점유하기 위해 대기하는 프로세스가 있어야 한다는 점유 대기,

다른 프로세스에 할당된 자원은 사용이 끝날 때까지 강제로 뺏을 수 없어야 한다는 비선점,

대기 프로세스는 순환형태로 자원줄을 대기한다는 의미의 순환대기가 있습니다.

이 네가지 조건이 동시에 성립할 때 발생합니다.

* 데드락의 예방법에 대해 아시나요?

앞서 말씀드렸던 4가지 필수 조건 중 하나 이상을 부정하여 방지할 수 있습니다.

한번에 여러 프로세스가 공유자원을 사용 하도록 하는 상호배제 부정

자원을 점유하고 있지 않을 때만 다른 자원을 요청할 수 있도록 하는 점유 및 대기 부정

모든 자원에 대한 선점을 허용한다는 비 선점 부정

자원에 고유번호를 할당하고 번호 순서대로 자원을 요구하도록 하는 순환대기 부정이 있습니다.

* 커널에 대해 설명해 보세요

운영체제의 모든 부분에 대한 기본 서비스를 제공하는 운영체제의 핵심으로 프로세스마다 자원을 얼마나 사용해야 하는지 결정해 주는 역할을 합니다.(스케줄링)

* 스케줄링이 뭔지 아시나요?

스케줄링이란 자원의 효율적인 사용을 위해 자원사용순서를 결정하는 작업입니다.

* 스케줄링의 종류에 대해 아시나요?

스케줄링은 할당된 cpu는 다른 프로세스가 강제로 빼앗을 수 없다는 특징을가진 비선점 스케줄링과 우선순위가 높은 프로세스가 cpu를 강제로 빼앗아 사용할 수 있는 비선점 스케줄링이 있습니다.

* 각각의 장단점에 대해 아시나요?

선점 스케줄링은 비교적 응답이 빠르다는 장점이 있지만 처리 시간을 예측하기 힘들고 높은 우선순위 프로세스들이 계속 들어오는 경우 오버헤드가 발생합니다.

실시간 응답환경에 적합

비선점 스케줄링은 모든 프로세스에 대한 요구를 공정하게 처리할 수 있지만, 짧은 작업을 수행하는 프로세스가 긴 작업 종료 시 까지 대기해야 할 수 있습니다.

처리 편차가 적은 특정 프로세스 환경에 용이합니다.

* 스케줄링 기법에 대해 아시나요?

선점형 스케줄링 기법에는 시분할 시스템을 위해 설계된 라운드 로빈 스케줄링이 있습니다.

프로세스들 사이에 우선순위를 두지 않고, 순서대로 시간단위로 cpu를 할당하는 방식의 알고리즘입니다.

Cpu를 독점하지 않고 공평하게 이용할 수 있다는 장점이 있지만 시간 할당량이 너무 크면 비선점 FIFO기법과 같아지게 되고 너무 작으면 컨텍스트 스위칭의 오버헤드가 커진다는 단점이 있습니다.

비 선점형 프로세스에는 FIFO 방식이 있는데 이름 그대로 선입선출의 방식으로 아무리 중요한 작업이 있다 하더라도 먼저 들어온 작업이 끝나기 전까지 실행될 수 없는 방식으로 비효율적인 방식입니다.

* 다른 스케줄링 기법은 모르시나요?

비선점형에는 평균대기 시간을 최소화 하기 위해 CPU 점유시간이 가장 짧은 프로세스에 CPU를 먼저 할당하는 방식인 SJF가있습니다.

실행시간이 긴 프로세스는 실행 시간이 짧은 프로세스에게 할당 순위가 밀려 연기 상태에 빠질 수 있다는 단점이 있습니다.

이를 보완하는 기법으로는 HRN 기법이 있습니다.

우선순위를 대기시간 과 서비스시간을 더한 값에 서비스 시간을 나누어 계산하여 높은 것부터 실행하는 방식입니다.

그리고 선점알고리즘으로는 비선점 스케줄링인 SJF 기법을 선점 형태로 변경한 기법인 SRT 기법이 있습니다.

시간이 가장 짧은 프로세스에 CPU를 먼저 할당하는 방식으로 차이점은 단지 선점형으로 바뀌어 중요한 프로세스가 있으면 점유시간이 길어도 먼저 실행 시킬 수 있는 권한이 생겼다는 것입니다.

* 커널 오브젝트에 대해 설명해 보세요

커널에서 관리하는 중요한 정보를 담아둔 데이터ㅓ 블록을 의미합니다.

생성 시 마다 핸들이라는 정수 값이 부여되며, 이것을 통해 특정 커널 오브벡트를 정할 수 있습니다.

* 핸들의 역할은 무엇인가요?

프로그래머가 커널 오브젝트에 직접 접근할 수 없기 때문에 간접적으로 접근하기 위해 사용하는 방식이 핸들입니다.

* 유저 영역과 커널 영역에 대해 설명해 보세요

프로그램이 동작하기 위해 사용되는 메모리 공간을 유저 영역이라고 하며, 하나의 프로세스에 할당되는 총 메모리 공간 중 유저 영역을 제외한 나머지 영역을 커널 영역이라고 합니다.

* 컨텍스트 스위칭이 무엇인가요?

Cpu가 하나의 프로세스를 실행하고 있는 상태에서 인터럽트 요청에 의해 다음 우선 순위의 프로세스가 실행되어야 할 때 기존의 프로세스의 상태 또는 레지스터 값을 저장하고 cpu가 다음 프로세스를 수행하도록 새로운 프로세스의 상태 또는 레지스터값을 교체하는 작업을 컨텍스트 스위칭이라고 합니다.

* 필요한 이유에 대해 아시나요?

여러 개의 작업을 효율적으로 처리하기위해 멀티 프로세싱, 멀티스레드를 사용하기 위해 필요합니다.

* 언제 발생하는지는 아시나요?

인터럽트 신호가 왔을 때 발생합니다.

* 인터럽트가 무엇인가요?

Cpu가 프로그램을 실행하고 있을 때 실행중인 프로그램 밖에서 예외 상황이 발생하여 처리가 필요한 경우 cpu에게 알려주고 예외를 처리할 수 있도록 하는 것입니다.

* 모든 인터럽트 신호에 컨텍스트 스위칭이 일어나나요?

그건 아닙니다. 자세히는 모르지만 입출력을 요청할 때, cpu 사용시간이 만료되었을 때,

자식 프로세스를 만들 때, 인터럽트 처리를 기다릴 때가 있습니다.

* 컨텍스트 스위칭에 발생하는 비용에는 뭐가 있는지 아시나요?

캐시 초기화, 메모리 매핑 초기화

* PCB는 무엇인가요?

PCB란 프로세스의 정보를 저장하는 블록을 의미하며 프로세스 번호, 상태, 레지스터 등등이 저장됩니다.

* 컨텍스트 스위칭 비용은 프로세스와 스레드가 똑같나요?

아닙니다 스레드는 스택 영역을 제외한 모든 메모리를 공유하기 때문에 컨텍스트 스위칭 발생 시 스택 영역만 변경을 진행하면 되기 때문에 프로세스가 더 많이 듭니다.

* 컨텍스트 스위칭 때는 해당 cpu가 아무 일도 하지 못하기 때문에 잦아지면 오버헤드가 발생해 성능이 떨어지게 됩니다.

그리고 cpu의 개수보다 실행 중인 상태의 스레드의 개수가 적거나 같을 경우 컨텍스트 스위칭이 발생할 일이 없다.

* 기아 현상에 대해 아시나요?

우선 순위로 작업을 처리할 때, 우선 순위가 낮은 작업은 영원히 처리되지 않는 문제를 의미 합니다.

* 해결방법에 대해 아시나요?

오랫동안 기다린 프로세스의 우선순위를 높여줌으로써 해결하는 에이징 기법과

요청순으로 처리하는 요청 큐가 있습니다.

* 에이징 기법의 종류에는 뭐가 있을까요?

비선점 스케줄링 알고리즘인 SJF HRN FIFO가 있습니다.

* 데드락과 기아의 차이점에 대해 아시나요?

데드락은 여러 프로세스가 동일 자원 점유를 요청할 때 발생하고

기아현상은 여러 프로세스가 부족한 자원을 점유하기 위해 경쟁할 때 발생합니다.

* 메모리의 4대 영역에 대해서 아시나요?

Code – 실행할 프로그램의 코드가 저장되는 영역이며 프로그램 시작부터 종료 시 까지 메모리에 남아있습니다.

Data – 프로그램의 전역변수와 정적변수가 저장되는 영역으로 프로그램 시작과 종료 시 할당 소멸되는 공간입니다.

Heap – 런타임에 크기가 결정되며 사용자가 관리할 수 있는 메모리 영역으로 사용자에 의해 메모리 공간이 동적으로 할당 및 소멸되는 공간입니다.

낮은 주소(위)에서 높은 주소(아래)로 할당됩니다.

Stack – 컴파일 타임에 크기가 결정되며 지역변수, 매개변수 등의 임시변수가 저장되는 영역입니다. 힙과 반대로 높은 주소에서 낮은 주소로 저장됩니다.

* 오버플로우에 대해 아시나요?

한정된 메모리 공간이 부족하여 메모리안의 데이터가 넘쳐 흐르는 현상입니다.

힙은 위에서 아래 스택은 아래에서 위이므로 상대영역을 침범하는 일이 발생하는데 이를 오버 플로우라고 합니다.

* 스레드 세이프란 무엇인가?

여러 스레드가 동시에 사용되어도 안전하다는 것을 의미하며 특정 함수나 변수가 여러 스레드에서 호출했을 때 같은 결과가 보장되어야 한다는 것을 뜻합니다.

* 방법으로는 세마포어나 뮤텍스
* 경쟁상태란 무엇인가요?

경쟁 상태란 여러 프로세스가 공유 데이터를 동시에 접근할 때 공유 데이터에 대한 실행순서에 따라 실행결과가 달라지는 상황입니다.

* 임계영역이란 무엇인가요?

임계 영역이란 프로세스간에 공유자원을 접근하는데 있어서 문제가 발생하지 않도록 한번에 하나의 프로세스만 이용하게끔 보장해줘야 하는 영역입니다.

* 문제란 무엇인가요?

여러 프로세스가 공유 데이터를 동시에 접근할 때 공유 데이터에 대한 실행순서에 따라 결과가 달라지는 상황을 뜻하는 경쟁 상태(데이터 레이스)가 발생할 수 있습니다.

* 문제를 해결하기 위한 조건에 대해 아시나요?

하나의 프로세스가 임계영역에 있다면 다른 프로세스는 들어갈 수 없어야 하고

임계 영역에 들어간 프로세스가 없는 상태에서 들어가려 하는 프로세스가 여러 개라면 어떤 것이 들어갈지 결정해 주어야 합니다.

그리고 다른 프로세스의 기아 현상을 방지하기 위해, 한번 임계 구역에 들어간 프로세스는 다음 번 임계 영역에 들어갈 때 제한을 두는 방법이 있습니다.

* 방법에는 뭐가 있을까요?

뮤텍스와 세마포어가 있습니다.

* 뮤텍스과 세마포어는 무엇인가요?

뮤텍스란 오직 하나의 스레드만이 동일한 시점에 뮤텍스를 얻어 임계영역에 들어올 수 있게 하는 것으로 오직 이 스레드만이 임계 영역에서 나갈 때 뮤텍스를 해제할 수 있습니다.

세마포어란 뮤텍스와는 다르게 락을 걸지 않은 스레드도 락을 해제할 수 있습니다.

둘의 가장 큰 차이점은 동기화 대상의 개수인데 뮤텍스는 하나뿐일 때, 세마포어는 하나이상일 때 사용합니다.

뮤텍스는 세마포어의 부분집합이다.

* 뮤텍스를 사용한 동기화 기법에 대해 자세히 말씀해주세요

뮤텍스라는 커널 객체를 이용해서 동기화를 하는 방법으로 커널 객체를 사용하기 때문에 프로세스 사이에서의 동기화도 가능하며 스레드 사이에서의 동기화도 가능합니다.

* 커널 객체를 이용해 동기화를 한다고 하셨는데 자세히 말씀해 주실 수 있나요?

스레드의 동기화 방법에는 유저모드, 커널모드 동기화가 있습니다.

유저모드 동기화란 커널의 자원을 이용하지 않고 수행하기 때문에 커널모드로 전환이 이뤄지지 않아 성능상에 이점이 있습니다.

커널모드 동기화에서는 커널모드로의 전환이 이루어 지기 때문에 성능상의 저하가 발생합니다.

* 그럼 커널모드 동기화는 사용하면 안되겠네요?

아닙니다 자세하게는 모르지만 다른 프로세스에 존재하는 스레들간의 동기화도 가능하다는 장점이 있습니다.

* 캐시 메모리에 대해 아시나요?

캐시 메모리는 주기억장치에서 자주 사용하는 프로그램과 데이터를 저장해 두어 속도상 이점을 가지는 메모리이며 속도가 빠른 장치와 느린 장치간의 속도 차에 따른 병목현상을 줄이는 역할을 합니다.

* 캐시 히트와 캐시 미스에 대해 설명해보세요

캐시 히트란 cpu가 참조하고자 하는 메모리가 캐시에 존재하는 경우를 뜻하며

캐시 미스는 반대의 경우를 뜻합니다.

* 캐시의 지역성에 대해 아시나요?

기억장치 내의 정보를 균일하게 접근하는 것이 아닌 특정 부분을 집중적으로 참조하는 특성입니다.

* 지역성의 종류에 대해 아시나요?

공간적 지역성과 시간적 지역성이 있습니다.

공간적 지역성은 프로그램이 명령어를 순차적으로 실행하는 경향 때문에 발생하며

시간적 지역성은 반복문을 통해 같은 주소를 반복해서 참조한 경우에 발생합니다.

* 프로세스와 스레드의 차이에 대해 설명해 보세요

프로세스는 운영체제로부터 시스템 자원을 할당 받은 작업의 단위를 뜻하며 각각 독립된 메모리 영역을 할당하며 기본적으로 최소 1개의 메인 스레드를 가집니다.

또 각 프로세스는 별도의 주소공간에서 실행되며 다른 프로세스의 데이터에 접근이 불가능합니다.

스레드는 프로세스 내에서 실행되는 여러 흐름의 단위를 뜻하며

프로세스내에서 스택만 할당 받고 코드, 힙,데이터 영역을 공유합니다.

별도의 레지스터와 스택을 가지지만 힙 메모리는 사용하지 않습니다.

* 그럼 멀티 프로세스와 멀티 스레드에 대해 아시나요?

멀티 프로세스란 하나의 응용프로그램을 여러 개의 프로세스로 구성하여 각 프로세스가 하나의 작업을 처리하는 방식으로 여러 개의 자식 프로세스 중 하나에 문제가 발생해도 다른 프로세스에 영향을 끼치지 않는 다는 장점이 있지만 독립된 메모리 할당으로 인해 컨텍스트 스위칭 시에 캐쉬의 데이터를 모두 리셋 해야만 한다는 단점이 있습니다.

멀티 스레드는 하나의 응용프로그램을 여러 개의 스레드로 구성하고 각 스레드로 하나의 작업을 처리하는 것을 말합니다.

스레드 간 자원공유가 간단하고 작업량이 작아 컨텍스트 스위칭이 빠르고 스택을 제외한 공유에 부담이 적으므로 빠른 응답시간을 가진다는 장점이 있지만 디버깅이 힘들고 하나의 스레드에 문제가 발생할 경우 다른 스레드에도 영향이 있다는 단점이 있습니다.

* 프로세스의 상태에 대해 말해보세요

레디, 러닝, 블록 상태로 구분할 수 있는데

레디는 cpu를 할당 받을 준비가 된 상태를 뜻하며

러닝은 cpu를 할당 받아 실행되는 상태,

블록은 이벤트나 입출력 대기 신호 발생을 기다리는 상태입니다.

* 프로그램 실행 과정에 대해 설명해보세요 or 실행파일 생성과정에 대해 설명해 보세요

전처리기 -> 컴파일러 -> 어셈블러 -> 링커 순입니다.

전처리기는 define이나 include 파일을 컴파일 가능한 코드로 바꾸는 역할을 하며

컴파일러는 코드를 어셈블리어로 바꿔주는 역할을 합니다.

그리고 어셈블러는 실행되기 앞서 바이너리 코드로 바꿔주는 역할을 하고

링커는 프로그램 내에서 참조하는 함수나 라이브러리들을 하나로 묶는 작업을 합니다.

이 과정이 끝나면 실행파일이 생성됩니다.

* 실행파일이 메모리에 올라간 다음 어떻게 되나요?

메모리에 올라간 명령어들은 cpu에 의해서 순차적으로 실행되는데 Fetch,Decode,Excution의 과정을 거칩니다.

Fetch는 메모리상에 존재하는 명령어를 cpu로 가져오는 작업을 뜻하고

Decode는 가져온 명령어를 cpu가ㅏ 해석하는 단계이며 무슨 일을 하라는 명령어인지 분석하는 단계입니다.

Excution은 해석된 명령어의 명령대로 cpu가 실행하는 단계입니다.

* 32비트 시스템과 64비트 시스템의 차이에 대해 설명해 보세요

32비트 시스템은 한 번에 처리할 수 있는 크기가 4바이트이며 메인 메모리에 등록할 수 있는최대 범위가 4GB이고

64비트 시스템은 한 번에 처리할 수 있는 크기가 8바이트이며 메인 메모리에 등록할 수 있는 최대 범위가 16EB(엑사 바이트)입니다.

2의 32승 , 2의 64승

알고리즘

* 테이블이(가) 표시된 사진

  자동 생성된 설명
* 정렬 알고리즘

1. 버블 정렬 – 내부/비교식 정렬

<시간복잡도> 최상, 평균, 최악 모두 n2

서로 인접한 원소를 검사하여 정렬하는 알고리즘으로 구현이 간단하고 직관적이지만

성능이 좋지 않습니다.

* 왜 좋지 않나요?

순회하며 정렬을 하는 방식이기도 하고 특정 요소가 최종 정렬위치에 있더라도 교환이 발생할 수 있기 때문입니다.

1. 선택 정렬 - 내부/비교식

<시간복잡도> 최상, 평균, 최악 모두 n2

인덱스와 뒤의 값 중 최소값을 찾아 맨 앞으로 옮기는 방식으로 정렬을 진행 하며

자료의 이동회수가 미리 결정되지만 안정성을 만족하지 못한다는 단점이 있습니다.

* 왜 안정성을 만족하지 못하나요?

값이 같은 레코드가 있는 경우에 상대적인 위치가 변경될 수 있기 때문입니다.

1. 삽입정렬 – 내부

<시간복잡도> 최선(이동없이 한번의 비교) n , 평균 n2

최악(입력자료가 역순인 경우) n2

배열의 모든 요소를 앞에서부터 차례대로 이미 정렬된 배열 부분과 비교하여 자신의 위치를 찾아 삽입함으로써 정렬을 완성하는 알고리즘입니다.

(두번째 값부터 key로 설정하여 왼쪽 자료들과 비교)

레코드의 수가 적을 경우 알고리즘이 간단하여 다른 복잡한 방법보다 유리할 수 있다는 장점이 있지만 비교적 많은 레코드들의 이동을 포함하며 레코드 수가 많고 크기가 클 경우 성능이 떨어진다는 단점이 있습니다.

1. 병합정렬 – 외부

하나의 리스트를 두개의 균등한 크기로 분할하여 두개의 정렬된 부분 리스트를 합한사며 정렬하는 방법으로 만들어진 두개의 리스트를 처음부터 끝까지 비교하며 작은 값을 새로운 리스트에 넣는 방식입니다.

레코드의 크기가 큰 경우 이동횟수로 인해 비효율적이라는 단점이 있지만 데이터 분포에 의한 영향이 적고 크기가 연결리스트를 사용하여 구현할 경우 크기가 큰 레코드를 정렬에 효율적인 알고리즘이다.

* 왜 연결리스트를 사용하면 크기가 큰 레코드들 정렬에 효율적인가요?

링크 인덱스만 변경하면 되므로 데이터 이동의 부하가 작아지기 때문입니다.

<시간복잡도> 전부 n log n

1. 퀵정렬

하나의 리스트를 피벗을 기준으로 작은 것은 왼쪽 큰 것은 오른쪽으로 분할하고 분할된 리스트를 정렬한 다음 합쳐 전체가 정렬되는 방식입니다.

속도가 빠르며 추가 메모리 공간을 필요로 하지 않는다는 장점이 있지만

정렬된 리스트에 대해 불균형 분할이 일어나기에 수행시간이 올라갑니다.

* 수행시간이 올라가는 것에 대한 해결법은 있나요?

피벗을 최대한 균등하게 분할 할 수 있는 데이터로 설정하는 것으로 해결할 수 있습니다.

<시간복잡도> 최선 n log n, 평균 n log n, 최악 n^2

DB

* 데이터베이스와 DBMS가 무엇인가요?

데이터베이스는 특정 목적을 위해 데이터가 저장되고 구성되는 관련 데이터의 조직화 된 모음입니다.

DBMS는 데이터베이스 관리 시스템을 의미하며 효율적으로 관리할 수 있도록 하는 응용프로그램을 뜻합니다.

* RDBMS에 대해 아시나요?

관계형 데잍터베이스 관리시스템을 뜻하며 데이터를 열과 행의 형태로 저장하는 방식을 말합니다.

범용적이고 성능이 좋다는 장점이 있지만 반드시 스키마 규격에 맞춰야 하기 때문에 갱신이나 변경 시 비용(인덱스 생성과 스키마 변경)이 많이 든다는 단점이 있습니다.

* NoSQL에 대해서는 아시나요?

RDBMS와 달리 테이블 간 관계를 정의하지 않는 방식으로 일반적으로 JOIN도 불가능합니다.

스키마가 없기 때문에 유연한 데이터 구조를 가지고 확장에 용이하다는 장점이 있지만 데이터 중복이 발생할 수 있고 스키마가 존재하지 않기에 명확한 데이터 구조를 보장하지 않으며 구조 결정이 어려울 수 있다는 단점이 있습니다.

* 두개는 어떤 경우에 사용하나요?

RDBMS는 데이터 구조가 명확하며 변경 될 여지가 없으며 명확한 스키마가 중요한 경우 사용하는 것이 좋습니다.

NoSQL은 정확한 데이터 구조를 알 수 없고 데이터가 수정될 수 있는 경우에 사용하는 것이 좋고 많은 데이터를 저장해야 해야 할 때 사용합니다.

* 인덱스

인덱스란 : 데이터를 빠르게 찾을수 있는걸 도와주는 요소입니다. 하지만 인덱스를 사용하면 인덱스메모리가 필요하기 때문에 더많은 메모리가 필요합니다.

클러스터링 인덱스: 인덱스로 행을 재배치합니다.테이블당 1개만 가집니다. 정렬이 이루어지기 때문에 빈번한 삽입,삭제를 하면 속도가 떨어진다.하지만 검색속도는 빠르다.

논클러스터링 인덱스:테이블당 다수의 인덱스를 생성할수 있으며 레코드 원본은 건드리지않고 인덱스 페이지만 정렬한다. 별도의 인덱스 페이지를 생성하기때문에 많은 메모리를 차지한다. 포인터이기때문에 검색속도는 클러스터링에 비해 느리지만 삽입,삭제가 빠르다.

* SQL이란 관계형 데이터베이스 관리 시스템의 데이터를 관리하기 위해 설계된 특수 목적의 프로그래밍 언어이며 관계형 데이터베이스 관리 시스템에서 자료의 검색과 관리를 위해 고안되었습니다.
* 스키마란 무엇인가요?

데이터베이스의 구조와 제약조건에 관해 전반적인 명세를 기술한 것입니다.

쉽게 말해 DB내에 어떤 구조로 데이터가 저장되는가를 나타내는 데이터베이스 구조를 스키마 라고 합니다.

* 특징에 대해 아시나요?

스키마는 데이터 사전이라고하는 저장소에 저장되며

시간에 따라 변하지 않는 특성을가집니다.

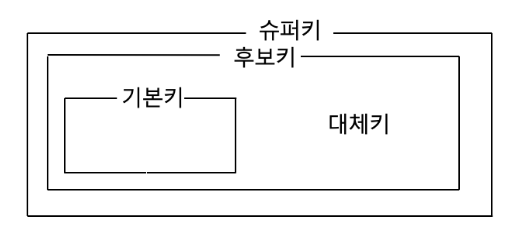
* 키의 개념에 대해 아시나요?

키는 데이터베이스에서 조건에 만족하는 튜플을 찾거나 순서대로 정렬할 때 다른 튜플들과 구별할 수 있는 유일한 기준이 되는 속성입니다.

* 최소성과 유일성에 대해 아시나요?

유일성은 하나의 키값으로 튜플을 유일하게 식별할 수 있는 성질입니다.

최소성은 키를 구성하는 속성들 중 꼭 필요한 최소한의 속성들 로만 키를 구성하는 성질입니다.



* 키의 종류에 대해 설명해 보세요

슈퍼 키, 후보 키, 기본 키, 대체 키, 외래 키가 있습니다.

슈퍼키는 유일성의 특성을 만족하고 최소성을 만족하지 않는 키입니다.

후보키는 유일성과 최소성을 만족하는 키로써 기본 키로 선정될 수 있는 후보를 의미 합니다.

기본 키는 후보 키 중 사용자가 선택한 키를 의미하며 기본키를 선택할 때는

널 값을 가질 수 있는 속성이 포함되는 후보 키, 값이 자주 변경될 수 있는 속성이 포함된 후보 키는 기본키로 설정하면 안됩니다.

대체 키는 기본 키를 대신할 수 있는 자격이 있지만 위의 이유 때문에 기본 키로 선택되지 못한 키들을 의미 합니다.

외래 키는 다른 테이블의 기본 키를 참조하는 키 입니다.

보통 릴레이션 간의 관계를 올바르게 표현하기 위해 필요합니다.

* 외래 키가 왜 있어야 하나요?

데이터가 항상 정확한 값을 유지해야 한다는 의미의 데이터 무결성 때문입니다.

예를 들어

A릴레이션의 고객 아이디가 변경되었는데

B릴레이션의 주문 고객의 값이 변경되지 않는다면 같아야 하는 값이 달라지기 때문에 데이터의 무결성이 깨지게 됩니다.

* 데이터의 무결성에 대해 자세하게 설명해 보세요

데이터의 무결성은 데이터의 정확성, 일관성, 유효성이 유지되는 것을 뜻하며 4가지 무결성이 있습니다.

1. 엔티티 무결성

모든 테이블이 기본키로 선택된 필드를 가져야 하며 기본 키로 선택된 필드는 고유한 값을 가져야 하며 빈 값은 허용하지 않는다.

1. 참조 무결성

외래 키와 관련된 규칙으로, 모든 외래 키의 값은 특정 테이블의 기본 키를 참조하거나, null 값이어야 하는 규칙입니다.

* + 외래키는 타 릴레이션과의 관계를 표현하는 역할을 하는데 외래키가 자신이 참조하는 릴레이션의 기본키와 상관없는 값을 가지게 되면 두 릴레이션의 관계를 표현할 수 없으므로 외래키의 의미가 없어집니다.

1. 도메인 무결성

필드의 무결성을 보장하기 위한 것으로 필드의 데이터가 올바르게 입력 되어야 한다는 것입니다.

* 정규화에 대해서 설명해보세요

데이터베이스의 이상 문제를 해결하기 위해 속성간의 종속 관계를 분석하여 여러 개의 릴레이션으로 분해하는 과정입니다.

* 이상 문제가 뭔가요

원하지 않는 자료가 삽입 되거나 삽입하는데 필요한 자료가 부족하여 삽입이 되지 않아 발생하는 삽입이상 문제

데이터를 삭제할 때 유지되어야 하는 정보까지 연쇄적으로 삭제되는 삭제 이상 문제

중복된 튜플 중 일부의 속성만 갱신 시킴으로써 정보의 모순성이 발생하는 갱신이상 문제가 있습니다.

* 장단점이 있나요?

방금 말한 이상 문제를 해결할 수 있고 구조 확장이 용이하고 저장공간이 최소화됩니다.

Join 연산이 많아지며 질의응답시간이 느려질 수 있다는 단점이 있다.

* 비 정규화에 대해 아시나요

하나이상의 테이블에 데이터를 중복 배치하는 기법입니다.

대량의 데이터를 빠르게 검색할 때, 조인이 지나치게 많아져 조회가 어려울 경우에 사용하며

조인 비용이 줄어 데이터 조회가 빠르다는 점, 데이터 조회 쿼리가 간단해져 버그발생 가능성이 낮다는 장점이 있지만 데이터 갱신이나 삽입비용이 높아지고 데이터를 중복 저장하므로 많은 저장공간이 필요하다는 단점이 있습니다.

* 트랜잭션에 대해 설명해 보세요

데이터베이스의 상태를 변화시키기 위해서 수행하는 작업의 단위를 뜻합니다.

트랜잭션은 총 4가지 성질을 가지고 있습니다.

* 원자성은 트랜잭션의 연산이 데이터베이스에 모두 반영되던가, 아니면 전혀 반영되지 않아야 한다는 의미로 작업을 완료하지 못했다면 지금까지 실행한 연산을 모두 취소하고 작업 전의 상태로 되돌려 원자성을 보장해야 합니다.
* 일관성은 트랜잭션이 성공적으로 수행된 후에도 데이터베이스가 일관성 있는 상태를 유지해야 함을 의미 합니다.
* 독립성은 수행 중인 트랜잭션이 완전히 완료되기 전에는 다른 트랜잭션에서 수행 결과를 참조할 수 없다는 성질입니다.
* 지속성은 성공적으로 완료된 트랜잭션의 결과는 시스템이 고장나도 영구적으로 반영되어야 한다는 성질입니다.

서버

* TCP/IP 4계층

네트워크 연결 계층 – osi 물리, 데이터링크 계층

물리적으로 데이터가 네트워크를 통해 어떻게 전송되는지를 정의하는 계층입니다.

노드간의 신뢰성 있는 데이터 전송을 담당합니다.

인터넷 계층 – osi 네트워크 계층

통신 노드 간의 IP 패킷을 전송하는 기능과 라우팅 기능을 담당합니다.

상위 계층으로부터 받은 데이터에 IP패킷 헤더를 붙여 전송

전송 계층 – osi 전송계층

프로세스간의 신뢰성 있는 데이터 전송을 담당하며 통신 노드 간의 연결을 제어합니다.

응용 계층 – osi의 세션, 표현, 응용 계층

사용자와 가장 가까운 계층으로 사용자가 소프트웨어 application과 소통할 수 있게 해줍니다.

* OSI 7계층

계층을 나눈 이유는 통신이 일어나는 과정을 단계별로 파악할 수 있기 때문

* 물리 계층

물리적으로 데이터가 네트워크를 통해 어떻게 전송되는지를 정의하는 계층입니다.

* 데이터 링크 계층

정보의 오류와 흐름을 관리하여 안전한 정보의 전달을 수행할 수 있도록 도와주는 역할을 합니다.

* 네트워크 계층

데이터를 목적지 까지 안전하고 빠르게 전달하는 기능(라우팅)역할을 합니다.

* 전송계층

양 끝단의 사용자들이 신뢰성 있는 데이터를 주고 받게 해주는 역할을 합니다.

오류 검출 및 복구, 흐름제어와 중복검사를 수행합니다.

* 세션 계층

응용 프로세스가 통신을 관리하기 위한 방법을 정의합니다.

* 표현 계층

데이터를 어떻게 표현할 지 정하는 역할을 하며 데이터의 암호화를 담당합니다.

* 응용 계층

사용자와 가장 가까운 계층으로 사용자가 소프트웨어 application과 소통할 수 있게 해줍니다.

* TCP와 UDP의 차이점에 대해 아시나요?

Tcp는 연결지향형 전송 규약으로 연결 설정 후 통신이 가능하며 데이터의 경계를 구분하지 않습니다. 또한 데이터를 재전송 할 수 있어 신뢰성 있는 데이터 전송을 보장합니다.

데이터가 손실되는 경우 재전송

Udp는 비 연결지향형 전송 규약으로 연결 설정 없이도 통신이 가능하며 데이터의 경계를 구분합니다. 또한 데이터를 재전송하지 않아 신뢰성이 떨어집니다.

* 다른 차이점은 없나요?

TCP는 1:1 통신 방식이고 UDP는 1:1 1:N N:M 통신이 가능합니다.

* IOCP는 왜 쓰셨나요?

기본적으로 Overlapped IO 방식으로 동작하며, ICOP 객체 내부의 스레드 풀을 사용하기 때문에 스레드의 생성 파괴 오버헤드를 줄이고 재사용 가능한 스레드를 효율적으로 관리하여 적은 수의 스레드로 많은 연결을 할 수 있기 입니다.

* IOCP 객체에 대해 아시나요?

IOCP는 디바이스의 입출력 완료를 통보하기 위한 포트로서, 빠른 입출력 통보와 최적

IOCP를 사용하기 위해 필요한 객체로써 커널에게 IOCP 객체를 요청하면 사용자는 핸들을 받아 IOCP 객체를 사용할 수 있게 됩니다.

커널에서 관리하며 자세히는 모르지만 DEVICE LIST라고 불리는 소켓을 관리하는 자료구조를 통해 관리합니다.

* Overlapped IO에 대해 설명해 보세요

말 그대로 입출력을 중첩시켜 처리하는 방식을 사용합니다.

하나의 스레드에서 여러 개의 IO처리를 가능하게 하고, 작업 완료에 대한 통지를 비동기로 진행하며

//1028 추가하기

* 논블로킹 io와 overlapped io 차이

논블로킹은 완료와 상관없이 우선 호출을 하고 성공, 실패 결과를 반환 받지만

오버렙드는 완료가 된 것만 통보 받은 후 처리한다는 차이점이 있습니다.

* Overlapped io와 IOCP의 차이점

Overlapped io는 작업이 완료되는 순간 이벤트나 콜백함수로 처리 하지만,

IOCP는 완료된 작업을 IOcP 큐에 넣은 후 컴플리션 객체를 이용해 사용자에게 알린 후 커널 스레드 풀링을 통해 IOCP 큐에 등록된 작업을 가져와 처리합니다.

* Ovelapped io\_pending에 대해 아시나요?

작업이 완료되지 않아 완료를 기다리는 상황을 의미 합니다.

* 네이글 알고리즘에 대해 아시나요?

TCP/IP 기반의 네트워크에서 작은 데이터를 버퍼에 모아 한 번에 전송하는 하여 전송횟수 가 줄어 들어 네트워크 효율성이 높아집니다.

* CAS에 대해 설명해 보세요

변수의 값을 변경하기 전에 기존에 가지고 있던 값이 사용자가 예상한 값과 같은 경우에만 새로운 값으로 변경하는 방법입니다.

모든 자료구조를 wait free로 구현하는 것이 가능하게 합니다.